Group QI
QUADCOPTER
Software Quality Assurance Plan

Version 0.2
24 MAR 2014

Gothenburg University

Contents

1. Overview

by James Eagle

1.1 Purpose

The purpose of this Quality Assurance Plan is to define the standards of quality that will be applied to the Software-based Quadcopter User Interface at a Distance Project (Project SQUID), a project designed to allow easy control of a small Quadcopter, unrestricted by distance or line of sight. Due to the complexity of the project there will be a number of challenges to overcome: ease of control, stable communication channels, responsiveness and any number of quirks that come with networked flight. The document will outline the processes to be put in place by the QI SQUID team to guide, measure and quantify the project's quality in order to meet those standards and mitigate the difficulties.

As Project SQUID will be controlling an autonomous drone, there will also be obvious safety and regulatory issues, namely complying with the Swedish Transport Agency´s regulations on unmanned aircraft systems (UAS) – TSFS 2009:88. All development, testing and operation will be done with this compliance in mind, in addition to the quality assurances laid out in the document. The safety of the public and the user is built into our Quality Assurance and software requirements.

1.2 Scope

The document covers the software implementation of two main software systems as outlined below. It expressly excludes the hardware implementation of the Quadcopter (including Ardupilot microcontroller). Effective communication between these systems is a key to the project's success. The full development life-cycle, from design to deployment (defined as submission) is covered in the document .

The Ground Control Client (GCC):
This software system is intended to be the user-facing interface of Project SQUID. Installed on the user's computer system, it allows the user to control the Quadcopter at a distance. The GCC encompasses the GUI (including video feed, control and telemetry visualisation, and map functions), the controller interface (keyboard, touch interface or hardware game controller) and the establishment of connectivity to the Onboard Control and Telemetry Observation unit and interpretation of its data feed.

The Onboard Control and Telemetry Observation (OCTO) unit:
The OCTO unit is based on the Raspberry Pi computer. The management plan covers the software and network tools built for and installed on it, as well as its communications with the modem, camera, sensors, battery monitors and other telemetry it must manage. It covers all connections and communications with the Ardupilot but, as mentioned, not the functionality of the microcontroller.

2. References

Swedish Transport Agency regulations on unmanned aircraft systems (UAS) – TSFS 2009:88 - http://www.transportstyrelsen.se/Global/Luftfart/Luftfartyg/The%20Swedish%20UAS-regulation%20(TSFS%202009-88).pdf

IEEE Standard for Software Quality Assurance Plans IEEE Std 730-2002

3. Software and quality assurance plan

3.1 Management

by Mai Phuong Van

3.1.1 Organization

The organization of the SQUID project includes the development team QI and the stakeholders (our supervisor – Einar and project coordinator - Imed). Group QI is responsible for developing the quadcopter as well as ensure the quality of the product. We work as a SCRUM team. Gabriele is our SCRUM master. She holds weekly meetings to keep track of what the team has done and is going to do and detects problems as soon as possible. The stakeholders are responsible for providing our team with the key requirements and ensure that we are on the right track. Our team and the stakeholders hold meetings so that the stakeholders know what we have been doing and give us feedback.

3.1.2 Tasks

The tasks in this section refer to all the tasks to be performed during the development, operation and maintenance of the software. These tasks are selected based on the Project document plan and Project Sprint schedule. All software development tasks for SQUID, within the scope of this document, will be performed by members of group QI.

Product assessment tasks:
- code peer review
- documentation revision
- comparing the product's function and requirement document
- test result from flying the quadcopter

Process assessment taks:
- project planning
- requirement management
- test management (verification and validation)
- risk management

3.1.3 Roles and responsibilities

The responsibilities for group QI in this project include:
- Arrange daily SRCUM to update the development of the software
- Develop and maintain the SQAP
- Generate and maintain schedule for software assurance activities
- Identify lesson learned that could be improved for the future
- Document software assurance related activities

The responsibilities for the stakeholders in this project include:

- Identify and notice the group problem in their software assurance activities
- Ensure development group have proper tools to conduct software assurance activities.

3.2 Documentation

by Jani Pasanen

3.2.1 Software requirements documentation

All work done for the Quadcopter project is and shall be based on actual requirements. These requirements are written down in the form of user stories into a google document and into the groups scrum/agile management tool, Agilefant. These user stories are the basis for the tasks that group members will have to work on in each sprint to reach the goals of satisfying the user stories (users requirements).

