Software Quality Assurance Plan

By Team Naegling

Contents

1. Overview

1.1 Scope

The scope of this document is to create a software quality assurance plan (SQAP) which we, in the project, can use as a guideline to keep the quality high.

This document will bring up how we will work to achieve high quality standard in accordance to how we will manage the project, which tools to use, how we will handle risks.

1.2 Purpose

The purpose of this document is to provide information on how we will work to achieve a high quality in our project and to get us, as a team, to consider quality when developing our product.

2. Management

2.1 Organization

The management structure for this project will be Agile, specifically Scrum oriented.

The organization consists of a product owner, a scrum master and a scrum team.

The product owner will prioritize the backlog during sprint planning and motivate the team with clear goals.

The scrum master will be the link between the product owner and the scrum team he also conveys the concerns from both partners. He will organize the meetings and make sure the team works according to scrum.

The scrum team works together with the scrum master to complete the committed work within the sprint.

To ensure the quality of the product the whole team together with the project owner will review the requirements every sprint to make sure everyone is on the same track and that the definitions of done are met.

The team has the responsibility to ensure the highest quality this means that it's the team who will be doing all the coding, testing and evaluating of the product. But it is the product owner that has the final word on when the task and product is done. It is also the team who will resolve problems and if the problem isn't resolved within a specific time the scrum master will turn to the product owner to verify the quality of the problem resolution.

It is the team who will be preparing and maintaining the SQAP document.

3. DOCUMENTATION

3.1 PURPOSE

This section identifies the documentation governing the development, verification and validation, as well as the use and maintenance of the software.

3.2 DOCUMENTATION REQUIREMENTS

To ensure that the implementation of the software satisfies the technical requirements, the following documentation will be used.

3.2.1 Software Requirements Specification (SRS)

Software specification review is to be used to check for adequacy and completeness of the requirements and functionality of the software. The Software Requirements Document defines all the use cases, functional requirements, non-functional requirements and constraints on this project.

3.2.2 Product- & Sprint Backlogs

The Product Backlog works as a list of the requirements in the SRS prioritized by the product owner, necessary for managing the implementations according to scrum. The sprint backlogs are derived parts from the prioritized product backlog in order to manage the work for the current sprint. The goal is for the team to know what is to be done during every iteration.

3.2.3 Software Test Cases & Reports

Software Test Cases will be used to describe a supposed result of a specific test run. Test reports will show the results of these executed tests.

3.2.4 Software Design Document (SDD)

Software design reviews are to be used for adequacy and completeness of the design documentation. This documentation should depict how the software will be structured to satisfy the requirements in the SRS. The SDD will describe the components and subcomponents of the software design, including databases and internal interfaces, as well as class and object diagrams.

3.2.5 User Documentation

User documentation will guide the end user in installing, maintaining, using and configuring the software, as well as all other essential product information.

4. TESTING STRATEGY

We have two main goals when it comes to the testing of this project, to detect deficiencies within the system and differences between implementation and requirements.

To fulfill these two goals, the testing strategy shall contain the following methods: unit testing, integration testing, acceptance testing, and regression testing. The following sections explain these methods further.

4.1 Unit Testing

When we unit test, we make a test of a small part of the source code. Unit tests are very useful for exposing bugs early on, as well as confirming the quality of the system architecture and design.

The tests are created by the developer and tests one function at a time. It's ideal that each line of code is tested, however it's neither time nor cost efficient. Instead we will define goals to make sure that the unit tests covers the most important parts of the code.

4.2 Integration Testing

An integration test executes plenty of modules at the same time to give an understanding of how the system will function as a whole. During this test the team evaluates specific points, most commonly where two components interact with each other. The reason behind this is to make it easier to find the root cause of the bug, this will take a lot of time if the team is to test the system as a whole instead. The test is based on one or more requirement(s), with the use of scenarios or use cases, in order to verify that the system functions as intended when the requirements were created.

