Embedded Systems Engineering

Software Engineering Lab 2

Jos Onokiewicz

7 May 2018
version 0.4

Documenthistory

Date	Version		Updates
10-04-2018	0.1	Jos Onokiewicz	English translation. More OO used in the C++ assignments. Added several IoT MQTT assignments using Mosquitto.
15-04-2018	0.2	Jos Onokiewicz	Removed unit test assignments.
07-05-2018	0.3	Jos Onokiewicz	Added more details to the assignment description. Added overview code architecture in git repository.
01-06-2018	0.4	Jos Onokiewicz	Added chapter 3 Software Engineering, updated Sprint 3 text.

Hogeschool van Arnhem en Nijmegen

Embedded Systems Engineering

INHOUD

PREFACE

This document contains the practical assignments that belong to the second term of ESD. We assume sufficient experience in the use of QtCreator and basic level Modern C++ programming. MQTT and the Roomba vacuum cleaner robot are used as an existing target for the reverse-engineering assignment.

Arnhem, May 2018
Jos Onokiewicz

1. Preparation of the development environment

This lab will focus on Modern C++ programming and system development using UML.

1.1 The transition to Modern C++

C++ have been improved several times. New standards became available in 2011, 2014 and 2017: C++11, C++14 and C++17. These three are often referred to as Modern C++.

Bjarne Stroustrup, the creator of C++, said that C++11 "Feels like a new language, the pieces just fit together better.” Compiler support for C++11: GCC 4.8 and later completely supports it. We mainly focus us on C++11 and some C++14. Full compiler support for C++14: GCC 5.0 and later.

Because there are several C++ standards, you must choose a compiler able to compile according to a chosen C++ standard.

If you become fluent with C++ you need only a few weeks to become proficient in for example Python, Java or C#.

1.2 Necessary tooling

The next tools must be installed under Linux:

- Qt Creator (version >= 4.4.1 using Qt version >= 5.9.2)
- Google C++ Testing framework
- Doxygen and doxywizard (for generating code documentation in html-format)
- plantUML
- Git
- Valgrind
- MQTT Mosquitto library

1.3 Drawing UML diagrams using plantUML

Drawing UML diagrams without much lay-out effort is very useful, especially in an agile project management approach.

It will take some time to learn the plantUML scripting language for drawing diagrams. For that reason several script for different diagrams examples are available. These examples can be easily updated to your own needs.

Source: http://plantuml.com/

1.4 Creating code documentation using Doxygen

"Doxygen is the de facto standard tool for generating documentation from annotated C++ sources, but it also supports other popular programming languages such as C, Objective-C, C#, PHP, Java, Python, IDL (Corba, Microsoft, and UNO/OpenOffice flavors), Fortran, VHDL, Tcl, and to some extent D.

Doxygen can help you in three ways:

1. It can generate an on-line documentation browser (in HTML) and/or an off-line reference manual LaTeX from a set of documented source files. There is also support for generating output in RTF (MS-Word), PostScript, hyperlinked PDF, compressed HTML, and Unix man pages. The documentation is extracted directly from the sources, which makes it much easier to keep the documentation consistent with the source code.

2. You can configure doxygen to extract the code structure from undocumented source files. This is very useful to quickly find your way in large source distributions. Doxygen can also visualize the relations between the various elements by means of include dependency graphs, inheritance diagrams, and collaboration diagrams, which are all generated automatically.

3. You can also use doxygen for creating normal documentation (as I did for the doxygen user manual and web-site).

Doxygen is developed under Mac OS X and Linux, but is set-up to be highly portable. As a result, it runs on most other Unix flavors as well. Furthermore, executables for Windows are available."

Source: http://www.stack.nl/~dimitri/doxygen/

2. IoT MQTT assignments

Public MQTT brokers are available on the internet. Your laptop must be connected to the internet to be able to use them.

MQTT Assignment 1. MQTT topics and messages using a public broker

Start your internet browser and go to https://www.hivemq.com/try-out/, click 'Try the websocket client'. Every MQTT client using the same broker must have an unique MQTT client identifier. This is already generated in the web page. Do not use a username and a password. Click the 'Connect' button.

