Let's focus more on Quality Engineering and less on Testing the software

Joel Montvelisky

joel@practitest.com

Abstract

What value do we provide as testers?
The short answer is that Testing has no value in itself. The testing team provides a service to our organizations, so we (the whole organizations) can deliver value-adding products and services to our customers.

Today, due in part to the evolution of development practices and in part due to the adoption of new technologies, engineers who used to "only" test can expand their value-adding activities towards Quality Engineering in the broader sense of the word.

In this paper we will briefly review the forces pushing forward the changes in the way we create and deliver products today.

We will map the way a traditional tester might fit into this new reality and expand her activities, roles and responsibilities.

Then we will explore the value these additional responsibilities can provide to our organizations as we actively take on the role of Quality Engineers.

Bio

Joel Montvelisky is a Co-Founder and Chief Solution Architect at PractiTest.

He has been in testing and QA since 1997, working as a tester, QA Manager and Director, and a Consultant for companies in Israel, the US and the EU. Joel is also a blogger with the QA Intelligence Blog and is constantly imparting webinars on a number of testing and Quality Related topics.

Joel is also the founder and Chair of the OnlineTestConf, and he is the co-founder of the State of Testing survey and report. His latest project is the Testing 1on1 podcast with Rob Lambert, released earlier this year - https://qablog.practitest.com/podcast/

Joel is also a conference speaker, presenting in various conferences and forums worldwide, among them the Star Conferences, STPCon, JaSST, TestLeadership Conf, CAST, QA&Test, and more.

1 Introduction - Evolution

Every profession or practice finds itself in constant evolution as the experience of its practitioners, the knowledge in the field of practice, and the technology used to perform it improve and expand constantly.

This evolution is usually the result of a number of factors converging in the same place and at the same time, interacting to push towards a more efficient and effective place overall.

An interesting finding is that, in many cases, the most important of these converging factors for evolution is “need”. If there is a strong desire to become better, faster or more efficient then we will look for ways to fulfill this need - and hopefully find them.

As has been apparently mis-attributed to Plato countless times:

“Necessity is the mother of invention.”

The origins of this quote may be unknown, but its accuracy is still very real.

Software Development is a profession in constant evolution. It evolves because we continue generating more advanced tools and technology. It is pushed forward as we find better languages and processes to do our work. And yet, the most important contributing factor to its evolution is the business need to constantly come up with newer and better solutions to the everyday needs of humans in their daily lives.

If that wasn’t enough, market and competitive forces constantly challenge organizations to deliver products faster, and we respond by evolving the way we work in order to answer their needs.

This means that we are on a constant path of forced improvement. We are far from reaching a final goal, a goal that is destined to move constantly away from us.

2 From Waterfall to Agile and then to DevOps

2.1 Waterfall

Back in the 90’s the predominant development practices were based on Waterfall processes. The most common representations of them being the V and W development models.

These development models were used in tens of thousands of projects (or more), with very good results and delivering many of the most complex products and projects of the time.

There are many details and processes that form “waterfall” based practices. But here we want to focus on only a number of these characteristics of teams working under this approach, the ones that are most relevant to our topic.

These points are the following:

1. The assumption is that customers know from the very beginning of the project exactly what they want to get from the finished product and are able to articulate it on a Requirements Specification document.
2. During the actual process, each step is separated into encapsulated phases for requirements, design, programming, and testing; followed by the delivery of the product and its ongoing maintenance phase.
3. In order to advance to the next phase in the process, the previous one needs to be completed.

4. If an issue is detected on any of the previous phases (e.g. if testing finds an issue with requirements), then the team will need to repeat (at least partially) all the phases from the one where the issue was detected and corrected.
5. Internal teams (product, programming, testing, etc.) work independently. There is no collaboration between the teams other than to provide feedback from one to the other.
6. Products, once finished, are shipped to external customers and end-users. Any feedback to the team is provided back via indirect channels such as Support or Customer Services.

Looking specifically at the QA teams, most of the time the objective testing on Waterfall projects was something along the lines of:

Find all the bugs before releasing the product.

Ensuring the final product has reached the desired levels of quality and stability.

Without going into too much detail, it is obvious that both these points are problematic.

