Software quality without testing Part 1

When I started to write this article, I thought of a transcript of the talk we held at We are Developers World Congress 2019. Half a day later, and five pages into this article, I thought to myself, no one will ever read this much text. So, I decided to split that article into a series of three.

In Part one, we cover Quality in general what we actually mean with software quality without testing, we talk about what in our experience is the truth about quality and what quality is for us therefor we go into quality characteristics and the definition of quality.
In Part two we'll cover the responsibilities of each team member, where should quality for each member start and where should it end (or should it end at all?) We introduce you to the quality hopper, which is a visual aid of how quality could be structured. And finally, we look into categories of projects and how to select the right project for your needs.
In Part three we cover how to setup a quality friendly environment in which it is possible to deliver quality before we move into testing the software.

Software quality without testing? Possible!?

Before we dive into let me just say the title of that article series is a clickbait, right from the start I want to apologize for the confusion. There is no such thing as software without testing. It would be the same as if we would talk about implementing software without code, which is simply not possible. Thinking about quality means for most, thinking about testing. Either low-level tests like unit- and integration-tests or high-level tests like (user)acceptance-test UI- or regression-tests. But quality should be and is more than just testing your software. Quality should be an organisation wide unified mindset; each role and each member of that role should have a quality first attitude even if their perspectives and opinion shift a little.

However, the truth about quality is something else entirely.

The truth about Quality

As we work for different clients in different projects and various industries, we often hear the same "problems" with applying quality. Just to name a view we collected some, generalized them and give you our thoughts to these quotes.

> We ensure quality because we provide automated tests.

You test the software if an expected behaviour is true sure, this is important in our fast-paced world. We need all sorts of automated tests; software grows more complex by the day. And within its complexity the need for Quality-First thinking is absolutely necessary. Testing the software with anything else then TDD or BDD means you test the software after its implementation, where it'll be expensive to find bugs in architecture and structure. So, you provide automated tests, but that doesn't mean you think about quality.

> It’s too expensive and takes too much time to write unit-tests.

A developer’s work is done only after his/her software is tested properly, we understand that TDD is not always possible, however thinking about writing unit-tests later equals to never. Writing unit-test that test your code properly with all its complexity a couple of days or even weeks later is just not possible, implement unit-test while the train of thought is still fresh so you could cover most of the complexity of your code.

Estimate your stories and tasks including the implementation of unit-tests. Don't ask, just do. We often hear the stakeholder don't want to pay for unit-tests. The truth is, the stakeholder often doesn't know what that means. In our experience a stakeholder will, not be mad at you if you deliver reliable and maintainable software.
Did you ever hear the statement this part of code is not testable? Start thinking about redesigning this part. Software, which is not testable is useless and expensive.
The reason you should write meaningful unit-test is obvious if you think with quality in mind. In a middle- or long run unit-test will help you, maintaining your software. Refactoring will not be such a huge pain. With the help of a good unit-test set you will easily see where your changes break something and where software behaves in a non-expected way. Your unit-test set should cover at least 60%-80% of your code, however, these numbers are only relevant if your unit-test tests are testing the software right.

> We've qualified testers who verify that our software behaves as expected.

Most projects and clients do have good tester or at least tester, that is true. We also see all the time that this QA-Engineers are used for regression tests or other monotone task. So, the QA-Engineer is used as a click-monkey - sentenced to do the same task repeatedly. It boring, it's error prone because in repeatable tasks you start getting blind and it's a waste of qualified resources. Again, invest in test-automation for these tasks, a QA-Engineer should focus on exploring new features testing or even breaking new features, before the feature goes life. This is where his qualification is needed. A QA-Engineer should find bugs the test-automation should verify that the software behaves as expected. And no QA-System either test-automation or QA-Engineer will test quality into the system under test. The QA-System will only show where quality is missing.

> It's just a Proof of Concept but we would like to use it productive as soon as possible

This is one of my favourite quotes/situations. In Part 2 we will go into the idea of project types and how to define a project. In this case, we should answer: It would be better, if we think about the requirements and the expected outcome, it could be that a PoC might not be enough for your expected result.

So, what is quality? Sales and delivery often use it as buzzword, like **we deliver high quality software** but what exactly does that mean.

