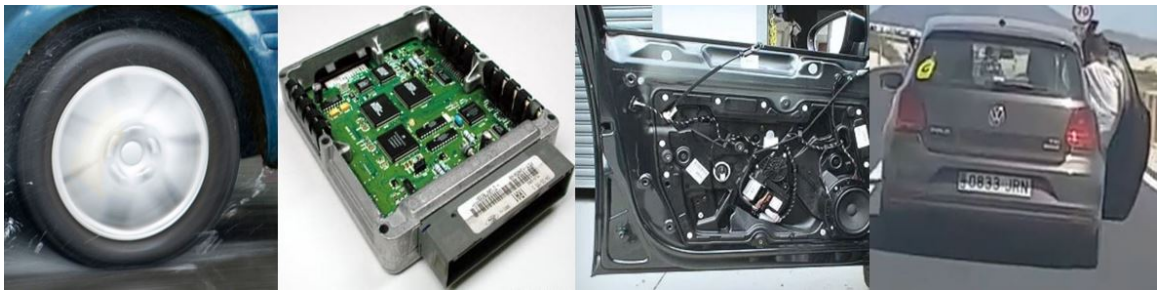
Rigorous Methods for Software Engineering (F21RS):

(2023-2024)

Specification of Coursework 1

A SPARK High Integrity Software Development Exercise



THIS IS AN INDIVIDUAL PROJECT

While discussion with fellow students as to the general nature of this project is acceptable, it is critically important that the solution you adopt as well as the associated code and report are completely your own work. The re-use of other peoples code (other than the code provided as part of this coursework) is not permitted and if identified will be treated as a disciplinary matter. Information on plagiarism can be found via

`https://www.hw.ac.uk/students/studies/examinations/plagiarism.htm`

Contents

1 Introduction

Software based control systems are increasingly used within the automotive sector – from cruise control and engine monitoring through to automated parking. In terms of safety, human error and excessive speed are a major cause of road accidents. Automated subsystems which detect such hazards and invoke corrective actions play a critical role in ensuring the overall safety of road travel.

The focus of this coursework is the development of a software based *Automatic Door Locking System* (ADLS). It is intended that the ADLS will be designed to address the following two hazards:

H-1: A door opens when the car is moving.

H-2: Occupants are locked in the car when it is not moving.

Your aim to implement the software component of a simple ADLS using the SPARK approach to high integrity Ada. You are provided with SPARK package specifications that define the safety-critical boundary of the system as well as a test harness[1], which is written in Ada. Your task is to develop the system-critical control component as well as the implementation details of the boundary packages.

In §2 a system-level description of the ADLS is provided. The tasks associated with this assessment are detailed in §3, while the testing requirements are outlined in §4. In §5 the deliverables that are expected of you are described. Finally, in §6 the submission requirements and deadline are provided. Note that this coursework counts for 20% of your overall course mark.

2 System-Level Description

The ADLS comprises of a central controller and 3 boundary subsystems that, i) sense the speed of the car's wheels, ii) control the car door locks, and iii) manage the driver's console. In order to reduce the effects of component failure 3 sensors are used to measure wheel speed. Each sensor generates a value in the range 0..151, where the units are kmph. Valid wheel speed readings are in the sub-range 0..150. A value of 151 denotes an undefined sensor reading. The sensor value returned to the ADLS controller is calculated as the majority of the 3 sensor readings. For example, if two sensors give a reading of **30** while the third gives a reading of **20**, then the majority value is **30**. The driver's console contains an indicator light that should be enabled when the doors are locked and disabled when the doors are not locked. In addition, the console contains a switch that allows a driver to select `Auto_Mode` or `Manual_Mode`. The automatic door locking capability is only available for deployment if the `Auto_Mode` has been selected by the driver.

It has been agreed with the safety engineers that a car is deemed to be 'moving' if it is traveling at great than 4 kmph. Conversely, a car is deemed to be 'not moving' if it is traveling at 4 kmph or less. When in `Auto_Mode`, if a car is moving then the ADLS controller should enable the door locks. Conversely, when in `Auto_Mode`, if the car's speed drops below the threshold valve for a moving car then the door locks should be disabled. A counter should be used by the controller to track the number of times the driver switches from `Manual_Mode` to `Auto_Mode`. Note that it is important that the counter is called `Auto_Mode_Cnt` (see §3).

[1] Software that runs your solution to the ADLS problem against a set of tests (encoded in a file called `env.dat`) and writes the results to a file called `log.dat`.

In order to address hazard's **H-1** and **H-2** highlighted above, it is critical that the following system-level constraints hold when ADLS is in the `Auto_Mode`:

SC-1: if the car is moving then the doors are locked.