Focusing on the first part, it is impractical and highly cost ineffective to run the number of tests necessary in order to ensure "all the bugs" are found before the release of any product. This is why this is done only in products where the cost of a defect released to the field is extremely high and/or can cause the loss of human life (such as the aviation, safety and healthcare industries).

And looking at the second part of this objective, as much as we can provide our inputs (the ones of the testing team) with regards to what we expect to see in the product, it is very hard to come up with the exact criteria of what the end user actually expects to see. This is the case, because in most cases there are hundreds or thousands (or more) of them, and each user will have her or his individual expectations and ideas regarding the product. And even in those cases when we are able to come up with such a list, it will still be second guessed by the additional members of the development and product teams who have other thoughts or interests in the matter.

2.2 Agile (Iterative processes)

Iterative processes started popping up in the late 90's but they took off only in the early 2000's with the spread of Agile approaches and Scrum methodologies.

For the sake of simplicity and pragmatism, in this paper I will encapsulate all the iterative approaches under the name Agile or Agile approach given that it is the current industry practice.

Depending on who you ask there are a number of different reasons why Agile processes caught on.

The one I find most appealing is the need to include customers in the development process to provide their constant feedback on the product as it was being created.

When Agile was first introduced, we understood that it was unrealistic to assume the customer could know exactly what he wants from the beginning of the project. It is also unreasonable to believe changes in their needs can be avoided (or even fought), and here Agile processes come to factor in change and flexibility into our operating approach, and to adopt it (their need for change) as a reality.

Again here, there are many characteristics we can mention about Agile approaches that are relevant to our analysis, but the most important ones are the following:

1. Only users know what they want, and sometimes it is hard to understand their needs up-front. That is why it is best to constantly show them what we are creating for them and get their feedback on the evolving product.

2. The reason for working in small chunks or User Stories, is to get this feedback and make the necessary changes and corrections to our product as quickly as possible.
3. Our process should be simple, and be ready to fix bugs and solve issues quickly as they appear.
4. The most effective way for a team to work is integrated and focusing on constant communication, with all its members interacting and collaborating with each other in order to achieve the goals of the team faster.
5. Our products are still being "shipped" to external customers. This means that once the product is released we get feedback from indirect proxies such as Customer Support, Professional Services, and sometimes from our Product Teams.

Following a similar exercise as we did before, we can define the intrinsic objective of the testing team as:

Work together with Developers to test and deliver products quickly,

receive feedback from users, and continue the iterative development process.

There are a number of changes here from what we saw in the waterfall approach, but some of the most important are:

- The realization that bugs are not the responsibility of the tester, they belong to the team as a whole.
- The focus has been shifted (maybe too much?) from ensuring quality to delivering quickly.
- The aim is to get feedback from users in order to validate and continue moving forward

Due in part to this shift from quality to speed, when some organizations started adopting Agile methodologies such as Scrum one of the first actions they did was to get rid of all their testers. The idea was that if testing was the responsibility of the developers, then they could work without testers in the team.

In most cases, after one or two terrible release cycles, most of these teams went back to hiring testers and including them as part of their Agile teams.

2.3 DevOps

The transition to DevOps is a typical example of evolution being caused by the convergence of two separate major factors that are apparently unrelated (plus a big business push for speed running in parallel to them).

Factor 1 - Web Technologies, that allowed organizations to stop shipping products to customers using media they would need to install in their servers or even clients. The solution being to host these applications as Services in their own cloud infrastructures (e.g. AWS or Azure).

Factor 2 - Widespread adoption of Agile and the reduction of the time it took to release updated versions to customers. Or in other words, the increasing number of releases a team had to do in a specific time span.

As companies started running their own products on their web infrastructure instead of shipping CDs to customers to install them in their own environments, it became easier to encapsulate features into a number of smaller user stories and just release them at an increased rate.

This forced the development teams to create products ready to be deployed faster and easier. Meaning that the development had to understand a lot more about the deployment process and environment better.

In parallel this also forced the operations teams to learn how to cope with issues in the products being deployed faster, bringing them closer to the work of development.

