Testing Strategies in an Agile Context

Zornitsa Nikolova

Abstract Testing in an Agile context is extremely important, not only with its function to ensure quality but also to guide development efforts into the right direction. This is related to a shift in testing paradigms, with quality being viewed as a factor early on in product development, rather than a late-stage reactive activity. It also requires the application of different approaches, such as automation, to enable the flow of potentially shippable product increments. Many teams find themselves stuck into the old ways of testing, especially as they work on legacy systems. However, investment in upskilling quality experts, applying the proper tools, and changing the way testing is done can bring tremendous value and opportunities for innovation. Proper change management needs to be applied to enable teams transition successfully into an Agile mindset and new practices.

Keywords Software testing · Software quality · Agile testing · Test automation

1 Introduction

Irrespective of the methodology that you use for managing your product development, quality assurance is important. And when we say “quality assurance,” we typically mean “testing.” Agile contexts are not different in this sense. In Agile, we value “working product more than comprehensive documentation,”¹ hence the majority of teams’ efforts shall be focused on creating software and making sure it does what it is supposed to do.

Yet, in many cases when we look at processes in various teams, even if they claim they are Agile, we still see quality as just the final step of creating the product. We design, we build, and then we test, usually with a focus on discovering bugs. Even though we talk about team ownership on results, it’s still a quality expert mainly

¹Manifesto for Agile Software Development, www.agilemanifesto.org

Z. Nikolova
Leanify, Sofia, Bulgaria

responsible for testing, reporting problems, and making sure Definition of Done is satisfied with respect to quality standards. Imagine a situation: it's the end of the sprint, most stories are finished from a development perspective, and they are in the QA column on the board. Teams' QA experts are working around the clock to make sure they have checked new features thoroughly. Yet, at the Sprint demo they still cannot report full success—developers finished a couple of more stories in the final day of the Sprint, and they couldn't complete testing . . . Sprint has not been entirely successful, and it is a vicious circle. Does it sound familiar?

Unfortunately, I still see the above situation way too often, even though we claim that Agile approaches are by now mainstream. Believe it or not, this way of working stands in the way of true Agile adoption in teams, and requires a certain change of paradigms, so that we can benefit from Agile software development.

2 The Shift in Testing Paradigms

Situations like the one described above happen often when the switch to Agile practices focuses primarily on the question “what methodology shall we apply?” Do we want to do Scrum, Kanban, Scrumban, or something else? While I believe it is an important question, I do not think focusing too much on it really helps us understand and adopt Agile practices. Frameworks and methods, such as Scrum and Kanban, are there to support teams achieve a certain goal. So, defining the goal, the purpose of applying Agile practices, is the first thing to do.

According to the latest State of Agile report from Version One [1], among the key reasons for adopting Agile are the need for faster development as well as enhanced software quality. Yet, in many cases, creeping technical debt and a lot of rework prevail, partially caused by changing requirements, but also, to an extent, by defects found late in development.

Teams that are successful in addressing such challenges apply a different way of thinking about testing, which is illustrated by the concept of Agile testing quadrants [2] (or the Agile testing matrix [3]) (Fig. 1).

The quadrants imply several important aspects of a change in thinking about testing and quality in general. First of all, we shall not think of testing only as a means to discover bugs—this is a very reactive and limiting view on the quality process. A much more empowering view on testing suggests that it has two faces—one focused on product critique (finding functional bugs, unwanted behaviors, performance, security, and other nonfunctional flaws) and another focused on supporting the team to take the right decisions upfront, by doing frequent small tests on unit, component, and feature level (the left side of the matrix). This second aspect of testing is largely underutilized though, especially in teams that transition to Agile from other paradigms. Traditionally, we are used to testing for bugs, and this is the common profile for a quality expert. Eventually, we end up with a lot of back loops from testers to developers for fixing issues that can easily be prevented

Fig. 1 Agile testing matrix

with upfront investment in tests that check how good requirements are defined, what is the level of understanding, and how we can design our solution in an efficient way.