SC-2: if the car is not moving then doors are not locked.

3 Tasks

The safety-critical software for the ADLS is to be written in SPARK. The safety-critical functionality is to be spread across 4 packages (subsystems) as described below:

`Sensors`**:** responsible for monitoring the speed of the car's wheels and providing the `ADLS` Controller with feedback on the wheel speed.

`Doors`**:** responsible for maintaining the state of the car's physical door locks and providing the `ADLS` Controller with control of the car's physical door locks.

`Console`**:** responsible for maintaining the state of the car's visual indicator that tells the driver whether or not the doors are locked. In addition, it is responsible for maintaining the state of the system mode, i.e. whether the system is in `Manual_Mode` to `Auto_Mode`.

`ADLS`**:** in collaboration with above subsystems, `ADLS` is responsible for the overall management of the car's automatic door locking capability. In addition, `ADLS` is responsible for maintaining a count of the number of times a driver switches from `Manual_Mode` to `Auto_Mode`.

SPARK package specifications (`.ads`) for `Sensors`, `Doors` and `Console` are available from:

`http://www.macs.hw.ac.uk/~air/rmse/SPARK/code/ADLS/`

You are required to undertake the following 5 tasks:

T1: Develop package bodies consistent with the specifications given for `Sensors`, `Doors` and `Console`. Depending upon how you define the package bodies, you may find that you need to refine the dependency contracts within the package specifications. In addition, for each of the 3 package specifications you should add a proof contract (i.e. `Pre` and `Post`-conditions) to each of its subprograms, *i.e.* functions and procedures.

T2: Develop an `ADLS` package (`.ads` and `.adb`) that implements the intended behaviour of an `ADLS` `Controller` procedure, i.e. `ADLS.Controller`, as described in §2. Don't forget the requirement relating to `Auto_Mode_Cnt`. You should include both a dependency contract and a proof contract for `ADLS.Controller`. Specifically, your proof contract should represent the safety constraints specified in §2, i.e. **SC-1** and **SC-2**. Hint: As you have no access to the context in which your `ADLS.Controller` will be called, you should not make any assumptions via a Pre-condition to the procedure.

T3: Ensure that your `ADLS` package is consistent with the 3 given package specifications as well as the test harness described below in §4.

T4: Use the SPARK proof tools to show that your code is consistent with the dependency contracts, i.e. use `gnatprove` in `flow` mode.

T5: Use the SPARK proof tools to show that your code is free from run-time exceptions and that your code is correct with respect to the proof contracts, i.e. use `gnatprove` in `prove` mode.

4 Software Testing Requirements

You are given a test harness for the purposes of testing your ADLS system software. The test harness is written in Ada (i.e. it is not SPARK compliant) and allows the simulation of the environment within which the ADLS system is intended to operate. **It is important that you do not change this code.** The test harness involves 3 packages that are described as follows:

`Env`**:** responsible for maintaining the state information associated with the `Sensors`, `Doors` and `Console` packages. The state information is read from a file called `env.dat` (see appendix A.1).

`Log`**:** responsible for logging the state information associated with `Sensors`, `Doors` and `Console`, as well as the majority sensor reading. The logger writes to a file called `log.dat` (see appendix A.2).

`Test_ADLS`**:** responsible for running *your* `ADLS` controller procedure against test data (i.e. `env.dat`). The actual reading of the test data and the recording of the results (i.e. `log.dat`) is performed by subprograms defined within the `Env` and `Log` packages respectively.

Note that the code for these 3 packages is provided in:

`http://www.macs.hw.ac.uk/~air/rmse/SPARK/code/ADLS/`

An example `env.dat` file is also given in this directory.

5 Deliverables

Your submission via Canvas **should** take the form of **TWO files**:

1. A stadalone **report file** (i.e. pdf, doc, docx) **AND**
2. A **ZIP file** containing your SPARK code and test results, i.e. the given `env.dat` file and the `log.dat` file generated by your solution to the ADLS problem (but **not** your report file).

Your report **must be** structured in terms of the following sequence of sections. Please maintain the sequence as shown below. Please include the associated deliverable identifier in each of your section headings, i.e. **D0**, **D1**, **D2**, etc.

D0: A title page that clearly identifies you, your course and your campus of study.

