ETSA03-STS

SOFTWARE TEST SPECIFICATION

for <Project>

Version 0.5

Prepared by <author(s)>

<Group X>

April 3, 2020

Revision History

<Note that you are might want to create intermediate versions between the releases. This is perfectly fine!>

Name	Date	Description	Version
Author(s)		Alpha release: First draft of STS.	0.5
Author(s)		Beta release: Unit testing and system testing drafted.	0.9
Author(s)		Final release: A complete test specification.	1.0

Contents

1. Introduction

1.1. Purpose

<Describe the purpose of the document, i.e., to describe the plan for testing the robot and general goals regarding quality assurance. Furthermore, the document shall specify the test cases and test procedures necessary to demonstrate that the robot satisfies the SRS.>

1.2. Document Conventions

<Describe any standards or typographical conventions that were followed when writing this test specification, such as fonts or highlighting that have special significance.>

1.3. Intended Audience and Reading Suggestions

<Describe the different types of reader that the document is intended for, such as developers, testers, and customers. Describe what the rest of this STS contains and how it is organized. Suggest a sequence for reading the document, beginning with the overview sections and proceeding through the sections that are most pertinent to each reader type.>

1.4. References

<List any other documents or Web addresses to which this STS refers. The correct version of the SRS is the most important, but you might also need to reference ETSA03 RoboTalk. Provide enough information so that the reader could access a copy of each reference, including title, author, version number, date, and source or location.>

2. Test Plan Description

2.1. Product Summary

<Describe the robot under development in the project. This section ensures that readers that do not read any other documents still understand the fundamentals of the product that should be tested. Do not write too much, a brief summary and a reference to the SRS is enough.>

2.2. Responsibilities

<Describe who will be responsible for the different activities in the remainder of the document. You already have a test lead, but you might want to distribute the work to balance the workload.>

2.3. Schedule

<Testing will be performed during the entire development project, but with increased effort prior to the Beta release and the Final release. Unit testing will be automated using JUnit, and the complete test suite will be executed by the individual developers before commiting source code to the shared repository. Before the Final release, all system tests will be conducted and reported as described in Section 2.4 using the test report template in Appendix A. The complete system testing is planned to performed, at the latest, during the morning hours of the announced release date: May 11.>

2.4. Test reporting

<The latest version of this STS and a complete test report will be part of the Final release. The test report will follow the test report template in Appendix A, and be complemented with reports from code coverage measurements and static code analysis. Both the STS and the test report will be uploaded to the corresponding release folder on Google Drive.>

3. Static Testing and Quality Assurance

During the robot development project, the group will apply static testing techniques to ensure a high quality software product. These techniques include both manual and automated activities, i.e., document reviews and code reviews, as well as using static code analysis tools.

3.1. Document reviews

<All documents will be reviewed by at least one project member beyond the original author. As the SRS is considered the foundation of software quality, it will receive particular attention. The SRS will be reviewed using perspective-based reading supported by a checklist at Exercise 4 on April 26. On that day, all reviewers will conduct an individual review of the latest version of the SRS. Our requirements engineer will then chair a review meeting and an assigned scribe will establish a formal review protocol. The review protocol will be part of the Beta Release to the regulatory body. The following roles and perspectives will be part of the SRS review:

- Requirements engineer: author of the document and chair of the review meeting
- Test engineer: reviews with the purpose to ensure that the requirements can be verified
- Sales engineer: reviews to ensure that the SRS is in line with the customer's needs
- Development lead: reviews to ensure that the requirements are feasible to implement
- Project manager: reviews to balance the project's scope given the time until the deadline
- Configuration manager: reviews to validate that the resulting robot will be a useful in Robocode

>

3.2. Code reviews

<Describe how and when you will perform code reviews, e.g., ad hoc, before each commit, after each commit, in batches of commits, after completion of a class, before each release,

before the final release... Also state who (or which role(s) will review the code. Finally, if you have a special approach to review source code for the purpose of requirements verification, explain it here.>

3.3. Static code analysis

<Two tools will be used to automatically assess the source code quality.