Secondly, we shall extend our scope of thinking about quality as something related to the technology side of the product. When we create software, we take decisions related to the technical design, architecture, and implementation, but a good portion of decisions are related to the business side of the product as well. How do we check if those decisions are correct? Previously, we would invest a lot of resources to build a complete product and only then launch it for a market test. This used to work for big companies (sometimes) but in the world of startups with limited resources it is not a viable approach. That is why startups naturally adopt Agile practices, including smart ways to test the business aspects of the product as early as possible, through prototypes (often low fidelity) and simulations. Business-oriented testing checks whether we are building the right product from a user and market perspective, while technology-oriented testing checks whether we have built it right.

Finally, an implication from the matrix is that testing is not only a QA expert’s job. The various types of testing require various expertise, and contribution by everyone—even customers and users themselves. So, it is in the collective ownership and responsibility of the entire team to embrace the different aspects of quality and engage in activities that contribute to the different types of testing. Quality is no longer a one-man’s quest for perfection—it is a team effort to ensure meaningful investment and strong product performance.

3 Investment in Automated Testing

The Agile testing quadrants offer a great thinking model around the aspects of testing that we need to address and master, so that we can have good quality on both business and technology level. However, it is also obvious that doing all those

tests manually is hardly possible, especially if we are aiming for frequent and quick feedback and minimized waste of waiting time. Naturally, the paradigm of Agile testing involves moving from manual to automated testing.

It is easier said than done though, unfortunately. In reality, many teams are faced with a huge legacy of inherited code, written for over 10+ years, and productively used by real customers. Often, to automate testing for such systems requires certain refactoring, which on the other hand is quite risky when unit tests are missing. It is a Catch 22 situation. Moreover, in some cases systems are written in proprietary languages (take SAP's ABAP, for example) that lack proper open-source tools and infrastructure for test automation. Investing a big effort in automation might be a good idea from a purely engineering viewpoint, but it might be hard to justify from a return-on-investment perspective. Doesn't it sound familiar? The constant fight between the team and the Product Owner on how much we shall invest in removing technical debt!

When planning our strategies for automated testing we need to consider a few aspects that might play a role in this decision-making exercise. First of all, it is important to acknowledge where our product is in terms of product lifecycle (Fig. 2).

The graphic represents the standard concept of product lifecycle with respect to its market penetration, applied to software products. In this context, there are slight differences as compared to physical products. First of all, with software products, especially following the ideas of the Lean startup and Agile business-oriented testing, we might see much earlier exposure to the market—already in the Conception and Creation phase. This means that we need to think about quality aspects quite early, as technical debt tends to build up in these early stages of software development and this leads to impediments in the growth stage. At the

Fig. 2 Software product lifecycle

Fig. 3 BCG matrix

same time, a mature product might go back to growth if we decide to invest in sustaining innovation (e.g., to extend its scope and cover a new market segment).

When talking about legacy systems, we shall consider first where they are in terms of lifecycle phase, and to what extent we plan to develop them further (either as part of their natural growth or through sustaining innovation). Any investment in further growth shall be accompanied by an investment that supports development, that is—investment in applying Agile testing and test automation is essential.

Similarly, we can look at our strategy for investment in automation and cleaning up technical debt using the Boston Consulting Group (BCG) matrix (Fig. 3).

Looking at products from the perspective of current and potential value to the business gives an important input when we try to estimate return on investment. Note that in this case we are looking at a return in the midterm rather than short term, as creating the infrastructure and building up automation from scratch is not a short-term task either. So, we can generally follow some of the strategies suggested in the figure. For “cash cows”—the products that are currently in a mature phase, yielding return from previous investments but also not likely to grow significantly in future—undertaking significant investment is not recommended. We might need to optimize to some extent, so that we can improve operational maintenance (e.g., by automating regression testing partially), but shall be conservative when it comes to big effort for automation. On the other hand, for our “stars”—products that are potentially in a growth phase and strategic for business—we might even want to consider a “stop-and-fix” effort. The sooner we invest in building up a solid infrastructure that enables us to continue development with the support of automated testing, the more stable velocity of development we can maintain overtime. Then for “question marks” we are in a position to prevent the buildup of technical debt in general. This means

Fig. 4 Agile testing pyramid

making sure that automation is part of the standard Definition of Done and the effort is planned accordingly by the team in each Sprint.

