The University of Melbourne

SWEN90010: High Integrity Software Engineering

Assignment 4

Due Date: 11:59pm, Sunday 31 May, 2015

1 Introduction

This handout is the assignment 4 sheet. The assignment is worth 10% of your total mark and is done in pairs (the same pairs as assignment 2 and 3).

The aim of this assignment is to use SPARK to implement and verify the Alloy model that you produced for assignment 3. The assignment evaluates your ability to apply safe-programming subset techniques to implement a high-integrity system, to apply design-by-contract and proof to verify your implementation, and to assess the correctness of your implementation with respect to the requirements. You are required to write design contracts and SPARK code corresponding to your Alloy specification from assignment 3, and to document the faults that you find.

2 Code

SPARK versions of the HRM, heart, impulse generator, and measures packages accompany this assignment. These versions are SPARK compliant, and include preconditions and postconditions for all procedures and functions.

There is also the manual operation mode, in `manualoperationmode.adb`, which is the same example as in assignment 1, but it uses the SPARK implementations. This is an Ada program – not a SPARK program. However, systems that mix Ada and SPARK programs can still be compiled because SPARK can be compiled using a normal Ada compiler.

NOTE that as part of this assignment, you do *not* need to write an automated mode example — but you should be able to use your code from assignment 1 with no changes (or possibly minimal changes).

3 Your tasks

The assignment should be done in pairs, with careful planning around which member of the pair does which task.

1. Team member one: Derive a SPARK package interface (including preconditions and post-condition) for the `ClosedLoop` package that models the operations from the Alloy model from your assignment 3 solution (although you are permitted to fix issues with your Alloy model). This interface must contain at least the four Alloy operations completed as part of assignment 3. There is no need to model additional operators from assignment 1 – only those from assignment 3.

2. Team member two: Implement a SPARK package body for the `ClosedLoop` package. This

may be based on your implementation from assignment 1, but it also must conform to the SPARK contract from Task 1.

3. Together: Once the SPARK implementation and SPARK contracts are written, use the SPARK tools to verify whether your program conforms to its contract.

   HINT: You should first try this for a few simple operations, rather than trying to put all this together in one hit. Then, repeat incrementally for other operations.

   Record all faults that are found using the SPARK tools. Each fault should be categorised as one of the following: (1) a fault in the contract; (2) a fault in the implementation; or (3) both. Briefly describe the fault; e.g. an index out of bounds fault, or failure to establish the postcondition of an operation. You do not need to consider syntax errors.

   Note: you should fix the faults found in your SPARK contracts and implementation, as some errors may be masking others. You may choose to ignore some warnings if you believe that the implementation is correct despite the warnings. The SPARK tools are not fool proof. Briefly justify why you think the implementation is correct.

4. Compare the security of the verified and tested SPARK implementation against the two solutions produced for assignment 1. Which is the more secure system *and why*? Argue a case for your answer. In your argument, you should consider the measures taken over the course of assignments 2, 3 and 4. Limit your argument to a single A4 page.

5. Assume that as part of this series of assignments (2-4), you modelled and verified *all* system behaviour in Alloy, that you implemented the complete system in SPARK from this extended Alloy model, and you implemented the fault-tolerant design from assignment 3. Describe the improvements this would have made to the *safety* of the system. Would this have made any impact? If so, what impact and why? If not, why not? Limit your argument to a single A4 page.

4 Criteria

Criterion	Description	Marks
SPARK Implementation [3 marks]		
Correctness & completeness	The SPARK implementation correctly and completely implements the Alloy model.	1 marks
SPARK	The SPARK implementation uses suitable SPARK features to ensure the integrity of the system.	1 mark
Clarity and code formatting	The implementation is clear and succinct. The implementation adheres to the code format rules in Appendix A.	1 mark
SPARK Contracts [4 marks]		
Correctness & completeness	The SPARK preconditions and postconditions are correct, complete, and clear.	3 marks
Consistency	The SPARK contracts are consistent with the Alloy specification from assignment 3.	1 mark
Discussion [3 marks]		
Completeness	All aspects related to the case are considered.	1 marks
Understanding	The argument links to relevant theory, and demonstrates an understanding of that theory, and application of that theory to engineering high-integrity systems.	2 marks
Total		10 marks

5 Submission

Submit the assignment using the submission link on the subject LMS. Go to the SWEN90010 LMS page, select *Assignments* from the subject menu, and then select *View/Complete* from the *Assignment 4 submission* item. Following the instructions, upload a zip file containing the following:

1. A directory called `code/`, which contains all code for the system.
2. A PDF file containings: (a) the list of faults found by the SPARK tools; and (b) your answers to tasks 4 and 5.

Only *one* student from the pair should submit the solution, and the submission should clearly identify both authors.

Late submissions Late submissions will attract a penalty of 1 mark for every day that they are late. If you have a reason that you require an extension, email Tim *well before the due date* to discuss this.

Please note that having assignments due around the same date for other subjects is not sufficient grounds to grant an extension. It is the responsibility of individual students to ensure that, if they have a cluster of assignments due at the same time, they start some of them early to avoid a bottleneck around the due date. Further, giving an extension in these circumstances simply crowds out assignments and exams after this date.

6 Academic Misconduct

The University misconduct policy applies to all assessment. Students are encouraged to discuss the assignment topic, but all submitted work must represent the individual's understanding.

The subject staff take plagiarism very seriously. In the past, we have successfully prosecuted several students that have breached the university policy. Often this results in receiving 0 marks for the assessment, and in some cases, has resulted in failure of the subject.

Appendix

A Code format rules

The layout of code has a strong influence on its readability. Readability is an important characteristic of high integrity software. As such, you are expected to have well-formatted code.

A code formatting style guide is available at `http://en.wikibooks.org/wiki/Ada_Style_Guide/Source_Code_Presentation`. You are free to adopt any guide you wish, or to use your own. However, the following your implementation must adhere to at least the following simple code format rules:

- Every Ada package must contain a comment at the top of the specification file indicating its purpose.
- Every function or procedure must contain a comment at the beginning explaining its behaviour. In particular, any assumptions should be clearly stated.
- Constants and variables must be documented.
- Variable names must be meaningful.
- Significant blocks of code must be commented.

  However, not every statement in a program needs to be commented. Just as you can write too few comments, it is possible to write too many comments.
- Program blocks appearing in if-statements, while-loops, etc. must be indented consistently. Tabs or spaces can be used, as long as it is done consistently.
- Lines must be no longer than 80 characters. You can use the Unix command "`wc -L *.ad*`" to check the maximum length line in your Ada source files.