DIT524 - PROJECT: SYSTEMS DEVELOPMENT

Date 2017-05-24

Isak Magnusson
Kosara Golemshinska;
Laiz H. B. de Figueroa;
Melinda Ivók;
Syeda Elham Shahed;
Rema Salman;
Nina Uljanić

QUALITY MONITORING DOCUMENT

- ***Usability***

The group designed a checklist, where the product owners had evaluated the usability of the application. This checklist should have been applied on the Demo day. However, the group had not thought about it until Sprint 7, when the second evaluation was done.

Documentation were created, such as Code standards, Quality management and Risk management, Project Plan and Requirements Specification to provide more details related to the features and the steps done by the group along the process.

Table 1. Checklist to measure usability with the end-users and product owners.

Usability Checklist				
	Not good	**Need improvement**	**Good**	**Excellent**
First Impression				
Content of the App				
Valuable				
Readable and scannable				
Amount of information in each page				
Understandability of features				
User Experience on the App				
Pages response				
Accessible navigation				
Consistent navigation				
Pop-up windows: minimum amount				
Color scheme				
User Experience on the Car				
Car response				

- ***Reliability***

In order to provide a failure-free experience of the product to the users, a number of measurements can be performed, among which evaluation the cyclomatic complexity. Firstly, to evaluate the complexity of our product, a tool such as SourceMonitor can be used for all the systems. As a rule of thumb, the complexity of any method/class in our code should not exceed 20 as this could lead to a higher probability of failure. This check would improving the overall reliability of the software.

Car's Cyclomatic Complexity Analysis: The first time the code was ran on the Source Monitor tool the highest complexity found were 9, in the method "goManual()" where the user has a lot of options to choose.

Application's Cyclomatic Complexity Analysis: The first time this code was evaluated on the tool the maximum complexity measurement for the Application was 17, which goes back to the "Joystick.java" that has the highest complexity of all the other classes implemented in the application, however the average complexity on this class is 2.73. The reason of the high complexity is, because of the *if* statements used for identifying the angles for chosen side's function wanted to be followed. On the other hand, The "MainActivity.java" has 13 as a complexity, and an average complexity of 5.29, because the Bluetooth pairing inside the "onCreate()" method with all its side cases.

Raspberry Pi's Cyclomatic Complexity Analysis: Not analysed since Source Monitor tool does not run python code.

- ***Code Quality***

The codes delivered followed the Code Standard established by the group at the beginning of the project, presenting headers, names, comments, clean and organized (Figure 1).

MENACE

```
/**
 * This sketch was created to control the robot car, initialize the serials attached on the car as the raspberry pi, sensors and bluetooth module.
 * There are data exchange between the pi and the mobile application (Andriod code).
 *
 * @author - Nina (Version 1), Laiz (Version 1, 2 and 4) and Rema (Version 2, 3 and 4)
 * @editor - Isak: Serial3 connection with the application when the car faces an obstacle in order to prompt the user for a new command.
 * @editor - Kosara: Serial connection with the raspberry pi and the car in order to send and receive data for the Identify red object feature.
 *
**/

#include<Smartcar.h>

/*============================================
              Hardware initialization
  ============================================
*/
SR04 sensorFront;
SR04 sensorBack;
Gyroscope gyro(6);
Odometer encoderLeft;
Odometer encoderRight;
Car car;

/*============================================
            Pin numbers initialization
  ============================================
*/
const int encoderPinL = 2;    // <---- the number of the left odometers pin
const int encoderPinR = 3;    // <---- the number of the right odometers pin
const int TRIGGER_PIN_F = 51; // <---- the number of the ultrasound sensor pin for the front
```

Figure 1. Example of the organization, comments, headers of the Arduino code.

- ***Compatibility***

The group formulated a checklist to validate the compatibility measurement for the general connections of the co-responsible hardwares/softwares of the systems used in the project. The list displays several exchanged data between the different softwares' systems, in order to, validate the required functions performance. This checklist was applied when the code achieved the definition of Done established by the team.

Table 2. Checklist for measuring the function's' performance

Compatibility Checklist			
	Not working	**Working**	**Buggy**
Bluetooth Pairing, Serial3: connection between the car and the application.			
Serial: connection between the car and camera, Raspberry Pi			
Press "forward" arrow button to see if forward function is completed			
Press "backward" arrow button to see if backward function is completed			
Press "right-side" arrow button to see if right-side function is completed			
Press "left-side" arrow button to see if left-side function is completed			
Press "blink" button to see if blinking lights function is completed			
Press "auto" button to see if autonomous function is completed			
Press "identify colored object" button to see if the function is completed			
Press "joystick" button to see if the function is completed			