3.2.2 Software design documentation

The design document consists of various technical analysis and design documents and diagrams. We as a group will write use cases, use case diagrams, sequence diagrams, statechart diagrams and other similar documentation to clearly illustrate the design of the end product, the quadcopter, and the software which we as a group have developed for it to meet the requirements.

3.2.3 Verifications and validation documentation

Verification and validation shall be documented by each developer when testing and validating their developed software. The purpose of V&V is to determine if developed software products conform to their requirements, and whether the software products fulfill the intended use and user expectations.

The verification plan and the validation plan document and define the verification and validation tasks and required inputs and outputs needed to maintain the appropriate software integrity level. It also provides a means of verifying the implementation of the requirements described in the requirements document and expressed in the design documents and in the testing as expressed in the project's test documentation.

3.2.4 User documentation

During development each developer shall document their code or installation so that others can reproduce the end result. This documentation will be the basis for the user manual and will sprint by sprint be merged together when each individual system and software is integrated with each other to form one single system.

3.2.5 Software configuration management (SCM)

For the purpose of documenting what SCM activities should be accomplished, how they should be accomplished, who is responsible for specific tasks, the schedule of events, and what resources will be utilized our group will utilize the functionality of Agilefant.

For the purpose of maintaining, storing, securing, and document controlled versions and related artifacts of the identified software during all phases of the software life cycle we will utilize the functionality of Git and Github.

For the purpose of managing bugs reporting and resolution we will utilize the functionality of the online tool hostedredmine.com.

3.2.6 List of documents that are to be reviewed or audited for adequacy

All design documents and this SQAP are subject to review and audition. Review shall be made at each sprint meeting and it is during the daily scrum meetings that it shall be determined if any document needs to be updated and who shall do it.

3.2.7 Tools

This section shall identify the software tools, techniques, and methods used to support SQA processes. For each, this section shall state the intended use, applicability, or circumstances under which it is to be used or not to be used, and limitations.

Toolname	**Version**	**Purpose**
Agilefant	3.5	Scrum management, SCM
GitHub, Git		Document and code versioning
Google Document		Used to edit and store the latest version of this quality assurance document
www.hostedredmine.com		Buggtracking
Scrum		Govern how the project work shall be conducted.
Dropbox		Shared filestore. Root for files that are staged for commit and push to GitHub. Not intended for version control but for easy file access and file share.

3.2.8 Maintenance

The SQAP and other documentation shall be regularly updated. It is during the daily scrum meetings that it shall be determined if any document needs to be updated and who shall do it.

Documentation shall be stored on Dropbox folder called Project Quadcopter. The files and folders under the folder Project Quadcopter shall be version controlled by GitHub.

Commit and push to GitHub shall be done every time a document is changed or added to the document store. It is during the daily scrum that team members shall inform that a document has added or changed and that commit and push to GitHub has been done.

3.3 Test

by Aurélien Hontabat

This chapter will deal with our verification and validation (V&V) effort towards testing if our product fulfills the intended use and user expectations.

3.3.1 Requirement Review

We will regularly review our requirements to ensure completeness and correctness.
As part of the review, we will evaluate whether requisite functionality and algorithms are viable and correct for our project. We will then update our requirements according to our conclusions.

3.3.2 Test Plan

Our Test Plan will describe the following activities:

Unit testing: A plan for testing a portion of our code. It will specify test cases that are relevant for this part of the software/hardware such as erroneous inputs. To be able to test the class or methods in question, we might need a test harness to call the method in a controlled manner, as well as automatically provide inputs and check the resulting output.

Integration testing: We will incrementally test larger subsets of interacting code, data structures or algorithms up to the entire program. Checking for example if each method is called with correct parameters

Stress testing: In this part of the Test Plan we will check if our product is scalable and can handle large amounts of data. This might in some cases already be part of the unit testing. We will estimate the resources required (time, memory) as well as its worst-case runtime O(n).

Each Test will furthermore be divided in the following steps:

Initiation and test environment/setup
Test steps

Expected outcome
What to do if defects are discovered during the testing process

Our Test Plan will be written prior to the coding phase, to ensure that we don't write the test with the code in mind.

3.3.3 Review of Third-Party Software/Hardware

Another important aspect of our V&V activity will be to determine if Third-Party hardware, software and libraries provide the following:

- Necessary functionality for our project
- Test cases that illustrate the use of the previously mentioned functionality
- Instructions for use

3.4 Tools, techniques and methodologies

by Gabriele Kasparaviciute

We will concentrate on some special programming techniques used for system reliability, adaptability and re-use components. Error avoidance refers to methods that reduce the likelihood of errors occurring in the use. Tolerance of errors associated with the writing program which means that the software continues to operate when the error occurs.

