Testplan

# Why do we test?

We test to make sure everything is working as intended and is safe to use. We can write as many lines of code as we want. But if it doesn’t work, there is no point to all the written code. You don’t want to release a broken application. So, we need to run a lot of tests and different tests to make sure that every aspect of our application is working properly. And if we find a mistake, we fix it and test again. This process will be repeated as many times as needed to make a good and working product.

# Types of testing

Unit testing:   
Unit testing is very useful when working with an agile method. you write tests for your code which keep testing even when you make a change. Because you will make a lot of changes throughout writing your code for your application, being able to test it all the time is very useful.

Integration testing:   
Once you finish a user story or a module with a lot of code that is working together, you start integration testing. It’s basically the same as unit testing but on a bigger scale. You test if all code related to one function requirement is working properly.

## Security testing:

Of course, you want to know whether your application is safe. You can have the best Unit tested application but that won’t matter if your security is not tight enough. Once the app is deployed it should be safe for the people who use it as well as the people who made the app or the owners of it.

## Performance testing:

Whenever it is important for multiple users to be able to use the application at the same time. it is important to stress or performance test the application before you can deliver it to the product owner. Also test the application for more users than it is meant to be used by. This way you make sure performance will never be an issue.

# Testing tools

These tools are what I intend to use. I’ve researched software that will help me testing in different ways and these tools are the most obvious or easiest to use.

## Unit tests:

|  |  |  |
| --- | --- | --- |
| **Software** | **Version** | **Reasoning** |
| *MsTest* | 2.1 | Default testing software in visual studio |

## Integration tests:

|  |  |  |
| --- | --- | --- |
| **Software** | **Version** | **Reasoning** |
| *XUnit* | 2.4.0 | Default testing software in visual studio |
| *FluentAssertions* | 5.10.3 | I use this software because it is a useful tool for checking statuscodes from api-responses. |
| *Microsoft AspNetCore Hosting* | 2.2.7 | I use this tool for connecting my integration tests to a local database. |

## Performance testing:

|  |  |  |
| --- | --- | --- |
| **Software** | **Version** | **Reasoning** |
| Jmeter | 5.4.3 | When researching performance testing Jmeter would be a consistent recommendation for software to be used. It can performance test every framework and technique you might have used. |

# Code quality analysis

## What is it?

Code quality defines code that is good (high quality) — and code that is bad (low quality).

This — quality, good, bad — is all subjective. Different teams may use different definitions, based on context.

## Why do we do it?

The code quality is important, as it impacts the overall software quality. And quality impacts how safe, secure, and reliable your codebase is.

High quality is critical for many development teams today. And it's especially important for those developing safety-critical systems.

## Code Quality Analysis: Good Code vs. Bad Code

Good code is high quality. And it’s clean code. It stands the test of time. Bad code is low quality. It won’t last long.

Essentially, code that is considered good:

* Does what it should.
* Follows a consistent style.
* It is easy to understand.
* Has been well-documented.
* It can be tested.

**Testing Isn’t Enough**

Programmers aren’t perfect. Manual code reviews and testing will never find every error in the code.

A study on “Software Defect Origins and Removal Methods” found that individual programmers are less than 50% efficient at finding bugs in their own software. And most forms of testing are only 35% efficient. This makes it difficult to determine quality.

**Coding Errors Lead to Risk**

The quality of code in programming is important. When code is low quality, it might introduce safety or security risks. If software fails — due to a security violation or safety flaw — the results can be catastrophic or fatal.

**Quality Is Everyone’s Responsibility**

Quality is everyone’s job. The developer. The tester. The manager. High quality should be the goal throughout the development process.

## How do we measure it?

There’s no one way to measure the quality of your code. What you measure may be different from what other development team measures.

**Key Code Quality Aspects to Measure**

***Reliability***

Reliability measures the probability that a system will run without failure over a specific period of operation. It relates to the number of defects and availability of the software.

Number of defects can be measured by running a static analysis tool. Software availability can be measured using the mean time between failures (MTBF). Low defect counts are especially important for developing a reliable codebase.

***Maintainability***

Maintainability measures how easily software can be maintained. It relates to the size, consistency, structure, and complexity of the codebase. And ensuring maintainable source code relies on a number of factors, such as testability and understandability.

You can’t use a single metric to ensure maintainability. Some metrics you may consider to improve maintainability are the number of stylistic warnings and Halstead complexity measures. Both automation and human reviewers are essential for developing maintainable codebases.

***Testability***

Testability measures how well the software supports testing efforts. It relies on how well you can control, observe, isolate, and automate testing, among other factors.

Testability can be measured based on how many test cases you need to find potential faults in the system. Size and complexity of the software can impact testability. So, applying methods at the code level — such as cyclomatic complexity — can help you improve the testability of the component.

***Portability***

Portability measures how usable the same software is in different environments. It relates to platform independency.

There isn’t a specific measure of portability. But there are several ways you can ensure portable code. It’s important to regularly test code on different platforms, rather than waiting until the end of development. It’s also a good idea to set your compiler warning levels as high as possible — and use at least two compilers. Enforcing a coding standard also helps with portability.