Use your own chosen unique topic strings to avoid a collision with other users and user groups of the broker. Choose always for the root string of your topics:

```
ESEiot/1718sem4/
```

Use wildcards for subscription (for example: `ESEiot/1718sem4/#`) and send some messages.

Sometimes this broker is offline!

The Mosquitto library must be compiled and installed for being able to use the command line for executing MQTT commands for starting the broker, sending and publishing messages.

!	Always check if you have the most recent files. Download the full code repository `trunk-IoT` for the next assignments. The repository release version can be found in the column Last Modification in the WebSVN web page and in the filename of the downloaded zip-file.

MQTT Assignment 2. MQTT web app and the command line

By clicking the file `index.html` in the directory `MQTTwebapp/basicMQTTwebapp` a web browser will be started showing several web pages. The first page shows an overview about related web page programming topics and the URL's of public and local brokers. The second page shows some buttons and leds. These leds can be switched on and off by clicking these buttons. The buttons and leds are connected by publishing and subscribing to MQTT topics/messages. The round trip time is measured. The third page enables us to connect to a broker and publish and subscribe to topics/messages.

Change the root topic texts in all web page files to:

```
ESEiot/1718sem4
```

The MQTT client ID is randomly generated. This was necessary to avoid the collision of MQTT client ids. The probability that two web app users had received the same MQTT client id is very small.

Subscribe in page three to all messages using the topic wildcard: #

Start a terminal and use the next command:

```
mosquitto_sub -v -h broker.hivemq.com -t ESEiot/1718sem4/#
```

Start a second terminal and use the next command:

```
mosquitto_pub -h broker.hivemq.com -t ESEiot/1718sem4/… -m …
```

Replace … by specific concrete topics and messages.

If you switch the WebApp leds on and off you will see the topics and messages used by the WebApp.

Switch the WebApp leds on and off by these commands.

MQTT Assignment 3. MQTT client fccfMQTT and cpMQTT

One temperature conversion is already implemented. Implement the reverse conversion.

Implement the same conversions using the the CommandProcessor class in cpMQTT.

---

3. UML and C++ system development

In practice the following situation may occur: the management of a company notices that the product documentation of one of their products fails short and is out of date. Renovations of the product cannot easily be carried out. The choice is made by a newly well-trained developer to present the product documentation with attention to analysis, design, implementation and testing as much as possible in a more formal way with UML and accompanying texts.

Because IoT technology is considered important for developing connected devices, MQTT for remote controlling and monitoring the Roomba system status should be added to the new release.

The software should become available for the RapsberryPi 3 using MQTT and the SenseHAT.

A limited number of Roomba's is available.

This assignment must be carried out in a project team (4 to 5 students).

3.1 Case: Reverse engineering the Roomba vacuum cleaner

Here we take the Roomba vacuum cleaner robot that we have to reverse engineer. This means that we have to look at how a Roomba works and that we discover the functionality and operation by doing experiments and consulting all sorts of sources such as YouTube videos and available manuals.

Take the simplest Roomba with the main task of autonomously vacuuming a room and charging its battery if necessary. The Roomba will clean based on pressing one of the buttons on top of the Roomba: for example clean mode or spot mode.

3.2 Agile development approach

An agile development approach should be taken by the project team. Sprints should be sized 2 weeks. After every sprint tested code should be available showing the implemented features by a demonstration. This should result in

Draw UML diagrams on a white board or on an A4 paper and take pictures. Later on, the UML diagrams should be drawn using an appropriate UML drawing tool.

Use a GIT repository. Use different directories for packages and subsystems. This enables the project team to test parts of the system in their own main().

Consider Jos Onokiewicz as an expert in developing systems. He is not a Roomba expert! The next texts are not detailed because the details are not available yet. The project team should discuss the details and design/implementation decisions with Jos Onokiewicz.

3.3 Development: analysis, design, implementing and testing

Important development topics:

- Problem statement: http://enfocussolutions.com/writing-a-good-problem-statement/
- Use case diagram, actors, identify subsystems, use <<extend>> and <<include>>.
- Use case texts, start with brief descriptions … fully dressed.
- State chart, events, states, conditions, actions, implement in a class, use an event queue.
- C++ standard: C++14, set this as a compiler option in the makefiles.
- Abstract base class for devices: class Device containing logging (Poco lib) and reading an

INI-file.

