Software quality without testing Part 2

In the first part of this three part series we learned about the importance of a quality mindset or better said, a quality-first mindset, also we looked into our building block to create long lasting quality and defined that each individuum has the responsibility to form his own but also help to form others mindset.

Today we go a bit deeper into the roles and their responsibility. Basically, is to say, that everybody is equally responsible to generate and deliver as much quality as possible. The how will differ from role to role and from individuum to individuum.

Quality depends on teamwork: The Quality Hopper

The Team enables quality

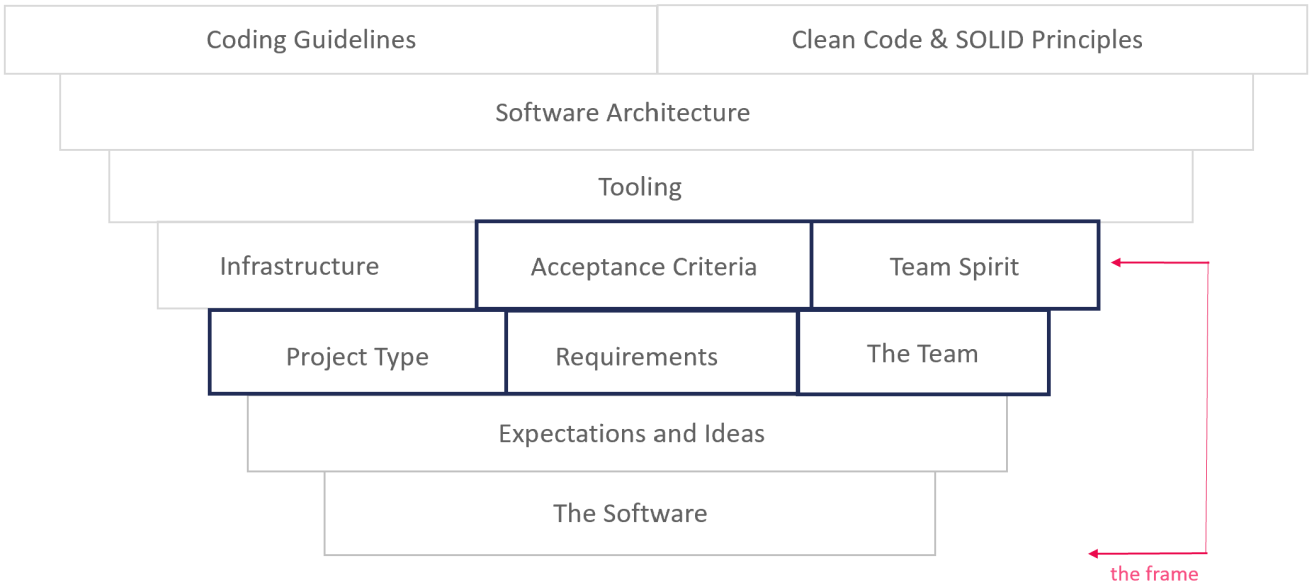
It is important to mention it again; the whole team is responsible to deliver quality. Quality Management or assurance is exactly that, an insurance or the last safety net to make sure that quality is as expected. There is no QA team in the world which could provide the team with better quality. All QA is doing, is to show that quality is missing, which most of the developers do know painfully well.

So, the real question would be how everybody in a team could enable quality without giving QA so much opportunity to throw your carefully implemented feature back to you.

The problem with feedback and criticism is that people tend to take it personally, which is tough in a development life cycle since we need fast paced feedback to deliver better software. Criticism and feedback are something we need like we need air to breath. In order to prevent that kind of social friction, we need a good team spirit and a better communication. No one likes the feeling of getting a task back, which has already been implemented, without so much as a comment or a mean comment like "fix it".

Sometimes it's easier to invest five minutes and talk with your teammate about the issue you found while code reviewing or testing the implementation. If you have the technical expertise or even if you don't have it, sit down with your teammate and fix the error together, pair programming style. Both of you might learn something. Pair programming seems slow on the outside but it's not. There are a lot of studies which say that pair programming is a very effective way to implement high quality software in a faster pace.Since there are four eyes and two brains and a continuous back and forth of ideas for possible solutions, this makes absolute sense. The nice side-effect these practices do have, is lifting team-spirit, motivation and achieving goals more easily plus you get to know your teammate. A team needs more than developers and QA-Engineers we need to talk about the other roles as well.

But first we want to introduce you to the quality hopper, which is merely a visual aid of what we think is important to build a framework for our quality-mindset. As you can see the quality hopper contains things like coding guidelines, clean code principles, architecture, tooling and so on and should be a representation of components in a software lifecycle. Not the technical implementation, but all the moving parts which are responsible to create a high-quality product.