The product lifecycle and BCG matrix offer a more business-oriented view on the question of investment in automation. Now let's look at the technical perspective. Mike Cohn's testing pyramid [4] offers a nice visualization of the system layers where test automation shall be considered, and to what extent (Fig. 4).

In our traditional way of working, most of the testing is done manually, and it typically requires access through a UI layer. This means that testing happens quite late, and it requires significant development effort upfront, hence potentially a lot of defects piling up and being discovered quite late when rework is more costly. As discussed in the previous section, the focus is on finding bugs and product critique, and this is an expensive way to address quality. No wonder that it is often compromised, especially when we are late with deadlines and there is pressure to deliver. Not to mention that manual testing is also much slower, of course, and this makes things even worse.

In the Agile paradigm, we need to reverse the pyramid, as shown on the right side of the picture. The biggest effort for automation is done on the unit test level. This is where ongoing development gets immediate feedback and bugs are quickly removed as part of the regular development process. On the next layer, we can automate acceptance tests based on cross-unit and cross-component functional calls within a certain use case or user story, but not necessarily involving the user interface. This integration testing is a perfect way to ensure working increments during sprints. The highest layer is testing end-to-end scenarios via the UI of the system.

Naturally, the cost of automation raises as we go up the pyramid—automating on the UI layer is typically resource consuming, and automated tests are hard to

maintain and update. Therefore, we shall spend most effort on the lower layers, automating on a unit and service level, while still doing manual testing on the UI layer. Note, however, that we can optimize manual testing strategies as well to get the biggest value of the effort spent there. Coming to the manual UI-based tests, we don't need to do full-blown regression testing, or cover all end-to-end scenarios, as we have already covered them on the integration tests layer. Here, the focus is more on nonfunctional (performance, security, usability, accessibility, etc.) testing, as well as simulation of real-life usage through exploratory and user acceptance testing.

To summarize, major investment in automation makes sense for products that are still in growth and innovation mode, and it is required for a long-term success. We shall also be selective on the amount of investment per system layer to gain maximum returns—investing in automation of unit and integration tests is typically advisable, as it speeds up development. When it comes to UI testing, we might consider automating some of the manually done smoke and regression tests, while taking into account the ongoing test maintenance effort and picking appropriate tools.

4 Transitioning to Agile Testing

Even when the team and the organization is convinced in the benefits of Agile testing, including investment in test automation, getting started on it might be another hard task. There are a lot of challenges to changing the entire process of how you plan and execute tests—from purely infrastructural (tools, test system design, etc.) through skills in the team to create and execute those tests to mindset changes that need to happen, and fears that need to be overcome. Starting from point zero is scary and not easy at all, and many teams might find themselves at a loss as to where they should start.

I am a strong believer in goal-oriented thinking, and systems such as OKR (Objectives and Key Results). Starting with the end goal in mind creates focus, motivation, and resourcefulness in people to overcome challenges as they go. So, defining our short- and midterm objectives is an excellent way to kick off a transformation of quality assurance in the organization. Of course, as in any goal-setting process, being unrealistically ambitious might fire back at some point, creating a sense of disbelief and demotivation in the team. We have to choose targets carefully, ideally in collaboration with the team.

A good practice that I have experienced personally might be to get a team of early adopters on board, train them, and get them support from an Agile coach with knowhow in Agile testing paradigms. This team becomes the catalyst for subsequent activities within the individual Agile teams as well. Note that you can have the Scrum Masters coaching teams in Agile testing practices, but if the Scrum Master is not a technical person, he or she might not be the most appropriate one to assume this role. At this point, the most important thing would be that the person is enthusiastic to learn and work to implement and improve these practices within the team. Once

we have the catalysts, they can initiate discussions in the Agile teams and create a baseline for starting the transformation. They will need to assess what they are already doing and suggest what would be the next important milestone or objective that the team needs to strive for in the midterm (the next 1 year, for example). From there backwards, they can then define reasonable objectives for the short term (let's say next 3 months). Here is an example of how this might look like.

