**Remarkably Busy Business’s**

**Issue Tracker System**

**ARCHITECTURAL DESIGN DOCUMENT**

# Architectural Design Document Title Page

**Version 1**

Release Date: 2024/07/03

Approved by:

Allan Hu \_\_\_\_\_\_\_\_\_\_\_\_\_

Nicolao Baretto \_\_\_\_\_\_\_\_\_\_\_\_\_

Puja Shah \_\_\_\_\_\_\_\_\_\_\_\_\_

Wah Paw Hser \_\_\_\_\_\_\_\_\_\_\_\_\_

# User Manual Version History Page

Version 1 - original 2024/07/03 by Allan Hu, Nicolao Baretto,  
 Puja Shah, Wah Paw Hser

# 

# Table of Contents

[**Architectural Design Document Title Page 1**](#_gk0t2zhwfro9)

[**User Manual Version History Page 2**](#_1pbj8w853juo)

[**Table of Contents 3**](#_otowkx7xhio5)

[**1.0: Introduction 3**](#_wfu8tcokvx32)

[**2.0: Disk space comparison of 2NF and 3NF 3**](#_hhmx7p1adh4h)

[2.1: Size of Roll-Up Implementation 4](#_olvdesnhrsel)

[2.2: Size of split implementation 4](#_4xb6tzcgf2gz)

[2.3: conclusions 4](#_pwff8p7wl6j2)

[**3.0: Architecture presentation and discussion 5**](#_8h8cjufg0qs)

[**4.0: Comparison with Alternate Design 16**](#_5a1fckkkwgi4)

[**5.0: Initial Interface Design 16**](#_elrcjtt2tcr)

[**6.0: Detailed Interface Design 16**](#_tupfklndd411)

[**7.0: Testing 16**](#_pg42egc6surr)

[7.1 Functional Test Case 16](#_5yk4u1c11fja)

[7.2 Performance Test Case 19](#_s97wtptv1cnd)

[7.3 Stress Test Case 21](#_srjvlny238t9)

[**8.0: Coding Convention 22**](#_9chxqpgkbeks)

# 1.0: Introduction

Welcome to the Remarkably Busy Business’s Issue Tracker System. This system will keep track of the requests and features that are sent in by customers and also department members. For more background information about this system, please refer to the Remarkably Busy Business Issue Tracker System Requirements Specification Version 1, and the Remarkably Busy Business Issue Tracker System User Manu w Version 1.

The purpose of this Architectural Design Document is to inform the user about the design specifications and structure of the system to guide them through implementation. A lot of the interface design is not included in this document, but you can refer to the source code modules that are included in this release.

# 2.0: Disk space comparison of 2NF and 3NF

For analysis we will make the following assumption about the ratio of the superclass to the subclass:

* There are 100 total requesters
* 40 requesters are employees

## 2.1: Size of Roll-Up Implementation

With a rollup implementation there is one kind of record providing enough space to fit an instance of the class or any of its subclasses. This record contains a name (30 bytes), a phone number (10 bytes), an email (24 bytes), and a department (12 bytes). There would be 100 instances of this record.

Record 1: (100 instances) (76 bytes per record)

* Name (30 bytes)
* Phone (10 bytes)
* Email (24 bytes)
* Department (12 bytes)

Total Space = (sizeOfRecord1 \* numberOfInstances)

= (76 bytes \*100) = 7600 bytes

= 7600 bytes

## 2.2: Size of split implementation

With a split implementation there are two kinds of records. The first kind of record contains the information that is stored for all instances in the family of the class, a name (30 bytes), a phone (10 bytes), and an email (24 bytes). The second kind of record will contain the information stored for only instances of the sub class, and a foreign key.

Record 1: (100 instances) (64 bytes)

* Name (30 bytes)
* Phone (10 bytes)
* Email (24 bytes)

Record 2: (40 instances) (42 bytes)

* Name (30 bytes)
* Department (12 bytes)

Total Space = (sizeOfRecord1 \* numberOfInstances) + (sizeOfRecord2 \* numberOfInstances)

= (64 bytes \* 100) + (42 bytes \* 40) = (6400 bytes) + (1680 bytes) = 8080 bytes

= 8080 bytes

## 2.3: conclusions

Our analysis shows that a roll-up implementation will use less disk space than a split implementation (7600 < 8080), so we will use a roll-up implementation.