With time the lines dividing the development and the operating teams blurred more and more, until we started seeing integrated teams that were in charge of both developing and operating their products - Hence DevOps teams

At first glance there is not a big difference between teams working based on good Agile processes and those working on DevOps approaches. But once you look closer there are some important differences to mention:

1. We have the same integrated teams defining and creating the features, and then deploying and running them in production. This means we can monitor the usage of our products and features directly from production, and understand if customers are happy with our products.
2. We can now detect, understand and fix problems as they happen in Production environments faster and better than ever before. This means that the cost of releasing a bug (and fixing it once it has been detected in the field) has been greatly reduced. So much so, that sometimes it makes more sense to skip parts of the internal testing process, if the potential bugs we would find in them are cheaper to fix only if they materialize in the production environment - or as it is known Testing in Production.

What is the objective of Testing in DevOps teams? In a sense it is similar to that of Agile but with some important variations:

Work with "Dev" to release quickly, enabling stability on the deployment process,

engineering fast feedback from production.

Notice that the focus here, similar to Agile, is to release quickly by working with Dev, but a lot more emphasis is placed on stability of the process (not only of the product), and maybe the most important change is the focus on providing fast feedback from production.

This is the feedback that will be used in order to either fix issues quickly in production or to continue improving the product in its future iterations.

3 An "incomplete" but important introduction to Modern Testing

Evolution of processes happens when people start thinking, and many times talking, about how to make something better.

In the case of Testing and Quality Assurance, and specifically around the context of Agile & DevOps approaches, one of the best examples we can see is in the emergence of "Modern Testing" and its growing community.

Modern Testing (or MT for short) is a concept initially developed and still currently led by Alan Page and Brent Jensen as part of their AB Testing podcast.

The ideas behind MT have also spread, and are being constantly reviewed and commented on by a significant and increasing number of testers from around the world that have come together to form a community around this new approach to Quality in general.

You can read a lot more about this approach on the official MT website - http://www.moderntesting.org/, where you can also find links to the AB Testing podcast mentioned above (highly recommended) as well as to the community itself.

I am not going to do a deep review of MT in this paper but I still want to mention two very important aspects of it, its mission statement and its seven principles.

The Modern Testing Mission Statement is the following:

Accelerate the Achievement of Shippable Quality.

The statement is both pragmatic and vague at the same time.

The focus is on accelerating the process. The goal is Quality (in the context of the process around releasing or "shipping"). Finally, the word "testing" is totally absent.

MT has seven guiding principles to help practitioners understand the most important aspects of this approach. These principles are the following:

1. Our priority is improving the business.
2. We accelerate the team, and use models like Lean Thinking and the Theory of Constraints to help identify, prioritize and mitigate bottlenecks from the system.
3. We are a force for continuous improvement, helping the team adapt and optimize in order to succeed, rather than providing a safety net to catch failures.
4. We care deeply about the quality culture of our team, and we coach, lead, and nurture the team towards a more mature quality culture.
5. We believe that the customer is the only one capable to judge and evaluate the quality of our product.
6. We use data extensively to deeply understand customer usage and then close the gaps between product hypotheses and business impact.
7. We expand testing abilities and knowhow across the team; understanding that this may reduce (or eliminate) the need for a dedicated testing specialist.

One of the most important aspects we can take away as Test Engineers from MT, is that we are not in charge solely of testing. The only way to improve the process is to change the way we approach quality, understanding that (a) testing needs to be done by every member of the team, and (b) there are many more additional quality-related operations and actions that require our urgent focus and attention.

Testing being the responsibility of every member of the development team is a point that has been reviewed by many people and papers, and so we will only review this "in passing" on some of the coming sections.

We will go more in-depth on additional quality related tasks requiring our attention and focus within the work of our development teams.

4 Focusing on Quality at the beginning of the process

There is a cliché that we need to "Shift Left", meaning to take part in processes that occur before the traditional (at least on Waterfall terms) testing phase. I prefer to call this focusing on Quality at the beginning of the development process.

There are a number of actions we can do, or actually need to do, as Quality Professionals that fall under this category.

4.1 Bring customers into the Process

As mentioned above, one of the most pragmatic business motivations for introducing Agile practices was to bring the customer's feedback earlier into the process, resulting in a better and more accurate product without delaying the release schedule.

The problem with this is that, when you bring customers into the process they tend to make a lot of "noise" that disrupts the developers during their work, and so for most organizations this is more of a "theoretical idea" than an actual practice.

