**Lesson 6 Cíntia Domingos**

**Beet Seed**

**1.**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Types of testing** | | **What is checked** | **When applicable** | **Restrictions** | **Peculiarities** |
| **Functional** | Functional | - Output of specific functionalities  - The software performs the intended tasks correctly based on requirements | Throughout the SDLC | Doesn’t check intermediate state of the system when performing the actions, only external behaviour | Can be requirement-based testing (requirements prioritized according to risk criteria) or business process-based testing (based on scenarios of the daily business use) |
| GUI | Testing the interface according to requirements: size, font, colour, position, error message display, alignment, images | Earlier stages to improve quality and reduce risks, later stages of development, when the interface is more complete | Doesn’t cover non-GUI functionalities | Needs to consider screen resolutions, type of device |
| Security and access control | Testing if the application has vulnerabilities, threats, risks, protection against malicious attacks | Throughout the SDLC | Requires skills in cybersecurity | Compliance with industry-specific standards and legal regulations |
| Interoper-  ability | If the software can interact with other components and systems without compatibility issues | When the software needs to interact with external system (for example, Bluetooth) | Requires access to diverse environments; Difficult to scale | Thorough documentation of integration requirements |
| **Non-functional** | Performance | Check how the application handles different amounts of users and/or data | When the project is almost complete | Automated testing, needs access to environments that are able to simulate the required amounts of users/data | Includes Load testing (increase load to limit or more at regular intervals), Stress testing (increase load from 0 to max at once, for example), Stability/Reliability testing (average load during long-term) and Volume testing (performance with increasing data volume) |
| Installation | The software is installed correctly and the functionality is not compromised; Compatibility with different OS; Upgrades; Uninstallation | When deploying/upgrading the app | Dependence on user environment – may be difficult to replicate issues | Resource-intensive (hardware, software, personnel) |
| Usability | User experience, ease of use, accessibility | Later stages of development, when the interface is more complete | Subjective assessments by end-users | May require collecting feedback from end-users |
| Failover and Recovery | How the system recovers from network/hardware failures | Throughout the SDLC | Access to different environments and back-up systems | Simulation of various failure scenarios and recovery mechanisms; Needs back-up of systems and data |
| Configuration | If the software works under different system configurations (OS, drivers, platforms…) | Throughout the SDLC | Access to various configuration options, environments, combinations | Complexity increases with the number of possible configurations and combinations |
| **Change-related** | Smoke | Main functionalities of the application are working | After changes in the application (new build/deployment) | Does not cover testing of every functionality | Quick assessment of quality |
| Regression | Confirm if all the functionalities work after a change to the application or environment (bug fixes, migrations, settings…) | Throughout the SDLC, after a change in the application | Requires comprehensive test coverage to identify potential issues; time-consuming | Can be functional or non-functional |
| Re-test | Check if a bug has been fixed | Throughout the SDLC, after a bug is fixed | Very specific fixes; requires detailed bug reports | High collaboration between devs and QAs |
| Build verification | Verifies that new build of the software meets minimum quality standards and is suitable for further testing | During creation or deployment of the build | Does not cover comprehensive testing of all functionalities | Usually automated |
| Sanity | Check if a specific function works according to requirements | During testing phase | Does not cover comprehensive testing of all functionalities | Usually performed manually |

**2.**

Regression testing and retesting are both change-related testing types and they serve different purposes in the testing process. Retesting is performing a test case again after its respective defects have been fixed to confirm if the functionality works as expected. On the other hand, regression testing is executing all previously executed test cases after changes in the application (bug fixes or new features, for example) to confirm there are no side-effects or errors due to those changes in the rest of the functionalities that were working correctly. Retesting is a very targeted process, focusing on specific fixes or changes. Regression testing has a broader scope, but both contribute to ensuring the quality of the software.

**Beet Sprout**

**2.**

I think non-functional testing is just as essential as functional testing to release a quality product and it should only be skipped in exceptional cases: for example, in early development stages, the focus may be only on ensuring the product meets the functional requirements and aspects such as security, performance and usability may be delayed to other stages. Also, if the resources are limited, the team may prioritize only functional testing to ensure that the core features are working properly. This can also be the case for prototypes of the product, where it only matters to see how it works.

Otherwise, non-functional testing relates to many aspects that are a huge factor for user satisfaction, such as usability, reliability, performance. It ensures that the application can handle unexpected loads and maintain stability in various scenarios. The speed of responsiveness of the app is also crucial for user satisfaction.

Only when performing both functional and functional testing can we expected a higher level of quality of the software.

**3.**

Smoke testing is used to determine if a deployed software build is stable and ready for further testing. This type of testing relates only to the core functions of the application. It is done whenever new functionalities are to be added. If any issue arises, the development team will need to fix it before moving on to more extensive testing.

Smoke testing is very useful, as it contributes to the early detection of issues: it allows for critical or blocker defects for the main functionalities to be found early in the development process. Defects encountered in later stages increase the cost of the project and may also delay the release of deliverables. Consequently, smoke testing also contributes to time and resource efficiency, because the team can prioritize their efforts. This type of testing is essential in the context of early development stages, of frequent build cycles and of complex applications that have various components or dependencies.

**Mighty Beet**

**Test Suite:** [**https://beetrootqa2024.testrail.io/index.php?/suites/view/75**](https://beetrootqa2024.testrail.io/index.php?/suites/view/75)

I added more than 4 test cases to practice, but identified 4 of them for homework with [HW].