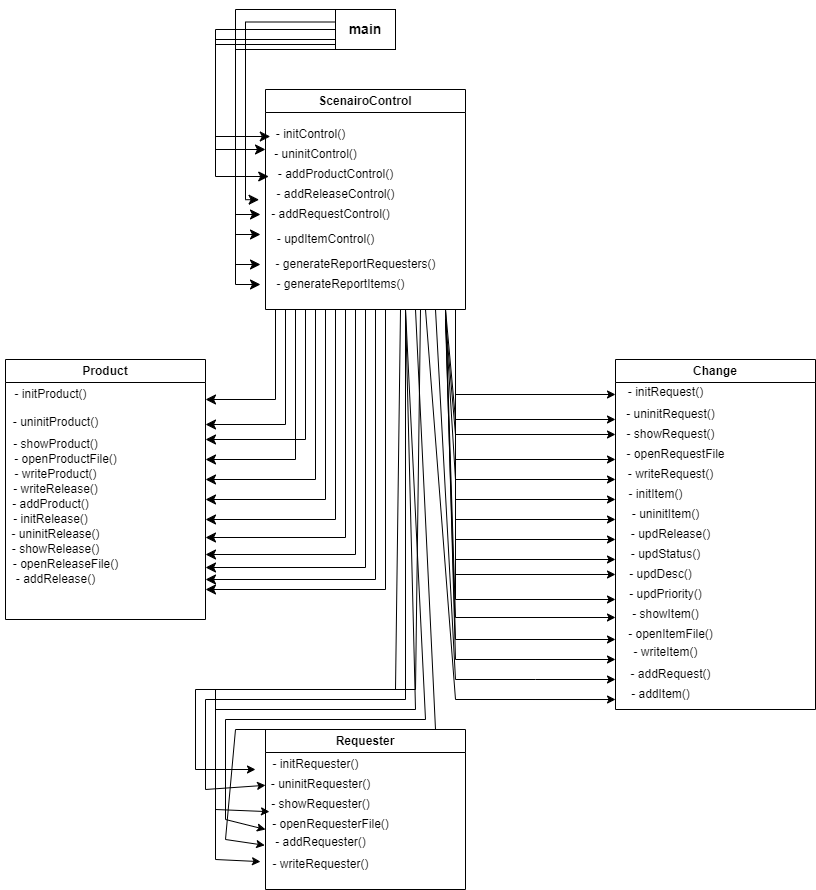
No changes need to be made to the UI described in the user manual to accommodate this design choice.

# 3.0: Architecture presentation and discussion

**3.1: architecture discussion**

The design architecture will include 5 modules, main, ScenarioControl, Product, Requester, and Change. We decided to implement our architecture using a Centralized Scenario design. The ScenarioControl will control the lower scenario modules including, Product, Requester, and Change. We decided to use this architecture design because it provides easy and quick transition between lower level modules and reduces coupling between modules.

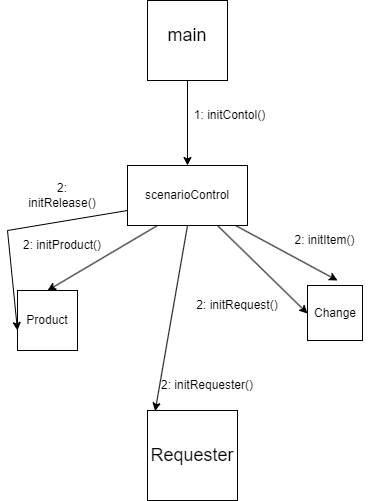
**3.2: Overall OCD**

****

**3.3: System scenarios**

Below is a list of all of the scenarios that this program could deal with along with the OCD associated with each scenario