We know that one of the traditional tasks of testers has always been to represent the customer within the development process. We can expand this charter into the more concrete and pragmatic task of bringing as much of the voice of our customers into our team's development process - without actually bringing the customer on-site.

Here are a couple of examples of things we can do, as Quality Engineers, to bring more of our "customer's voice" into the development process.

4.1.1 Visit customers to get their feedback first hand.

This allows us to understand our users better, see how they work with our products in their environments, get more realistic inputs, and ask questions that we find relevant.

An additional bi-product of this activity is that it allows us to follow up with these same customers we visited later on, in case we have additional questions during our work on the development stages of the product.

I have been doing this for over 15 years, with great results for all my teams.

4.1.2 Bring real data / applications / configurations / etc.

Many times, if we cannot get real users to come to us, we can bring real data to help us understand more about how people are working with our products, and also allow us to test them closer to the way they will be used once we release them "into the wild".

If you are visiting customers in person, as mentioned in the point above, then it is relatively simple to ask for configuration files, or copies of environments, etc. But even if you do not visit them, there are ways to get this information.

Just as an example, some 15 years ago I was working for a company where we had a large Enterprise Product. A big issue with the release of this product was the upgrade of the database backend with every new version. The system was very complex and the upgrade would break if customers did any modifications or customizations to the database (DB) schema - and many customers did a lot of customizations to their DB schemas...

Each time a customer had an issue with the upgrade, they would contact our Customer Support team for help. Support would take their DB and try to upgrade it. When this failed they solved the issue, and if needed they would contact our Dev team for assistance. At the end, they would return the migrated DB to the customer.

After a while, we came up with an idea. We would ask our support team to request permission to keep a copy of the DB in store, so that we could test our upgrade process on it during the next development cycle. This would assure the users that their project would be successfully upgraded and it would give us real data for our testing.

About half the customers agreed, and within some months we had an automatic system running once a week, migrating tens of real databases from real customers and fixing bugs in the upgrade process as they were discovered.

Within two releases we cut DB migration issues by close to 80%.

4.1.3 Create “Personas” for your team

Persona or customer profiles are a great tool both for the design process, as well as for the testing or development operations of our teams.

If we have a descriptive and complete profile of our users, that includes things such as Age, Experience, Constraints, etc. this will guide us while designing our solutions and in case we need to make decisions regardless of things such as bug severities, prioritization of features, etc.

Personas are also a great tool when you need to create test scenarios, as they help you to envision the way users will interact with our solutions based on their personal characteristics and even constraints.

4.2 User story validation

Are we creating the right product? Focusing on the right feature? Moving forward while including the feedback needed for the process?

There are many points we can review in order to ensure our team’s efforts are focused on the right path, and it all boils down to ensuring our user stories are correct.

4.2.1 Capture inputs from other departments

We have to make sure to get as much information to define the user story from places such as Customer Support, Professional Services, Customer Success, Documentation, etc. This will ensure the story is complete and also that it will not be delayed later on because we forgot to cover an important aspect. It will also help us to detect conflicts and inconsistencies with other areas of the product

As testers we are already used to looking at the big picture, encompassing all the application and not only the narrow area we are currently working on. This gives us an added perspective and the ability to find places where the user story may be in conflict with functionality already available in the product.

4.2.2 Generate an MVP - Minimum Viable Product

It is very easy to get carried away, and try and solve the whole problem in “one step”. But the idea of an iterative process is to start small, validate, and then continue developing once we know we are on the right path.

One of the tasks we have as Quality Professionals is to ensure we follow this rule and limit the scope of our user stories to the minimum set of features (and value to our users) to get this validation.

4.2.3 Define success criteria and the instrumentation to measure it

We are looking to get validation on the functionality we are creating, but how do we actually know if the users liked it or not? When you are working on products with tens or hundreds of thousands of users it is not a matter of asking them for their individual feedback.

Instrumentation will help us to understand what is happening to the functionality in production when working on larger scale applications.

Are people using the feature? What parts are they using and/or not using?

If there is an error or an exception, do we have enough information in order to understand it and fix it quickly?

We need to think about everything we will want and need to know once the feature is released, and ensure we have the ability (within the code!) to know this.

If we do not have requirements for instrumentation as part of our user stories then we will be blinded once the feature is pushed out to production.

