SOFTWARE ENGINEERING

ASSIGNMENT\_2

To analyze the requirements and provide a detailed description for the use cases. The assignment will be performed by groups of two members (same as assignment 1). The group need to meet, discuss, understand the requirements. Similar to what happens in real projects. A new video store intends to offer rentals (and sales) of entertainment material to the wider public. The video store will keep a stock of video tapes, CDs (games and music) and DVDs. The inventory has been ordered from suppliers. All entertainment media will be bar coded so that a scanning machine integrated with the system can support the rentals, returns, and sales. Existing customers will be able to place reservations on entertainment material to be collected at a specific date.

* **Functional** – the product’s compliance with functional (explicit) requirements and design specifications. This aspect focuses on the practical use of software, from the point of view of the user: its features, performance, ease of use, absence of defects.
* **Non-Functional** – system’s inner characteristics and architecture, i.e. structural (implicit) requirements. This includes the code maintainability, understandability, efficiency, and security.

The structural quality of the software is usually hard to manage: It relies mostly on the expertise of the engineering team and can be assured through code review, analysis and refactoring. At the same time, functional aspect can be assured through a set of dedicated **quality management activities**, which includes quality assurance, quality control, and testing.

Often used interchangeably, the three terms refer to slightly different aspects of software quality management. Despite a common goal of delivering a product of the best possible quality, both structurally and functionally, they use different approaches to this task.

**Testing shows presence of mistakes.**Testing is aimed at detecting the defects within a piece of software. But no matter how thoroughly the product is tested, we can never be 100 percent sure that there are no defects. We can only use testing to reduce the number of unfound issues.

**Exhaustive testing is impossible.**There is no way to test all combinations of data inputs, scenarios, and preconditions within an application. For example, if a single app screen contains 10 input fields with 3 possible value options each, this means to cover all possible combinations, test engineers would need to create 59,049 (310) test scenarios. And what if the app contains 50+ of such screens? In order not to spend weeks creating millions of such less possible scenarios, it is better to focus on potentially more significant ones.

**Early testing.**As mentioned above, the cost of an error grows exponentially throughout the stages .Therefore it is important to start testing the software as soon as possible so that the detected issues are resolved and This means that approximately 80 percent of all errors are usually found in only 20 percent of the system modules. Therefore, if a defect is found in a particular module of a software program, the chances are there might be other defects. That is why it makes sense to test that area of the product thoroughly.

**Pesticide paradox.**Running the same set of tests again and again won’t help you find more issues. As soon as the detected errors are fixed, these test scenarios become useless. Therefore, it is important to review and update the tests regularly in order to adapt and potentially find more errors.

**Testing is context dependent.**Depending on their purpose or industry, different applications should be tested differently. While safety could be of primary importance for a fintech product, it is less important for a corporate website. The latter, in its turn, puts an emphasis on usability and speed.

**Absence-of-errors fallacy.**The complete absence of errors in your product does not necessarily mean its success. No matter how much time you have spent polishing your code or improving the functionality if your product is not useful or does not meet the user expectations it won’t be adopted by the target audience.

While the above-listed principles are undisputed guidelines for every software testing profession

* Testing must be an independent process handled by unbiased professionals.
* Test for invalid and unexpected input values as well as valid and expected ones.
* Testing should be performed only on a static piece of software (no changes should be made in the process of testing).
* Use exhaustive and comprehensive documentation to define the expected test results.