For this chapter, we highlighted "The Team" and "Team Spirit" and we talked already about how important it is to have a good team spirit. We'll go now shortly into the typical roles of a team and their responsibility's regarding a quality-mindset.

Stakeholder

A stakeholder needs to know about his vision for the product he wants to see implemented, he should know what he wants or at least should have a good general idea. It would be optimal if the stakeholder has the skillset to communicate it to PM or the team, depending on the organization. We encounter situations where the vision and the requirements change almost daily, and the development team must make changes to their implementation accordingly.

Yes, we do hope you work in an agile environment, but that behaviour is far from agile, that is plain and simple chaos. If you encounter that kind of stakeholder, as a team you could provide the stakeholder with knowledge about agile development and agile mindset, hold workshops with the stakeholder to form a clear vision, a roadmap and plan of action as well as a way to communicate with the team. Also, the stakeholder is responsible for:

- Project frame
  - How and where is the team located (Remote vs. local vs. near-/off-shore).
  - Providing the team with infrastructure, hardware, licenses.
- Time
  - A realistic deadline based on a realistic estimation coming from the whole team (not only the Business roles).
- Market
  - A stakeholder needs to know the market and adapt the vision and the requirements accordingly. Now the power of agile can be used for what it was intended, fast adaption to an everchanging market without generating chaos in the organisation
- Steady communication with the team and to the end-user (over PR or Marketing).

Project manager, Product Owner and Business Analyst (Business roles or BRs for simplicity of the article)

As the link between the business-side and technical implementation the Business roles do have the responsibility to translate the stakeholder's requirements into manageable but also implementable packages, therefor people in these roles do need the skillset to communicate with both sides of the development process.

This role has the challenging task to manage expectations and communication of realistic timelines and delays of deadlines to the stakeholder. But also needs to work close with the dev-team by translating requirements from a visionary view or idea (Stakeholder) to implementable epics,user-stories and acceptance criteria.

It is important, that these roles don't talk about a technical detail, but about the feature's behaviour. BRs should only explain what a feature should do in certain situations. Based on that BRs should work with the dev team on a common understanding for epics and user-stories and define the acceptance criteria accordingly. So, the responsibilities of these roles could be something like that:

- Steady communication
    - Should make sure that a steady stream of communication is assured
- Behaviour over technical Details
    - These roles have nothing to do with the implementation of a feature it shouldn't be necessary to explain an expert (development) how to implement something. It is more important that the "What to implement" is very clear communicated.
- Accuracy
    - Where is a story beginning, where should it end. What are the distinctions to a story? Acceptance criteria should be formulated clear, accurate and meaningful.
- Market
    - Even if the stakeholder knows the market, the BRs should know the market as well and even if it's for the reason to challenge the requirements of the stakeholder to get a more market relevant product.

Developer

A developer develops software obviously. But that also means a developer is the main source when it comes to make sure quality is as high as expected, especially the part where we talk about implementing the right software. Since people in this role, implement and develop the vision and ideas of a stakeholder, they need to understand perfectly well what it is they need to implement.

In our experience this often proofs to be the main source of failing projects. When communication lacks in clarity, often features are implemented which don't deliver the expected output. A developer must ask questions. He needs to use the platforms for grooming, refining and planning to get on the same page as the stakeholder and BRs. Only if the development-team has a full set of acceptance criteria and full understanding of the business-side of a story, only than a full commitment should be and can be given.

The reality often is different, the development-team is pushed to an early commitment, acceptance criteria are often not well formulated and stories don't have a clear purpose, obviously that makes it hard to implement the right feature, meaning, that the idea and the vison of the stakeholder is not realized.

Also challenges for a developer are, and we'll not go into details here since we will talk about this in detail in part three, things like versioning, automation of build-, deploy- and delivery-systems, framework selection, code-guideline coaching junior staff in the art of clean coding and clean architecture, selecting the right architecture for the product and the right patterns to implement new features. Finally, how to implement

proper unit-tests during the development of features. So, the responsibilities of these roles could be something like that:

- Deep Technical knowledge of environment and the software which is in development, depending on role in the development team and seniority level
- Domain knowledge
- Team player
    - Software grows more complex, there is not one software project with commercial success which is implemented by one person alone the days of nerdy hackers who sit in a dark cellar and don't talk to anybody are over. Software-development is a team sport, there is no way around it.
- Steady communication
    - As every other role a developer also needs to provide steady communication. Where are issues expected or unexpected, what is the status of the product, resource problems, technical debt, medium to big refactoring needs - especially if the team must work with legacy code.
- Apply coding and architectural guidelines by
    - reviewing code.
    - practicing clean code.
    - applying SOLID principals.
    - using pair-programming as a tool to transfer knowledge and work on critical and high-risk changes and additions.
- challenging requirements and user stories
    - This is most important to get the user stories and acceptance criteria, development-teams need to successfully implement features.
- Implementing unit- and integration-tests simultaneous with the feature.