- SpotBugs is a static code analysis tool that analyses Java byte code to check for more than 400 bug patterns, such as null pointer dereferences, infinite recursive loops, bad uses of the Java libraries, and deadlocks. We will run SpotBugs using its default settings on the entire Java project.
  - The source code in our Final Release shall not contain any SpotBugs issues (test classes excluded).
  - SonarLint is a static code analysis tool that reports common code smells and violations of established code conventions. We will run SonarLint using its default settings on the entire Java project.
  - The source code in our Final Release shall not contain any SonarLint rule violations of the following severity: major, critical, or blocker (test classes excluded).

>

4. Test Design Specification

<Unit testing will be a continuous activity throughout the development process, and the developers will add new test cases as they develop the robot production code. The testing will be largely automated, using JUnit and the framework provided through RobotTestBed.>

4.1. Test Environment

<Describe the environment(s) that will be used for the dynamic testing in the project. Report the OS and hardware specification of the computer(s) used during the testing. Specify which version of Robocode will be used, and also the JUnit version.>

4.2. Unit testing

<All classes, if not listed under Section 4.6, will be tested by automated unit tests developed in JUnit. The target for this white-box testing activity is 80% instruction coverage of the Java project as measured by EclEmma (excluding the test classes themselves).>

4.3. Integration testing (Optional)

<If applicable for your robot design, you can add a description of how you plan to do integration testing. Are there any classes that should be tested together? Describe it here, most likely as an automated white-box testing activity similar to the unit testing.>

4.4. System testing

<All requirements, if not listed under Section 4.6, will be verified by test cases developed using a black-box approach. RobotTestBed will be used to automate system testing when applicable, e.g., for the quality requirements related to battle performance.>

4.5. Acceptance testing

<The acceptance testing for this robot development project will be conducted by the customer. We have no influence of how the customer will proceed with this testing activity, thus it will not be further elaborated in this document.>

4.6. Static testing only

<Due to the nature of the product and the operating environment, all methods can not be tested by unit tests and all requirements can not be verified through system test cases.

The following source code units will not be exercised by automated unit tests, but instead they will be reviewed as described in Section 3.2:
<Class, method>
<Class, method>
<Class, method>
...

The following requirements will not be verified through running system test cases, but will instead be addressed by code reviews and walkthroughs as described in Section 3.2.
<Feature, Requirement ID>
<Feature, Requirement ID>
<Feature, Requirement ID>
...

Do your best to motivate why this approach is valid. Why should your customer trust the results from your analysis?>

5. Test Case Specification

This section specifies the test cases used in the project.

5.1. Automated unit tests

<The source code is tested by the following test suites:

Class	Test class	Numbers of test cases

>

5.2. Automated integration tests (Optional)

<If you have any, describe the automated integration tests here, in line with the unit tests.>

5.3. Automated system tests

<Describe the test cases automated using RobotTestBed here>

5.4. Manual system tests (Optional)

<Describe any manual system test cases here. If you can come up with any! The only thing the Regulatory Body can think of would be to use the Interactive from the Robocode sample bots.>

6. Requirements Traceability

<Show a simple traceability matrix here to connect individual requirements and test cases>

	TC1	TC2	TC3	TC4				
Req1	X							
Req2		X						
Req3		X	X					
Req4				X				

A. Appendix A: Test Report Template

ETSA03 Test Report for <RobotName>

This document reports from testing and quality assurance activities conducted by <Group X>.

All results in the report are approved by:
<Signature>
<Name>
Test lead

A.1. System test results

Test cases executed by: <Name>
Date: <Date>
Robot version: <Version>
Robocode version: <Version>
Test environment: <OS> running on a <Hardware Specification>

Test case ID	Pass/Fail	Comment

A.2. Requirements verified through static testing

Report any requirements that were not dynamically tested here. Motivate why they have been sufficiently verified through static testing.

A.3. Coverage report from automated testing

Instruction coverage by unit test suite (see Section 5.1): XX% <Report and motivate any deviation from target in Section 5.1>

A.4. Quality report from static code analysis

Number of SpotBugs issues: XX <date of analysis>

<If not 0, list remaining issues here and explain why they have not been removed>

Number of SonarLint rule violations of severity major or higher: XX <date of analysis>

<If not 0, list remaining rule violations here and explain why they have not been removed.>