D1: A statement of any assumptions you have made about the informal system-level description given in §2, and how they have impacted on your design decisions. Note that we are not looking for a restatement of the given description. We are looking instead for any additional information that you believe is required in order to improve the consistency and completeness of the given system-level description. *[3-marks]*

D2: A diagrammatic representation of the software architecture of your ADLS software. Your diagram should communicate the subsystems and their local state, as well as their connectivity, *i.e.* imports and exports. The test harness should not be part of your diagram. *[7-marks]*

D3: Listings of each source file, i.e. **ALL 4 packages** (both specifications & bodies) that define the ADLS software. Ensure that your source files are well formatted and are readable. *[24-marks]*

D4: The summary file generated by `gnatprove` using `flow` mode when analyzing ALL the SPARK code for your ADLS, i.e. the `gnatprove.out` file.
[1-mark]

D5: The summary file generated by `gnatprove` using `prove` mode when analyzing ALL the SPARK code for your ADLS, i.e. the `gnatprove.out` file.
[1-mark]

D6: The `log.dat` file generated by your ADLS system software for the `env.dat` file given in: `http://www.macs.hw.ac.uk/~air/rmse/SPARK/assignment/env.dat`
This `env.dat` file will be made available by the end of week 5. *[1-mark]*

D7: With explicit reference to your code, describe how you protected against run-time exceptions, i.e. what you protected against and how you achieved the protection.
[4-marks]

D8: Select a programming language that you have previous experience of, e.g. Java, C, C++, etc. Focusing on language features, compare and contrast your chosen language with SPARK. Specifically describe using examples (i.e. code fragments) at least two strengths and two limitations of each language. You should aim for around 500 words (excluding example code fragments). *[7-marks]*

There will be 2-marks allocated according to the clarity, quality and accessibility of your report. Note that this coursework counts for 20% of your overall course mark.
[50 marks in total]

6 Submission Requirements and Deadline

- The deadline for this coursework is **Monday 23 October 2023 (week 7): 3.30pm for Edinburgh registered students and 11.59pm for Dubai registered students.** As noted above, as well as your **report file** you are also required to submit a **ZIP file**. You should use the following naming convention for your ZIP file:

  `<surname>-<username>-F21RS.zip`

  For example, `smith-as42-F21RS.zip`.

- **Note that this is an individual project which means that your submission MUST be your own work.**

- **This coursework counts for 20% of the overall course mark.**

- **The University Policy on the Submission of Coursework can be found via the following link:**

  `https://www.hw.ac.uk/services/docs/learning-teaching/policies/submissionofcoursework-policy.pdf`

A Example Test Data & Results (Appendix)

A.1: An example `env.dat` file

The data within `env.dat` is used to simulate the values of the state variables associated with the boundary packages of the ADLS software. Each row represents the set of values at a particular point in time. The first row denotes the initial set of values in the test run while the last row denotes the final set of values. Columns 1 to 3 denote the values associated with the 3 sensors as described in §2. Column 4 denotes the status of the `Auto_Mode` switch associated with the driver's Console. A 0 in column 4 denotes the `Auto_Mode` is disabled while a 1 denotes that the `Auto_Mode` is enabled.

```
0   0   0    0
1   1   1    0
2   3   2    0
3   3   3    1
4   4   4    1
5   4   4    1
5   5   5    1
10  10  10   1
```

A.2: An example `log.dat` file

The data within `log.dat` is generated by the logger (see `Log`). It records state and related information across a simulated test run of the `ADLS.Controller`. The particular file shown below was generated using the `env.dat` file given in A.1. Note that there are double the number of entries in `log.dat` as there are in `env.dat`. This is because the logger is invoked twice within `Test_ADLS`, *i.e.* before and after each invocation of `ADLS.controller`.

```
SENSOR-1  SENSOR-2  SENSOR-3  MAJOR  DOORS   INDICATOR  AUTO_MODE  AUTO_CNT

   0         0         0        0    ------  ---------  ---------     0
   0         0         0        0    ------  ---------  ---------     0
   1         1         1        1    ------  ---------  ---------     0
   1         1         1        1    ------  ---------  ---------     0
   2         3         2        2    ------  ---------  ---------     0
   2         3         2        2    ------  ---------  ---------     0
   3         3         3        3    ------  ---------   ENABLED      0
   3         3         3        3    ------  ---------   ENABLED      1
   4         4         4        4    ------  ---------   ENABLED      1
   4         4         4        4    ------  ---------   ENABLED      1
   5         4         4        4    ------  ---------   ENABLED      1
   5         4         4        4    ------  ---------   ENABLED      1
   5         5         5        5    ------  ---------   ENABLED      1
   5         5         5        5    LOCKED     ON       ENABLED      1
  10        10        10       10    LOCKED     ON       ENABLED      1
  10        10        10       10    LOCKED     ON       ENABLED      1
```