(Test) Automation Engineer, Quality Engineer

A well-trained Quality Engineer (doesn't matter if manual or automation engineer), should bring a vast knowledge of quality and well establish quality mindset into his team. Test-planning, -design, -execution and implementation of automated tests is the daily business of a person working in QA, but before that the experience of a QA-Engineer can be used to setup a quality friendly organisation.

Challenging requirements and cooperate with stakeholders and BRs during defining user-stories and acceptance criteria especially with the view for flaws in behavioural design during the early stages of a software lifecycle, are important skills which a QA-Engineer should have and put to use in a way that the whole team profits and learns from it.

QA-Engineer - manual testing supports the team with:

- Finding bugs (technical & conceptional)
- Explorative testing of new features
- Designing test
- Planning test-execution for releases as well as user-acceptance-test

Test Automation-Engineer supports the team with:

- Implementing regressions tests on different levels (UI, DB, API, Integration, end-to-end, ...) and creating therefor a safety net.
- Automate and manage environments.

- Test-data management for QA-Engineer, Developer and the test-automation.
- Gives support for unit-test
    - not necessarily how these tests have been implemented but what the tests cover and if they have value.
- Gives support for implementing the technical part of DevOps.

Both support the team with:

- Steady communication in all directions to communicate
    - the current state of the software
    - findings and issues
    - performance changes (e.g. application takes too long to load and UI test-automation fails therefore)
- Challenging Requirements and User-Stories
    - Working with the stakeholder and BRs as early as possible together on creating meaningful and valuable requirements, epics and stories.

During the description of the individual roles, we mentioned requirements user-stories and acceptance criteria,quite a bit, also it's part of the quality hopper, therefore we'll give you a very brief look at our take on that. Briefly because that is a topic which could fill an article by itself and we don't want to overdo it.