4.3 Acceptance Testing

Acceptance testing is when the customer evaluates the quality of the system and makes sure that it meets their requirements. The scripts are generally smaller than both unit and integration tests since the customer generally have limited resources considering time. The acceptance test covers the entire system and is operated with realistic data by using either the scenarios or the use cases, previously stated in the requirements document. At the end of the project we will include a beta test session so that we in the team will have a fuller understanding of how the product is received by the customers and iron out unexpected bugs.

4.4 Test Completion Criteria

During each implementation sprint, tests will be conducted and their completeness will be judged by the following criteria:

1. *Unit Testing*: Complete when:

    a. All methods/operations have been tested separately.

    b. All major and minor bugs found have been logged and fixed.

2. *Integration Testing*: Complete when:

   a. All module integrations have been tested.

   b. All issues/defects have been logged and corrected.

3. *Acceptance Testing*: Complete when:

   a. The product owner is satisfied that the product has met the agreed upon requirements.

5. TOOLS AND TECHNIQUES

5.1 TOOLS AND TECHNIQUES FOR ASSURING QUALITY OF FUNCTIONAL REQUIREMENTS

The team has applied the following techniques in order to ensure the quality of the functional requirements:

1. Peer review: All artifacts are stored and shared on Dropbox, code is shared on GitHub. This helps the team to review the work at the same time and be able to modify it whenever needed.

2. Customer review: The team sends documentation to the product owner for feedback. The product owners' feedback is then discussed and included. The team meets with the product owner during sprint reviews- and planning meetings in order to ensure customer involvement.

The Naegling team intends to use the following tool for verification and validation of user stories:

1. Pivotaltracker: The team uses Pivotaltracker to prioritize the backlog and the user stories and assign tasks to the team members. This is easily done by creating issues with specific details for the team members by the lead.

5.2 TOOLS AND TECHNIQUES FOR ASSURING THE QUALITY ATTRIBUTE REQUIREMENTS

During the design phase, the team has developed non-functional requirements and reviewed those with the product owner. After and during the development phase, the team will use specific tools to assure that the product meets the following quality attributes from the Software Requirements Specification (SRS) document of the project.

Quality Attribute	**Tool/Technique Used**	**Rationale for using the tool/technique**
Unit Testing	The developer will test the code frequently while developing.	When testing often bugs and faults will be discovered faster and the developer can analyze and pinpoint where the fault is easier.

Fault Tolerance	Error Detection Error Handling Fault Handling Excel	It will be used to detect, handle and record the number of defects and the rate of defects through time. The fault tolerance techniques used will depend on the specific fault. Faults will be recorded with Excel.
Performance	Stress tests and Performance monitor.	These tools will help to meet the system performance requirements during development and in production.
Usability	User questionnaire or surveys. Our team members can act as users.	These techniques will help to understand the user specific requirements and show how the system is user friendly. In the SRS document there are various use cases that will be useful to refer to while testing usability.

6. Maintenance

6.1 Maintenance

Maintenance will be done by the whole team after our product is released. Just after release we will focus a lot of our time to make sure the product becomes stable and after it's stable we will gradually reduce the amount of people focused on maintaining.

6.2 Open Communication with Customers

We will have an open communication with our customers to receive input about our product, such as software issues that we have missed and suggestions for improvements. We shall provide an email where they can send feedback and we will provide information to the public about updates and hot fixes through our website.

6.3 Testing

We will plan to do a comprehensive testing phase before releasing a new version. We will carefully test new and old functionality. We will reflect on possible side effects our changes might cause.

7. Training

For each sprint in the start of the project we will include some research and training to ensure we have the knowledge to perform our tasks and also ensure that the quality of our work will be high.

8. Risk Management

The team will all through the process of development be responsible to manage the risks that occur during the project. The incentive to raise issues and concerns during the project should be encouraged as this will prevent the possibility of loss when risks can be mitigated or reduced.

The functions the team will use to process these risks are to Identify, Analyze, Plan, Track, and Control. These functions should be activated when needed during the project. All risks should be documented in a Risk Management Form.

