**Assignment 1 – Software Quality Assurance**

Modern Software Quality Assurance is made up of many components.

**Tracking Bugs:**

Atlassian JIRA, Bugzilla, CodePlex etc. This software should be as simple as possible to use. It should be easy to evaluate the overall health of the product. The software should have good reporting options so you can evaluate bug counts and the progress being made by the team. This software should have most of the following functionality:

* Project name
* Issues
* Feature/Ownership areas
* Defect Frequency (How often a user will experience the defect)
* Priority (Assigned developers should know what needs to be worked on)
* Issue Template (How you expect the bug to be logged)
* Target Fix Date (When the bug should be fixed by)
* Build Information (Build a bug was found in and what build contains a fix)
* Conversation (Allows conversation to take place between developers. Useful for future reference)
* Attachments (eg. Screenshots or System Logs)
* Workflow Management
* Linked terms and bugs (Keeps all relevant information for a set of related bugs together)

**Bug Reporting**

Bug reports should contain a summary, steps to reproduce the bug and the result of the bug. The report should be as clear and simple as possible to save the developers time.

**Bug Model**

A bug model is a predictive model that gives a rough idea of where you want to be at various milestones. It gives an idea of how many bugs should be tracked at any given time. It is not always accurate but helpful none the less. It can help with productivity and highlight when people are not working on the right things. Here is an example of a bug model:

|  |  |  |  |
| --- | --- | --- | --- |
| **Week** | **Number of Bugs Logged** | **Number of Bugs Fixed** | **Bug Count** |
| 1 | 30 | 10 | 20 |
| 2 | 30 | 10 | 40 |
| 3 | 25 | 10 | 55 |
| 4 | 25 | 20 | 60 |
| 5 | 20 | 20 | 60 |
| 6 | 40 | 20 | 80 |
| 7 | 30 | 20 | 90 |
| 8 | 20 | 20 | 90 |
| 9 | 20 | 30 | 80 |
| 10 | 15 | 35 | 60 |
| 11 | 10 | 35 | 35 |
| 12 | 5 | 40 | 0 |

**Using a Matrix to Define Priorities**

This is just one example, methods can vary. The top row represents percentage of users affected by the issue. The left column represents the severity of the bug. Orange is high priority, yellow is medium priority and green is low priority. Developers must find out the severity and percentage affected of each issue. This allows them to clearly see what needs to be prioritised.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **100% - 75%** | **75% - 50%** | **50% - 25%** | **<25%** |
| **Crash** | Priority 1 | Priority 1 | Priority 1 | Priority 1 |
| **Non- Functioning** | Priority 1 | Priority 1 | Priority 1 | Priority 2 |
| **Incorrectly Functioning** | Priority 1 | Priority 1 | Priority 2 | Priority 2 |
| **Incorrectly Functioning with Workaround** | Priority 1 | Priority 2 | Priority 2 | Priority 3 & 4 |
| **Performance** | Priority 2 | Priority 2 | Priority 3 & 4 | Priority 3 & 4 |
| **Cosmetic** | Priority 2 | Priority 3 & 4 | Priority 3 & 4 | Priority 3 & 4 |

**Milestones:**

Projects should be broken up into milestones such as:

* Alpha: All high priority items addressed.
* Beta: Medium priority items addressed.
* Release: Low priority items addressed.

This helps keep developers on track and release better software. By the time the software is released it shouldn’t have any severe issues as they would have been addressed in Alpha. Addressing an issue does not necessarily mean fixing it, but that there has been a discussion about the issue and a plan is in place to deal with it if necessary.

**Test Case Management Systems**

TCMS is a repository to maintain test cases and execute them. This software should have useful features such as detailed reporting and analysis. Examples of TCMS software are Zephyr, TestRail, qTest, Test Collab etc.

Test management is the process of taking your project's requirements, building a test plan, writing the tests, planning the test activities and capturing the results. This section explains what test management is, and what features you should look for in a test management tool.

**Testing**

* Black Box Testing or Behavioural Testing. It is a method of software testing that examines the functionality of an application without looking at the inner workings of the software. Testers test the UI and layout. They come at testing from the perspective of a user. They attempt to test the application in real world scenarios.
* White Box Testing. It is a method of testing the inner workings and code of an application. It does not involve testing the functionality of the application. Testers have the technical abilities of a developer.
* Grey Box Testing. It is a combination of Black Box and White Box Testing. Testers partially know the internal structure of the application. They require high-level and detailed documents describing the application, which they collect in order to define test cases.

**Testing Core Functionality**

When testing the core functionality of an application you must think about it in the simplest possible terms. For example, testing a simple username and password login. Something simple like this ends up with 7 test cases.

|  |  |  |
| --- | --- | --- |
| **Username** | **Password** | **Result** |
| Valid | Valid | Success |
| Invalid | Invalid | Fail |
| Valid | Invalid | Fail |
| Invalid | Valid | Fail |
| Blank | Blank | Fail |
| Valid | Blank | Fail |
| Blank | Valid | Fail |

**Backend Testing**

Applications must be tested to see if they can handle the number of users expected and that the software can scale as needed. This can be achieved using loads tests, performance tests and stress tests. Loads testing simulates real world load and determines if the system can handle high demand. Performance testing determines the speed, responsiveness and stability of the system. Stress testing checks whether the performance is satisfactory under extreme or unfavourable conditions. If the software passes all this testing it is most likely ready for full scale release.

**Unit Testing**

This is where individual components of software are tested. A unit is the smallest testable part of software. In OOC the smallest unit is a function. Frameworks, drivers, stubs and fake objects are used to assist in unit testing.

**Integration Testing**

This is where individual units are combined and tested in a group. They aim to expose defects in interfaces and between components.

**System Testing**

This is where the complete and integrated software is tested. This is to verify it meets specified requirements.

**Acceptance Testing**

This is where the system is tested for acceptability. It checks whether the system meets business requirements and is ready for delivery.

**Conclusion**

Software Quality Assurance is an ever-changing beast. Every year there are new methods and processes. There is a constant effort being made to enhance quality practices. There isn’t one strict set of practices to follow, each company has their own practices. With software getting more advanced and complicated each year, Software Quality Assurance becomes ever more important as it can drive down costs and production time if implemented effectively.

**References:**

<http://softwaretestingfundamentals.com/black-box-testing/>

<https://www.quora.com/What-is-a-test-case-management-system>

<http://softwaretestingfundamentals.com/unit-testing/>

<http://softwaretestingfundamentals.com/integration-testing/>

http://softwaretestingfundamentals.com/acceptance-testing/