Testing Document

Testing functional requirements........1
Testing non-functional requirements........3
Test-Driven Development........4
Random & automatically generated tests........4
Testing the user interface........4

Testing functional requirements

Requirement	**Test**
The programmer can create event to get updated values from the sensors by the ports they are connected to	This kind of event is a "Subscribe" event, when it executes it will be cause execution of 'subscribe' method in 'CommandHandler' in 'RobotActuator'. In 'CommandHandlerTest' we have 2 tests for 'Subscribe': - subscribeTest – this test sends a json string of a board without index (which means index 1) and check that after subscribing to the sensor in this board, it is in the ports map of the RobotSensorData. - subscribeWithoutIndexTest – same test, only this time the json string contains an index for the board.
The programmer can create event to stop getting update values from the sensors by the ports they connected to	This kind of event is a "Unsubscribe" event, when it executes it will be transferred (through RabbitMQ) to 'unsubscribe' method in 'CommandHandler' in 'RobotActuator'. In 'CommandHandlerTest' we have 2 tests for 'Unsubscribe': - unsubscribeTest – first we check this port is in the ports map of RobotSensorData (which means we subscribed to this sensor). Next, this test sends a json string of a board without index (which means index 1) and check that after unsubscribing to the sensor in this board, it is not in the ports map of the RobotSensorData. - unsubscribeWithoutIndexTest – same test, only this time the json string contains an index for the board.
The programmer can create event to set the value of the sensors of the robot	This kind of event is a "SetSensorMode" or "SetActuatorData" event, when it executes it will be transferred (through RabbitMQ) to 'SetSensorMode' and 'SetActuatorData' method respectively in 'CommandHandler' in 'RobotActuator'. In 'CommandHandlerTest' we have 4 tests for this requirement: - setSensorModeTest, setSensorModeWithoutIndexTest – first we subscribe to the required sensor, then we set its' mode and check it was updated to the right mode.

	- setActuatorDataTest, setActuatorDataWithoutIndexTest – first we subscribe to the required sensor, then we set its' data and check it was updated to the right data.
The programmer can create event that will cause the robot to drive with the desired speed	This kind of event is a "Drive" event, when it executes it will be transferred (through RabbitMQ) to 'Drive' method in 'CommandHandler' in 'RobotActuator'. In 'CommandHandlerTest' we have 2 tests for 'Drive': - driveTest, driveWithoutIndexTest – first we subscribe to the required sensor, then we execute drive command with a specific speed and then we check that the sensor has the speed we supplied.
The programmer can create event that will cause the robot to rotate its' motors with the desired speed and angle	This kind of event is a "Rotate" event, when it executes it will be transferred (through RabbitMQ) to 'Rotate' method in 'CommandHandler' in 'RobotActuator'. In 'CommandHandlerTest' we have 2 tests for 'Rotate': - rotateTest, rotateWithoutIndexTest – first we subscribe to the required sensors, then we execute rotate command with a specific speed and an angle for the motors' sensors. then we check that the sensor has the speed we supplied.
The programmer can use with number of boards	In each test we have the option to add to each board an index or a nickname, which lets us the option to use a few boards at each execution.
The programmer can create event to register all the boards and ports used by the sensors in the system	This kind of event is a "Build" event, when it executes it will be cause execution of 'build' method in 'CommandHandler' in 'RobotActuator'. This method requires to create GrovePi and EV3 objects that will try to connect to the physical boards. Thus, we were unable to test this method without connecting to the boards.

Testing non-functional requirements

The tests for the non-functional requirements will base on statistical analysis. We declare use cases that prove the appropriate properties, such that requirement that stand in the use case will be held.

Requirement	**Test**
Simple installation	The user follows the quick start instructions in the User Manual file and install all the components required the program. The installations take the user one workday at most. In the and of the installation the user can run the program as well.
Provide efficient communication in aspect of speed	The user run the program that contains reactive reactions of the robot (like movement etc.). The robot reacts dynamically.
Add hardware in a convenient way that does not require redesign	The programmer buys new device and want to add support with him. He follows the instructions in the Developer Manual file and add support in this device. Finally, the programmer runs the existing commands he implemented on the new board, by adding them to the js file and run the program.
Add software in a convenient way that does not require redesign	The programmer wants to support new command. He follows the instructions in the Developer Manual file and add support in this command. The changes made in the appropriate locations such that the patterns of the code doesn't change. Finally, the programmer runs the new command he implemented, by adding it to the js file and run the program.

Test-Driven Development

This strategy is not applicable in our project.
The main purpose in TDD development strategy is to check that the requirements specified in the ARD are implemented and work.
In our project we had to create the full path from the BPjs code to the robot commands (which includes RobotBPjs, Robot, RobotActuator & RobotUtils) to check that the requirements are implemented as we described.
Moreover, in our tests we needed to check specific values of the devices and the sensors in the robot, those tests are depended on the implementation of the code.
Thus, we had no option to use TDD in our project.

Random & automatically generated tests

In our tests we had to create a test class for some classes, for example: we created a TestBoard class that implements the IBoard interface specifically for the tests we created so it will not try to connect to the GrovePi and the EV3 boards.
In such class we had to return values that will fit our tests, random values would take the option to test values as we did in our tests.
in conclusion, we must use deterministic values for our tests so each test will test that the code works and not how the robot actuates (LEGO already did that...).

Testing the user interface

We do not have user interface in our project.

1. Testing build, integration & deployment

For testing build, integration & deployment we used Travis CI.
Travis CI is a hosted continuous integration service used to build and test software projects hosted on GitHub.
Travis CI automatically detects when a commit has been made and a pull request is created in a branch in GitHub repository that is using Travis CI, it will try to build the project and run tests, the point of this is that we can often discover very quickly if our commit broke something, and fix it before it becomes a problem and before we merged it to our master branch.
For each commit we run Build that has two jobs, running in two sequential stages:

1. Stage 1: Compile.
2. Stage 2: Run unit tests.