8.1 Identify

The first step is to identify the risk and could be done by anyone in the team. The potential risk should be discussed in the team and then documented in the Risk Management Form.

8.2 Analyze

The next step is to analyze the potential risk and the potential degree of severity and frequency. The members in the team should estimate the cost of, how severe, the potential risk is and if it would occur and how likely it is, and in what time frame it will affect the project. This should be discussed so everyone in the team can input their opinions.

After initial analysis the risks should be prioritized based on the level of severity and probability. This is done with the help of table 4-1. The team then evaluates the

	Frequent	Probable	Occasional	Remote	Improbable
Catastrophic	IN	IN	IN	H	M
Critical	IN	IN	H	M	L
Serious	H	H	M	L	T
Minor	M	M	L	T	T

Negligible	M	L	T	T	T

Table 4-1

T=Tolerable L=Low M=Medium H=High IN=Intolerable

- Tolerable - Almost no effect on the outcome of the project.
- Low - The effect is very small and could have some impact but very small.
- Medium - The effect could have some impact on schedule or cost, high enough to require control.
- High - Will in almost certain degree happen, have great impact on the project's outcome. Need close monitoring.
- Intolerable - Would have severe impact on the project's schedule and cost. This level need constant monitoring.

8.3 PLAN

In this step we need to decide how to handle a certain risk depending on the information gathered. What type of action needs to be done if it will occur and if it is possible to make changes to minimize the risk of occurring. Many risks can be averted during planning to improve the process that might be impacted to reduce the risk, often organizational issues can be handled this way. Other risks can be helped by specifying methods to handle risk as they might be affected by external factors. And for the final approach there should be a contingency plan for risks that are high to address the outcome, as they might not be able to be reduced in plan.

The team also need to set the measurements, metrics, of the risks in the document so they can be observed and tracked during the project. In any of the events associated with a risk starts the appropriate action should be executed.

8.4 TRACK

After planning we need to track or monitor the different states of risks with the help of the metrics related to the risks and actions taken to prevent them from occurring. Reporting and monitoring risks are done both as a team and individually.

8.5 CONTROL

Risk control checks that planned actions are followed, that the events that triggers response is handled and improves the risk management processes. It is managed by the team during the project. It also control the activities and changes are documented.

8.6 IDENTIFIED RISKS

This is the section where we specify our identified risks.

Risk type	Risk	Probability	Effects	Strategy
Management	Inherited schedule faults	Probable	Critical	Daily scrum, continuous information, documentation
Management	Lack of instructions	Remote	Minor	Stakeholder communication
Management	Management Change	Remote	Minor	Giving less power to the manager
Management	Unseen risks occurring	Probable	Unknown	Continuous risk reporting
Management	Poor planning	Occasional	Serious	Information exchange, communication
Design	Requirements change & inflation	Probable	Serious	Prioritize within current implementation, documentation
Design	Assumptions about users	Occasional	Minor	Design an easy and logic interface that is understandable
Implementation	Specification breakdown issues; unclear scope/objectives	Occasional	Serious	Weekly manager to make critical decisions and for information sharing
Implementation	Poor productivity	Remote	Critical	Short iterations
Implementation	Knowledge issue	Frequent	Critical	Information sharing, studying
Technical	Loss of data	Remote	Critical	Backup storage
Implementation	Messy code	Remote	Minor	Keep it simple and understandable, use relevant names, comments
Technical	Technical unavailability	Remote	Serious	Less technical dependency
Technical	Tool change	Occasional	Negligible	Stick to what is necessary and what works
People	Staff turnover	Remote	Catastrophic	Team cohesiveness, documentation
People	Misunderstandings	Occasional	Minor	Social contract, project plan, documentation

People	Low morale, lack of commitment	Remote	Critical	Team cohesiveness
People	Conflicts	Remote	Serious	Social contract, Team cohesiveness, Honesty, Good Leadership
People	Staff Absence	Occasional	Serious	Online communication tools
People	Environmental Disturbances	Occasional	Minor	Respect and authority