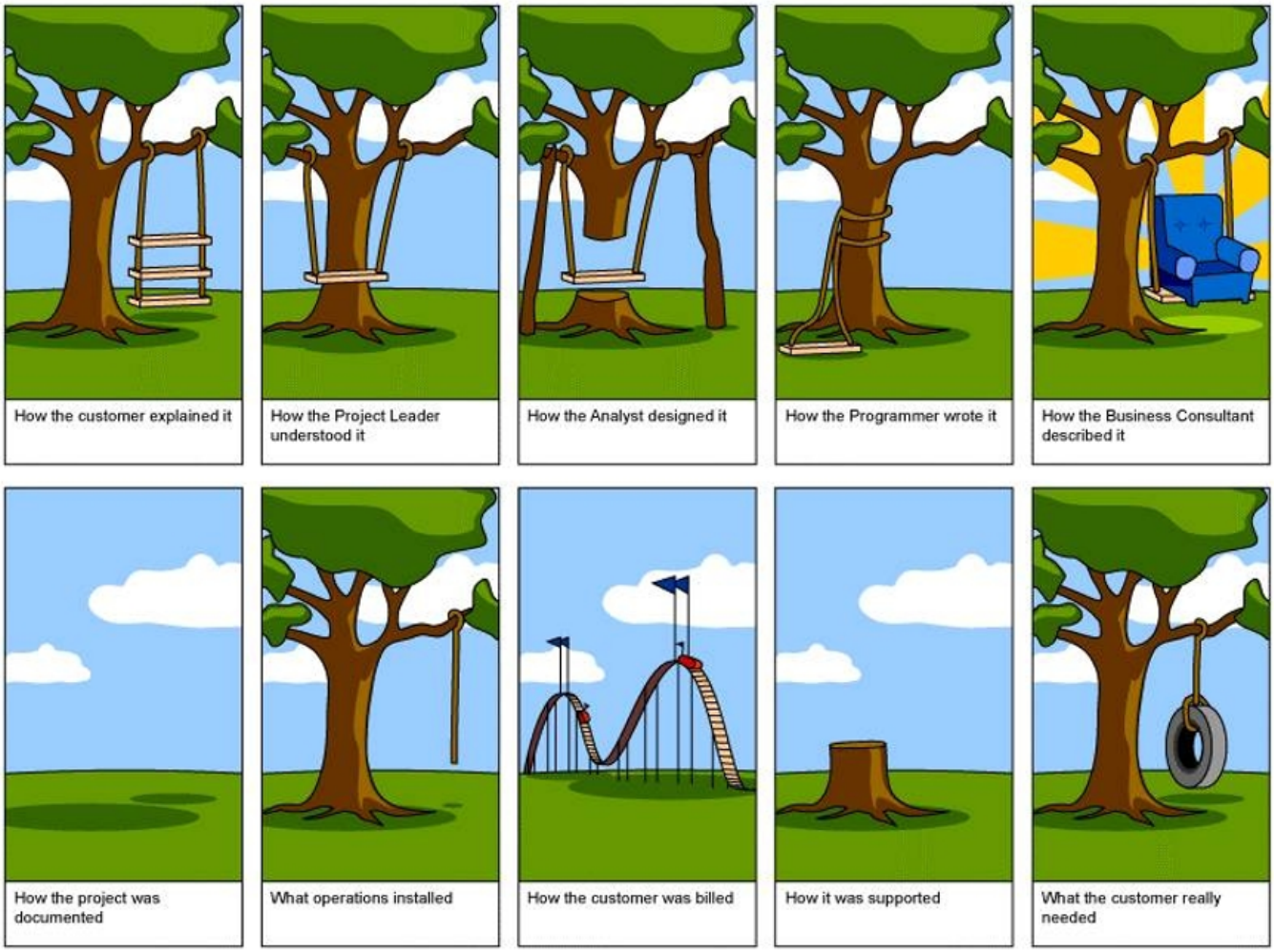
Requirements

Imagine the following scenario. You work in an agile team; it's beginning of week two in your iteration. The daily just started when the stakeholder joins in. This happens sometimes, so no reason to worry. The daily shows, that you are on the right track and if nothing unexpected happens, then you could reach all goals for that iteration, which motivates the team just a bit more.

However, before the daily ended the stakeholder wanted to say something. "Guys, we need a digital whiteboard, by the end of the next iteration!" - silence. Finally, one of your team members askes the important questions. "A what? What for? Why don't we use one of the hundredth solutions out there?"

Vaguely the stakeholder starts to explain what he needs but doesn't go into much details. As he thinks you have all the information you need, he just leaves with the words, we are agile, this should be no problem.

Don't get us wrong, this story is exaggerated, we dearly hope that we don't have to work with this kind of people but the essence of the story, that we need more detailed requirements and a better picture of the stakeholders expectations, is true, and the ill communication happens all the time.



Without clearly defined requirements, which we than could use to create well designed user-stories, we can't hope to build the right software. We do understand different things if not clarified properly.

Requirements should describe the general idea. It can be very detailed but doesn't have to. To describe a requirement more detailed you could use Epics.

Epics can be a breakdown of a requirement into estimable and plannable units. An epic should go more into details since it's a part of a requirement, it is possible to define very clearly what should be implemented, what not, and when is the epic successfully done.

User-stories should be the most detailed part of the whole process. Requirements and Epics are important to understand where we want to go. But user-stories tell us how to go there. User stories should tell us what the goal is and why it's needed in a clearly defined way:

> As [a specific user],
> I want [some goal]
> so that [some reason]

A user-story should only explain the behaviour of a feature. We don't want to describe technical details since a user-story can be and should be used as reference and documentation for a feature. A big part of user-stories are acceptance criteria.

Acceptance Criteria help as to distinct the goal of a story. They give us behaviour for different situations and edge cases. Behaviour Driven Development (BDD), or better said, the idea of describing acceptance criteria

not technical but with behaviour, has proofed to work very well. We could use the Gherkin syntax to unify communication and us it as a "building pattern" to write acceptance criteria which only describe behaviour. The Gherkin syntax is simple, it's a structured and natural language based and uses only a couple of keywords.

> **Given** there is a precondition
> **When** something is done
> **Then** the feature has to behave in a specific way.
>
> **AND** this keyword can be used do combine and create more complex given/when/then statements.

Tasks gives us a way to break down user-stories even more and go into technical details of the implementation. Some teams will go as far as defining APIs, classes or even methods during the planning so a feature-team can work on that story regardless if some necessary part of feature is implemented.

Summary

Quality needs people who can communicate, take and give feedback without making it personally. To form a quality mindset means to understand that every person in a team has his or her own responsibility regarding quality, which only ends if the project is terminated. Quality needs also a clarity in visions, ideas, requirements, and expectation.

In Part three we'll cover categories of projects and how to select the right project for your needs. How to setup a quality friendly technical environment in which it is possible to deliver quality before we move into testing the software.