The definition of Software quality on Wikipedia is the following:

> In the context of software engineering, software quality refers to two related but distinct notions:
>
> 1. Software functional quality reflects how well it complies with or conforms to a given design, based on functional requirements or specifications. That attribute can also be described as the fitness for purpose of a piece of software or how it compares to competitors in the marketplace as a worthwhile product. It is the degree to which the correct software was produced.
> 2. Software structural quality refers to how it meets non-functional requirements that support the delivery of the functional requirements, such as robustness or maintainability. It has a lot more to do with the degree to which the software works as needed.
>
> Wiki-Article

So even the wiki article refers to a function or technical view of software quality. We, however, want to talk about the mindset, and therefore it's important to define our understanding of quality

Quality Characteristics

The Quality building blocks

Like any other framework or conceptual thought, we need something to lean on, so we could build a foundation. Take a tree for example. You can throw around some seeds and hope you'll get someday a tree which is in a city like Vienna very unlikely, more likely the next dove will be happy snacking away on your seeds. Plant the seeds in good soil, water the seeds, fertilise it, and you will find yourself quickly with your expected tree. The same goes for quality. Throwing around some test automation, or some part of a CI/CD pipeline will gain you nothing except frustration because nothing is working as hoped and some poor guy, must spent hours on maintaining and "improving" it. Which in return is worthless and expensive. Like the tree, quality needs structure and order to grow.

The Team

Our fist building block is kind of obvious. It's the team, how we work and interact with each other. A forming and storming team will not produce overall good quality, to build a team you need trust, team players, a common vision, and reachable goals. Don't set goals to low, this will bore the team. But don't set goals unachievable high either, this will demotivate the team and the performance will drop from not reached goal to not reached goal. The forming and norming phase of team building is where the team should learn how to set goals right. And will then help to transition into the performing phase, where a team might produce magic over long distances. This leads us almost seamless to the next building block.

Communication

A team, the whole organisation is nothing without good communication. Communication must be learned; teams and it's lead as well as management should do internal and external workshops somewhat regularly. Methods like BDD can help to simplify the communication between stakeholder, PO's and the development team. Which leads to better understanding for the different point of views, but also minimizes social friction because of a common understanding and language, which understands everybody

Metrics

Metrics and monitoring are important no argument there. We need this to measure what we call objective quality.

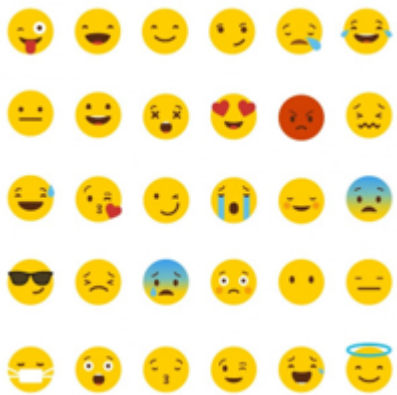
- It will give us the possibility to define quality gates in a CI/CD pipeline
- Static code analyser and its KPI's help us to identify architectural- and code-smells such as but not only:
  - Code-Duplications
  - Circular references
  - Coding Guideline violations
- We will gain a better understanding on how the software performs
  - Performance-tests
  - Load-tests
  - Mean time to recovery/repair (MTTR)
- But also gives us the KPI's we need to say our software is well tested
  - Code-Coverage (Unit-tests)
  - Path- Coverage (Unit-, Integration test)
  - Story mapping (Ui-, Integration-, Unit-test)
  - Number of defects/bugs found

These metrics are looking beautiful, if they are displayed on a monitoring system but also can give us...

- ... a false sense of security.
- ... a wrong picture of the actual state of the software if ...
  - ... the tests considered for code-coverage are not meaningful tests.
  - ... the tests are only testing the happy path.

So objective quality is important, yes. But it should be critically scrutinised to get the most out of it.

Emotions



Human beings are emotional driven. We feel if a something is good. Obviously, this has a lot to do with demographics, ages, origin, childhood and social situation. Just think about "IOS vs. Android" which is almost a religious discussion. Take a step back, you'll see that both OSes do basically the same, they even use the same Apps and both OSes' are of high quality.
The same goes for roles in a project. The stakeholder has another perspective of what important and therefor what good quality is than an end-user or a developer might have. This all need to fall into consideration when asked what we think (fell) is high quality. Since no mental health human being is above these emotions, subjective quality is something we must deal as good as possible on a daily basis. (btw. Android Rocks 😃)

Summary

Quality is a mindset. Sure, there is a technical implementation of quality with all the tests a QA-Engineer may provide, and all the metrics one can capture and form informative KPI's, but overall, we need to shape our thinking into quality right from the beginning of a project. As a responsible part of a team which implements and deliver software it's your task to form your one mindset first and the mindset of your team members second.

In Part two we'll cover the responsibilities of each team member, where should quality for each member start and where should it end (or should it end at all?) We introduce you to the quality hopper, which is a visual aid of how quality could be structured. And finally, we look into categories of projects and how to select the right project for your needs.