5 Quality Engineering towards the end of the process

There are also a number of actions we can take that fall either around the traditional (again, based on Waterfall terms) testing phase or in the phases that follow it within the DevOps process.

5.1 Coaching and Training on how to test

Even if your company “decided” that everyone in the development team will test, this does not mean that tomorrow all your Devs will be able to test (even if they want to). It is imperative that you, as the Quality and Testing Specialist come up with a plan that will train and coach your developers on how to test correctly.

There are a number of ideas that will help you to do this in a pragmatic and effective way.

5.1.1 New Dev Training program.

Most organizations have an on-boarding process already defined, you need to make sure that the onboarding process for developers includes at least a couple of sections on testing.

Look for things such as testing techniques and heuristics, as well as how to set up environments, interview relevant people, come up with testing ideas, document tests and bugs, etc.

5.1.2 Pair testing sessions with developers and testers

I find this is the most efficient way to teach developers what testing is all about.

Most times developers are not even aware about the process of testing or how we, the testers, go about reviewing the system and thinking about additional scenarios to run.

A good and proven approach to pair testing is to start by pairing a Dev with an experienced Tester. Give the tester the lead during the first session, then on the second or third paired session the team can switch places, so that the tester can mainly provide feedback to the dev while she does most of the actual testing work.

It is also good to change pairs, so that both Devs and Testers can get the perspective and knowledge of others within their teams.

5.1.3 Test Process Definition

Having a defined process helps people to know what is expected of them, and what steps they need to follow. On the other hand, if your process is too heavy and exhausting, then most people will simply choose to ignore it.

Having a testing process that is specifically defined for Agile Teams is important, make sure it is based on communication between individuals, on documenting ONLY what is needed, and in having open collaboration between the team members.

5.1.4 Testing Briefings and Debriefings

These practices are important to all testing operations, and they are critical when doing development testing.

You want to make sure Developers have all the information they need before planning and especially before they run their tests. Most times the information on the User Story is only a small part of what we need, so it makes sense to take 5 - 10 minutes before the actual testing operations to ensure we have everything we need (knowledge, plans, tools, data, etc.)

Once we are done with our testing, it is important to take 5 minutes to review our work and findings, with the objective of verifying that all the important parts of the functionality were covered by our tests. This is done based on the previous knowledge or data, and also in light of the findings of the test itself.

5.2 Test enablement for Developers

In addition to training and coaching, we can operate in a way that will enable developers, testers and other non-testers to test more efficiently.

5.2.1 Create “Testing Cookbooks”

Cookbooks are informal step by step guides that help people who are not familiar or experienced in testing techniques to perform their testing operations.

They should include all the steps to plan, run and document their testing. They can also be created for different types of testing or areas of the product - with the idea of making them short, focused and easy to follow.

5.2.2 Develop Different Testing Artifacts

Different people will approach testing differently, and for this they may prefer different testing artifacts to guide their manual testing operations.

Some people will like detailed testing scripts, others will prefer Exploratory Charters, while some may even prefer checklists and heuristics, etc.

In the case of Developers my experience is that they are good at working with checklists and heuristics, and less good with Exploratory Charters.

In any case it is good to have artifacts to suit the needs of the different testers in your team.

5.2.3 Test Environments

How many times have you heard the phrase “it works on my machine”?

We all know that the environment and data have a lot to do with the results of your tests. The more realistic and “less clean” the testing environment, the higher the chances you will run into the kind of bugs that our users run into “in production”.

It is unrealistic to think developers will take the time to create their own testing environments that include good data, realistic set ups, etc.

It is better to have a number of these environments ready, and to allow anyone in the team to use them whenever they need to.

With the use of containers such as Docker, Kubernetes, etc, this is even simpler and easier to achieve.

5.3 Deployment and Release process

Many think that deployment and releases should be the task of the Ops team and not the testing team, but they are missing a big point here - Deployment process, releases, and specially rollbacks need to be planned, developed and tested carefully and thoroughly, just like any other aspect of the product.

5.3.1 Release risk management.

Every release is different, and they do not have the same level and type of risks. In many organizations the Test Manager doubles as the team Risk Manager, and in this case the Testing Specialist can also work as the Risk specialist of your deployments.