- Motion control: Pilot and RotationMotor class, in the 'motioncontrol' package. The Pilot class should contain 2 RotationMotor objects.
- Use the class Dataframe for managing dataframes that can be communicated by a serial link to the Roomba.
- Class CollisionDetector.
- Class BatteryChecker.
- Class CleaningStrategy containing the intelligent vacuum cleaning algorithm code.
- Use the class SenseHAT for showing the Roomba status by the led matrix and controlling the Roomba by the stick.
- MQTT client class RoombaMQTT, using the CommandProcessor class as a base class.
- Use JSON for the MQTT messages: JSON++ https://github.com/nlohmann/json
- A Roomba simulator is available.
- Deployment diagram, showing the Roomba, Battery charger and RaspberryPi and the main libs and code components.

- Use a shared GIT repository for this project.
- Commit only cleanly compilable code (solve warnings as much as possible). If parts of the code are not yet compilable, comment-out these lines.
- Commit always as soon as possible related updates to the code.
- Use good clear commit messages. A proper, well written git log is an important indicator for how well thought out a project is.
- Do not commit unrelated updates in one commit step.

- Code documentation: use Doxygen. Avoid unnecessary code comments.
- Code documentation: add @todo and @bug comments to the code.
- Use a cleancoding approach to ensure good code quality (high readability)
- Every team member should use the same coding guidelines and naming conventions. Use as much as possible the C++ Core Guidelines.

3.4 Code architecture in git repository

The next pages contain for every sprint a summary of important development steps related to the directories used.

- **Sprint 1: initial version project development directory in GIT**

	Sub directory	**Contents**
1	Docs	Doxyfile, Mainpage.dox and additional .dox files. Do not use absolute directory paths. Configure the Doxyfile that all C++ code present in the next directories is processed. Put the 'problem statement' text in a .dox file.
2	_libMQTT	Copy SVN repository. Update in MQTTconfig.h the next constant GROUP{1718} to GROUP{1718sem4}.
3	_libUtils	Copy SVN repository.
4	_libSenseHAT	Copy SVN repository.
5	RoombaMQTTwebapp	2 webpages containing a short intro text (About page) and a webpage containing the 3 Roomba push buttons (Control Roomba page).
6	RoombaMQTT	class RoombaMQTT derived from CommandProcessor, heartbeat notify message (use class ParLoop), subscription to remote control messages published by the RoombaMQTTwebapp.
7	RoombaMQTT-test	Makefile, … test code RoombaMQTT
8	UML	plantUML scripts and related png files. A Doxygen .dox file can contain links to the UML png files. UML files: use case diagram.
9	<project application name>	Makefile, Main.cpp, AppInfo.h shows at least a message showing the version number and that the application has started. Main.cpp contains @author for every project team member.

- **Sprint 2: intermediate version project development directory in GIT**

The following diagram must be discussed in detail with the customer and the external system development expert.

Do some pair programming.

	Sub directory	**Contents**
1	Docs	Update Doxyfile, Mainpage.dox and additional .dox files.
2	_libMQTT	Copy SVN repository if updated.
3	_libUtils	Copy SVN repository if updated.
4	_libSenseHAT	Copy SVN repository if updated.
5	RoombaMQTTwebapp	Implement the three Roomba buttons (Clean, Spot and Dock). Related messages should be published. Replace the About text by a Roomba application related about text.
6	RoombaMQTT	Add a heartbeat notify message (every 20 seconds) using the ParLoop class. This message must be in JSON format and should contain the application name, version and some data about implemented features. Subscribe to the messages send by RoombaMQTTwebapp. Show the receiving of the messages to cout or cerr.
7	RoombaMQTT-test	Necessary? Test in 9
8	UML subdirectories in every subdirectory where UML diagrams are used	Update and add new UML files: use case diagram, deployment diagram, class diagram. Add links in related .dox files.
9	<project application name>	Instantiate a RoombaMQTT object publishing the heartbeat notify messages in JSON format. Add the code for processing the command line arguments (MQTT broker URL and port number). Update the code version.
10	Sensor and actuator classes.	Two RotationMotor objects in the class Pilot. Implement motion control based on motion curvature (R and omega). CollisionDetector class.
11	RoombaSenseHAT	Class derived from SenseHAT for showing led patterns related to the state of the application. For example: showing the pressed button.
	IOcontrol	
12		Add more …

- **Sprint 3: final version project development directory in GIT**

The following diagram must be discussed in detail with the customer and the external system development expert.