***Reusability***

Reusability measures whether existing assets — such as code — can be used again. Assets are more easily reused if they have characteristics such as modularity or loose coupling.

Reusability can be measured by the number of interdependencies. Running a static analyzer can help you identify these interdependencies.

# CI/CD

## What is software release management?

Release management oversees all the stages involved in a software release from development and testing to deployment. Release management is required anytime a new product or even changes to an existing product are requested.

## How to manage your software?

Release management oversees all the stages involved in a software release from development and testing to deployment. Release management is required anytime a new product or even changes to an existing product are requested.

As you can see, there are 8 steps in the release management cycle. In this semester we will focus on step 3 (software build).

## What are the important concepts to know?

* Version control
* Software branches
* CI
* CD

## Continuous integration

### What is it?

It’s a software development practice in which developers merge their changes to the main branch many times per day. Each merge triggers an automated code build and test sequence, which ideally runs in less than 10 minutes. A successful CI build may lead to further stages of continuous delivery.

### How does it work?

If a build fails, the CI system blocks it from progressing to further stages. The team receives a report and repairs the build quickly, typically within minutes.

By working in small iterations, the software development process becomes predictable and reliable. Developers can iteratively build new features. Developers can fix bugs quickly and usually discover them before they even reach users.

Continuous integration requires all developers who work on a project to commit to it. Results need to be transparently available to all team members and build status reported to developers when they are changing the code. In case the main code branch fails to build or pass tests, an alert usually goes out to the entire development team who should take immediate action to get it back to a “green” state.

### Why do we need it?

In business, especially in new product development, we often don’t have time or ability to figure everything upfront. Taking smaller steps helps us estimate more accurately and validate more frequently. A shorter feedback loop means having more iterations. And it’s the number of iterations, not the number of hours invested, that drives learning.

For software development teams, working in long feedback loops is risky, as it increases the likelihood of errors and the amount of work needed to integrate changes into a working version software.

Small, controlled changes are safe to happen often. And by automating all integration steps, developers avoid repetitive work and human error. Instead of having people decide when and how to run tests, a CI tool monitors the central code repository and runs all automated tests on every commit. Based on the total result of tests, it either accepts or rejects the code commit.

## Continuous delivery

### What is it?

Continuous Delivery is an approach in software development where teams produce software in short cycles and ensure that the software can be reliably released at any time. It focuses on building, testing and releasing software quickly and frequently. The goal of is to make implementations (regardless of the size) as boring and predictable as possible. In this way, deployments become part of your routine and can be executed at any time.

### How does it work?

To achieve this, we need to make sure our code is always in a deployable state, no matter how many developers make changes every day. That's why we eliminate the integration, testing and “hardening” phases that normally come after “development complete”.

### Why do we need it?

It helps us to make the building process easier for us. Imagine if you have to build every version you would like to test yourself, it would be a nightmare.

# Testing

## Testing methods

To prove that all our code is able to be tested the user stories will be used as a reference. The user stories serve as a way to define a goal within the project. These goals need to be tested to make sure they work as intended.

|  |  |  |
| --- | --- | --- |
| User story | Tasks | Test |
| US-01 As a teacher, I want to be able to show a model to my class so that I can use it during my lessons. | * Link a 3d object to a virtual class for students to see * Make students able to join classes to see the 3d object | * Integrate test the api call made for linking a 3d object and get an expected response back. Do the same for students joining class. This will also be done via api call |
| US-02 As a teacher, I want to be able to spectate the students so that I can observe them during their assignments. | * Get a teacher access to the actions of a student. In real time the teacher can see the actions made by a student on a 3d model | * Make a test in which person A can access the same 3d model as person B. The model from person A also has to be in the same state as person B. |
| US-03 As a teacher I want to look back the actions of my students. So I can see whether or not they did the assignment right. | * By using a logging system for the actions done by the students on the 3d model teacher can look back at what his students did. | * Person A makes some changes to a 3d model and Person B checks the logs if he can see the actions made by person A |
| US-04 As a teacher, I don’t want students to be able to interact with each other. So that I can individually judge the students. | * Students cannot see the 3d model of others they can only Interact with the object shown to them | * Person A and Person B both have an 3d object they can see but cannot see each other’s 3d model. |
| US-05 As a teacher, I want all my students to work on the same model at the same time without trouble, so that I can teach an entire class at the same time. | * All backend functions have to work even when a large number of people are using the backend function at the same time. | * With unit and integration testing you can simulate a lot of users which all use the same function. If this works successful the test passed. |
| US-06 As a teacher, I want the program to work fast, even when there are 50 students using it, so that I can keep giving productive lessons. | * The code has to run fast and not be influenced by the number of people using it. | * By performance testing you can measure the speed and performance of your written code. |
| US-07 As a developer, I want my software application to be scalable, so that in the future the application can be expanded on. | * The application has to be built with scalability kept in mind. Our project will be improved in the future. |  |

## Test matrix

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| User story | Integration testing | Unit testing | UX testing | Performance testing |
| US-01 | X |  |  |  |
| US-02 |  |  | X |  |
| US-03 |  |  | X |  |
| US-04 |  |  | X |  |
| US-05 |  | X |  |  |
| US-06 |  |  |  | X |
| US-07 |  |  |  | X |