Do we truly understand what could go wrong? Do we have contingencies? Is this the only way or the best way to do this? Someone needs to ask these questions and get proper answers.

5.3.2 Staged processes

Does your team have different environments where to test the system? Is at least one of these environments realistic enough in hardware and software?

It is important to ensure all testing has been done, and enough of it in an environment that resembles the production system. It is not enough to have realistic data (as was covered in a previous section) if the machinery is radically different from the one we will work in production.

5.3.3 Deployment & rollback testing

We test how to work with the system, but many times we also need to test how we are going to deploy the system and what our behavior will be if we find something is wrong.

Rollback testing is something that is usually done in theory. "Yes, if needed then we will rollback the DB, bring a backup, and redeploy the production code to the previous version..." But what about the dependencies? Are there any other "surprises" that will only show themselves when we actually start running this rollback process in the real environment.

These are usually the times when you will need to work fast, and avoid thinking as much as possible, and so it makes sense to try and document all the steps needed, as well as rehearse constantly to ensure you know what to expect and how to behave in every step of the process.

5.3.4 Scheduling and notifications

Every hosted system undergoes maintenance and has scheduled downtime. When this happens, we need to make sure our users are aware of this, and can prepare accordingly.

Ensure the defined notification procedures are followed and that scheduling has been done according to the best available windows.

Especially when you are working on Software as a Service, we need to remember that a big part of this service is around communication with our users.

5.4 Production analytics

We talked about the need to monitor our products in order to ensure they are fulfilling the demands of our customers. In addition to that, we also want to have monitoring in place for the more menial objective of making sure our service is operating at the correct level and avoiding any outages.

5.4.1 Instrumentation planning and testing

Most of the time a system "falls" only after a steady process of deterioration. Resource availability decreases over time, response time starts to increase, one of the subsystems gets stuck, then another

and another until the whole service falls.

Correct instrumentation will ensure timely notifications are given when there are still actions to be done and time to react.

What instrumentation to develop into the product? What deterioration levels should trigger alerts? All these are aspects to be analyzed, planned, reviewed, refined and constantly improved.

Instrumentation and alerts also need to be tested, especially since we will only notice their absence when it is already too late to do something about it.

5.4.2 Feature validation analytics

As previously mentioned, analytics and instrumentation need to grow as our product does. Every important new feature should include instrumentation, both for product health as well as for user adoption.

5.4.3 Dashboards and alerts

If you want people to improve something then it helps to measure this, and to share these metrics with as many people in the team as possible.

This is true about product monitoring and performance analytics as well.

Make sure the information is not buried in a dark and forgotten log, but displayed in bright colors and simple letters so that everyone can see it and understand what is going on with the system.

6 Conclusion – Quality Assurance work that generates Value

I mentioned “in passing” that one of the initial by-products of the implementation of Agile in some organizations was the dismissal of their whole testing teams. This was followed by an almost Universal reverse process of re-hiring testers to be part of the Agile teams and either perform or lead the testing process internally.

When talking with a number of people who worked in these teams, many of them mentioned that one of the reasons for these actions was that the value of the QA team was perceived to be limited only to the actual testing.

This concept of “perceived value” is an important one, since it will guide not only the way others see the work of the tester, but also the opportunities he or she has to take an active role in additional tasks of the team.

Perceived value is very hard to measure or pin-point, and so we all just disregard it as something that is not really important to our everyday work. This could not be further from the truth.

We need to expand our perceived value, as this will be one of the tools we will use in order to expand our charters. Still, how can we get a better understanding of it, given that it is hard to define?

I have been working on this question with a number of colleagues in the field, and I was able to come up with a set of questions that will help you and your team understand what is the concrete value you provide to your organization.

These questions are the following:

- What decisions do you help to make in your team and process? How?
- How do you change and improve the process by which your company delivers products?
- If you and your team were not there, what would be different?

By answering these questions together with the non-testing members of your organization, you will be able to get a better understanding of your perceived value. Then you can use this information in order to plan how to expand your work into the additional areas you see fit.

References

AB Testing Podcast - https://www.angryweasel.com/ABTesting/

Modern Testing - http://www.moderntesting.org/

State of Testing Survey & Report program - https://qablog.practitest.com/state-of-testing/