Structured Programming

Structured programming is important, because a clear control structures use makes programmers think more responsibly about their programs. Hence, they make fewer errors during program development. However, avoidance of unsafe sentence management is only the first step to a reliable programming. There are several programming languages structures, which also tend to cause errors. References, parallelism, recursion, these mentioned programming and design techniques are useful, and there is no need to ban them completely. However, developers will use them with caution. Their potentially dangerous effects should be controlled as much as possible, by strengthening structures to abstract data type implementation.

Handling Exceptions

Once the program takes some kind of error or unexpected event, it is called an exception. Exceptions may cause software or hardware errors. If there is likely a high chance for an exception to happen, program will include a code that will detect and compute it.

CONSTRAINT_ERROR arises when the state attempts to assign a variable value, the possible values for the range.
NUMERIC_ERROR arise through faulty arithmetic. (eg division by zero)
PROGRAM_ERROR occurs when navigation structure is violated.
STORAGE_ERROR there is a lack of memory, distinguished dynamic variables.
TASKING_ERROR crashes when there is miscommunication between tasks.

Re-use of the software

One of the engineering disciplines of characteristics is that it is based on the design of systems that maximize the use of existing components.
In contrast, most of the software products are geared to the skill (craft-oriented).
There is no common software reuse components database, which is well documented and can be used in the software design. In this project, we are going to reuse parts of code found on GitHub by other programmers.

3.5 Training

by Henrik Persson

During the course of the development of the product, there may arise a number of instances where a member will have inadequate experience and/or knowledge of a certain subject in order to carry out a task given to them. If such a case should arise, the group together with the product owner will make a formal decision on whether or not special training in the given subject is required. All group members in QI are on the same level of adequate knowledge and experience in terms of programming in Java and working together using SCRUM, however there are a number of fields of knowledge which may need to be improved upon.

- Programming in C
- Programming in Python
- Navigating the quadcopter safely using the different control methods:
    - Keyboard
    - Gamepad
    - Phone touchpad
- Proper hardware modification and simple circuit theory

These areas of knowledge may be improved by means of group learning sessions, short online courses, coupled with practice sessions.

3.6 Risk management

by Gabriele Kasparaviciute

Risk analysis and management is a sequence of steps, which is followed by a designer to find and analyze risen issues. The risk is a problem that may happen or not. The key to a good set is to determine a problem and the likelihood of impact on the project plan decision. Software is a really complex matter, therefore many problems may occur during the development of it, as is often the case, it is necessary to evaluate the risks. Risk management and assessment is key to the successful implementation of the project and continues through the life of a project. It includes processes for risk

management planning, identification, analysis, monitoring and control.

As mentioned before, it is necessary to identify risks, their probability during this project and a way to mitigate it. Table below sums up all steps that should be taken to consideration in order to manage project risks, which are:

1. Risk Identification. Risks can be identified during daily scrums, also during retrospectives. Risks may also occur while trying to understand requirements, so it is important to have requirements workshop where the purpose is to make sure that all team developers are on the same page.
2. Risk Assessment. This step is up when a risk is finally identified. During this step the likelihood and the impact of the risk are taken in consideration. To assess risk, it should be shown in a graph where risk is calculated by multiplying likelihood and impact.
3. Risk Treatment. There are couple of ways to treat an emerged risk: mitigate, avoid, evade and contain. The best solution is to first of all avoid the risk in the beginning. But if this step is reached, it is good to have a prepared strategy steps (ref: table below) to know what to do next. Risk treatment also depends a lot on where risk is on the likelihood and impact graph.
4. Risk Register and Review. This can be done in a table where all the above mentioned steps are segmented and evaluated. Review should be balanced during the retrospective.

Risk	**Lack of team commitment**
Probability	Low
Mitigation strategy	Talking to team members and scrum-master about the loss of motivation, problems that occurred during the project, spending leisure time with the team.
Risk	Misunderstanding of requirements
Probability	Medium
Mitigation strategy	During the project, final design report will be complemented with necessary information to understand requirements better, such as use case scenarios, domain model and so on.
Risk	Inadequate estimation of required resources
Probability	High
Mitigation strategy	Doing more research before trying to implement something that would be costly and asking people with better knowledge.
Risk	High level of technical debt
Probability	High
Mitigation strategy	Giving more time for team developers to

	accomplish their tasks and prioritize them better with a better involvement of product owner. Also monitor technical debt during every sprint.