Objective: Enable automated testing for key integration scenarios in the newly developed modules of product XYZ within Q1'2020

Key results: 80% coverage with unit tests for newly created classes; full coverage of priority 1 integration scenarios as defined by Product Owner

This is a close-to-accurate quotation of an objective that we set in a team I used to work with. They were developing a new add-in product on top of a legacy system that did not offer a very favorable environment even for starting with unit testing. When we started talking about applying Agile testing concepts to new development, the reaction of people was: "This is totally impossible. It would mean that we double or triple the development effort if we also have to write automated unit or integration tests." Still, they were somehow convinced to give it a try (the pain they felt each time they had to touch anything in the legacy was a strong factor in making them more open to experiment). We had to start small and check if we could make it happen at all at a reasonable cost. So, we started with new development only, focusing to cover new classes with unit tests, and key integration scenarios with service-level integration tests. We did not do automated UI testing at this point—manual UI-level tests were run as usual, just to create a baseline for us for checking the effect of automated integration tests as well. It took several sprints before we could really feel the difference. However, in the moment when we started iterating on features developed a couple of sprints earlier based on the user feedback that we got, the benefits of automated integrations tests became very obvious. There was no need to convince anybody anymore—developers happily planned tasks for creating more automated tests in their sprint backlog. A release later we started extending our strategy to cover with automated unit and integration tests also those legacy parts that we had to touch and rework as part of the new development efforts. Essentially, we were doing continuous innovation, and it made sense to start investing also in covering the old pieces with good testing.

Along with setting objectives and measurable results, we also decided to experiment with techniques such as TDD (test-driven development). It was not an easy mindset shift for developers either, but over time they appreciated the added focus on simplicity that TDD drives. We could experience quality improvement in system design, and gradually see a reduction of defects that were discovered at later stages. On the level of business requirements, we introduced BDD (behavior-driven development) or "specification by example." This was a great way to formulate requirements in a way that enabled straightforward creation of integration test cases as well. All of this eventually had a significant impact on both business- and technology-facing aspects of the product and was a good way to minimize

effort spent on documentation (such as requirements and test cases) by creating lean executable specifications.

Regarding UI testing, we intentionally limited the scope of automated testing. We researched a few different tools to see how we can automate part of the tedious regression testing that was done repeatedly before each new release. Our experience showed that tools tend to cover one of two scenarios. In the first scenario, tests are very easy to create by simply recording the manual test case, and then replaying it. However, if any of the screen components change (and this is often the case when using certain development frameworks that rebuild components, changing their IDs at each system build), the effort to adapt them is quite high. In the second scenario, tools allow for better componentization and identification of screen components, which makes initial effort high but leads to less difficult adaptation in case of changes. In our case, we picked a tool that supported the first scenario, and started automating only basic regression tests on critical user journeys to make sure we can quickly cover high-priority testing needs. In addition to that, we engaged in much more Quadrant 2 testing using low- and high-fidelity prototypes to run early tests with real users. This reduced significantly the need to rework UIs later in development, and minimized the effort to update UI tests as well.

The example I shared, in combination with the concepts discussed in the previous sections, might give you a few ideas as to how to start applying some of the Agile testing concepts. When you map your own transformation strategy, however, write it down, start measuring against the KPIs you have defined, inspect, and adapt on a regular basis to make sure that your goals are achievable and you are getting the results that you expect.

5 The People Perspective

Finally, I would like to draw attention to another important aspect as well—creating the appropriate environment for teams and individuals to feel safe in the transition and to effectively achieve an Agile mindset shift. No matter what type of change you are undertaking, having people on board, feeling appreciated and valued, and engaging them in the process is among the key factors to success.