Check if updated code in the SVN repository has become available.

Do some code refactoring for improving the code quality.

Do some pair programming.

Use the RaspberryPi for connecting the serial link to the Roomba or simulation.

Use the SenseHAT for showing in the led matrix the state of the battery and the pressed buttons.

Basic cleaning strategy: move until collision and turn 45 degrees to the left or right and move on. Implement this control loop in the class CleaningStrategy.

Add to Docs directory Development.dox containg links to UML images (use case diagram, class diagram and deployment diagram). Use doxywizard to add this page to the generation steps.

Update the Docs directory containing the code documentation.

Maintain a well-organised GIT-directory (high quality code architecture).

4. Software engineering

4.1 Classes

Class: user defined data type, using encapsulation.

A class should have one single responsibility.

Rule of Three

If a class defines one or more of the following, it should probably explicitly define all three

- destructor
- copy constructor (deep copy, if class have one or more pointers referencing heap space)
- copy assignment operator (deep copy, if class have one or more pointers referencing heap space)

Rule of Five

C++11 implements **move semantics**, allowing destination objects to grab data from temporary objects:

- destructor
- copy constructor
- move constructor, using &&
- copy assignment operator
- move assignment operator, using &&

Well-formed classes exhibit:

- Completeness, class should have all operations and data needed to carry out its main responsibility.
- Sufficiency, avoid redundant and unnecessary operations on data
- Primitiveness, class should have operations at the same primitive level
- High cohesion, everything that is present is required together.
- Low coupling, little interdependence on other classes.

4.2 Code refactoring

"Code refactoring is the process of restructuring existing computer code—changing the factoring—without changing its external behavior.

Refactoring improves nonfunctional attributes of the software. Advantages include improved code readability and reduced complexity; these can improve source-code maintainability and create a more expressive internal architecture or object model to improve extensibility. Typically, refactoring applies a series of standardised basic micro-refactorings, each of which is (usually) a tiny change in a computer program's source code that either preserves the behaviour of the software, or at least does not modify its conformance to functional requirements.

Many development environments provide automated support for performing the mechanical aspects of these basic refactorings. If done extremely well, code refactoring may help software developers discover and fix hidden or dormant bugs or vulnerabilities in the system by simplifying the underlying logic and eliminating unnecessary levels of complexity. If done poorly it may fail the requirement that external functionality not be changed, introduce new bugs, or both."

Source: https://en.wikipedia.org/wiki/Code_refactoring

4.3 Pair programming

Difficult tasks are better solved by collaboration.

"Pair programming is an agile software development technique in which two programmers work together at one workstation. One, the **driver**, writes code while the other, the observer or **navigator**, reviews each line of code as it is typed in. The two programmers switch roles frequently."

"Pair programming increases the man-hours required to deliver code compared to programmers working individually. Experiments yielded diverse results, suggesting increases of between 15% and 100%. However, the resulting code has about 15% fewer defects.
Along with code development time, other factors like field support costs and quality assurance also figure in to the return on investment. Pair programming might theoretically offset these expenses by reducing defects in the programs."

"Expert–novice pairing creates many opportunities for the expert to mentor the novice. This pairing can also introduce new ideas, as the novice is more likely to question established practices. The expert, now required to explain established practices, is also more likely to question them."

Source: https://en.wikipedia.org/wiki/Pair_programming

4.4 Unit testing

Implement some unit tests if time is left.

4.5 Seven habits of highly successful coders

- Document your code.
- Get your application out as quickly as possible, get user feedback.
- Use several programming languages (C, C++, Python, C#, …).
- Have a coding standard.
- Simple over complex.
- Use self-describing names in code.
- Develop good communication skills.

Source: https://www.youtube.com/watch?v=6OOqROUQ60s