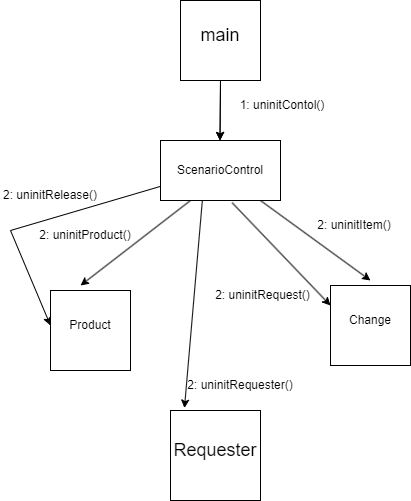
1. **System start**



Scenario description

* When the issue tracker system is started, the main module tells the scenario control module to initialize by calling initControl().
* The scenario module then tells the product and release module, requester, and change request and items module to initialize by calling initProduct() and initRelease(), initRequester(), and initItem() and initRequest() respectively.

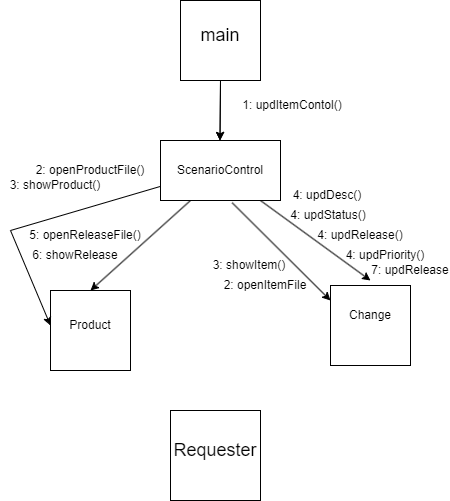
1. **System exit**

****

Scenario control

* Main module tells the scenario control module to shutdown by calling uninitControl().
* The scenario module then tells the product and release module, requester, and change request and items module to uninitialize by calling uninitProduct() and uninitRelease(), uninitRequester(), and uninitItem() and uninitRequest() respectively.

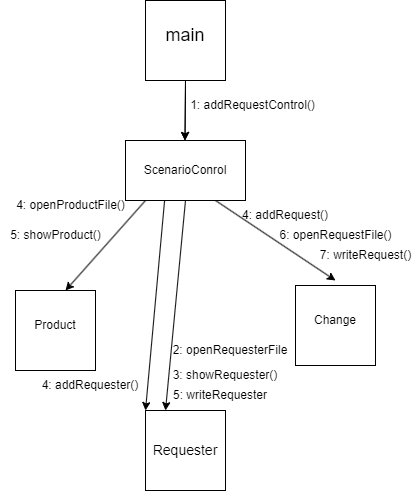
1. **Update change item**

****

Scenario control

* The user wants to update a change item
* The user can either see the change items for all products or just one product
* In the later, the user will be shown a list of products where they will choose one
* The user will now select what status of change item they want see
* A sorted list of change items will be shown and the user will select the one they want to change
* The user can now select what they want to change
* If the status is changed to done, release will need to updated to reflect that

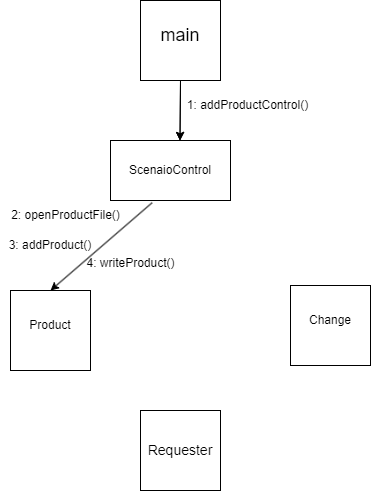
1. **Create change request**

****

Scenario control

* A new bug or feature is requested by a customer
* A list of requesters will be shown
* If the requester is not in the list, then a new requester will be added by entering the requester’s name, phone number, and email address into the system. If the requester is part of the company, then the department of the requester will also be entered
* A list of products will be displayed, where the user will select the product for the change request
* A list of releases for that product will be displayed, where the user will select the release for the change request
* The date and description of the request will now be entered