This perspective is a complex one as well. First of all, let's look at teams as a whole. As we discussed in the beginning, in an Agile context quality is a team responsibility. This means that we need to support the building of this collective ownership by coaching teams into self-organization and focus on common results rather than individual achievements. It might require changing the entire performance management approach that we apply in the organization, switching from individual to team goals and product-related metrics—and many Agile organizations do that. We might also need to rethink the KPIs that we use. I was recently discussing the topic of Agile testing with a quality engineer, and he shared that their work is being evaluated by the number of bugs found. What kind of behaviors and thinking does a KPI like that support? What would be the motivation

of people on this team to invest in preventing bugs rather than discovering them late?

In addition to goals, roles and responsibilities in the teams might need to shift, also creating demand to people to extend their expertise into what we call the T-shaped profile (people with deep expertise in a specific topic, such as testing or development, for example, and complementary skills in adjacent topics—development, UX, etc.). This will enable teams to be more flexible in handling the collective responsibility on quality and will strengthen the communication and understanding between team members. It is not a process that goes overnight, however. It requires some space for teams to experiment and for team members to learn (potentially by failing in a controlled way). Managers might need to step back and let teams reshuffle tasks, and individuals cross the borders of their specific function, so that they can learn.

This last point leads us to the individual perspective as well. In most organizations, people are hired to fit a certain job description with clearly defined expectations and responsibilities. With the concept of self-organizing teams and quality as common responsibility, the role of a tester or even quality engineer might alter significantly or even become obsolete. Naturally, this creates fear and uncertainty and leads to additional resistance to change. The role of managers in such an environment is to balance these fears, provide support and guidance, so that team members who need to refocus will be able to do it, see opportunities for personal growth, and continue adding value to the team. Managers need to ensure those individuals have the resources and environment to learn new skills and can see how their role can change to fit the altering needs of the team as well. It is important to note that quality engineers often have a unique view on the product that enables them to play a very important role in activities related to product discovery, user journey mapping, acceptance criteria definition, identifying scope of prototypes and simulations, and test automation, of course.

Looking at the topic from a broader perspective, how we ensure quality in an Agile context is a very significant part of doing Agile and being Agile. It involves the entire team and requires appropriate thinking, skills, and focus. Picking the right strategy would depend on the maturity level of the organization, and the will of people inside to replace traditional approaches with new ones—and it is related to the value that we expect to get from the change on a team and individual level.

6 Conclusion

In this chapter, I have offered a broad view on Agile testing and quality as a process that underlies the success of the product from both business and technology perspectives. I believe the main value of Agile approaches comes from the fact that they do not put a strong demarcation line between technical and nontechnical roles and responsibilities in the team. On the contrary, Agile thinking involves development of customer-centric mindset and understanding of the business domain

in the entire team, while also bringing nontechnical team members on board in tech-related discussions. In well-working Agile teams, this creates an environment of collaboration, common ownership on outcomes, and joint care on all aspects of quality that we have looked at.

Agile practices have been evolving significantly for 25+ years now, and we can leverage what multiple teams and practitioners have achieved through empirical learning. Yet, in most cases, we need to do our own piece of learning as well, mapping our strategies, experimenting and adapting as we go.

As a summary, here are some key takeaway points that we discussed:

- Testing in an Agile context is very important to ensure that we are both building the right product and building it right.
- Agile thinking involves a shift in testing paradigms as well, shifting it left as early in the development process as possible.
- We need to engage different practices and the whole team to be successful.
- For traditional organization, and even some Agile teams, this requires a significant transformation that takes time and needs to be properly managed.

References

1. 13th Annual State of Agile Report. <https://www.stateofagile.com/#ufh-c-473508-state-of-agile-report>. Accessed 23 June 2019
2. Crispin, L., Gregory, J.: Agile Testing. A Practical Guide for Testers and Agile Teams. Addison-Wesley, Toronto (2009)
3. Brian Marick's blog. <http://exampler.com/>. Accessed 23 June 2019
4. Cohn, M.: Succeeding with Agile. Addison-Wesley Professional, Upper Saddle River, NJ (2009)

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.