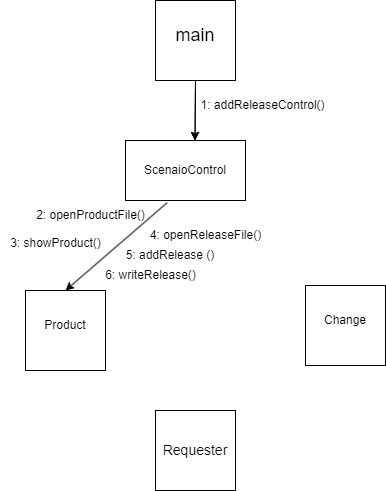
1. **Create product**

****

Scenario control

* User wants to add a new product into the system
* The user will be prompted to enter the name of the product
* If the length of the product name is not between 1-10 characters, an error message will be thrown
* The user will then be asked whether they want to try again or go back to the main menu

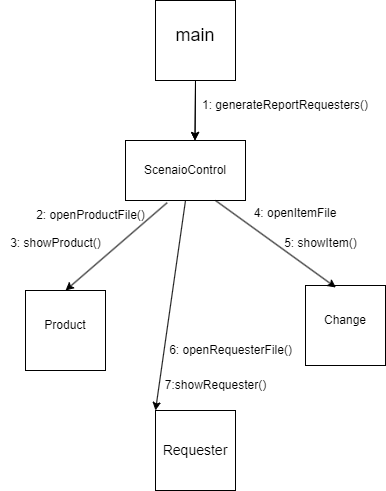
1. **Create product release**

****

Scenario control

* The user wants to create a new release for a product
* The user will be shown a list of products and select the one that needs a new release
* The user will be prompted to enter the release name of the product
* The user will be prompted to enter the date of the release
* A error message will occur if any of the above three values do not fit the required format

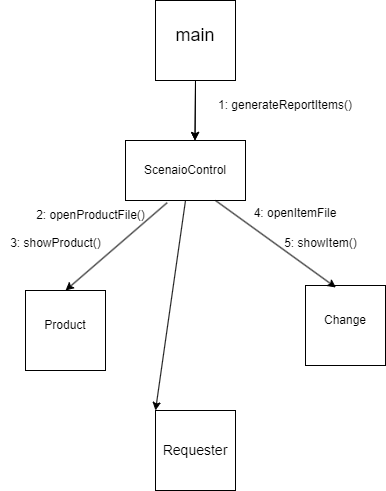
1. **Report of people who need to know a particular change**

****

Scenario control

* User wants to see a report of people who need to see a particular change
* The user will be shown a list of products and they will need to enter the product
* The user will be shown a list of change items for that product and they will need to enter the item they want.

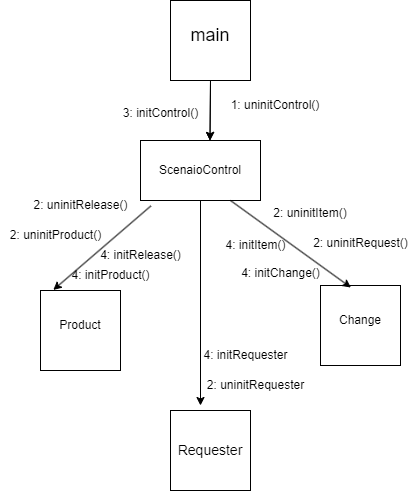
1. **Report of unresolved change items for a particular product**

****

Scenario control

* User wants to see a list of change items that have not been resolved
* The user will be shown a list of products and they will need to enter the product

1. **System restart**

****

Scenario control

* User wants to restart the system
* The issue tracking system will first shutdown
* Main module tells the scenario control module to shutdown by calling uninitControl().
* The scenario module then tells the product and release module, requester, and change request and items module to uninitialize by calling uninitProduct() and uninitRelease(), uninitRequester(), and uninitItem() and uninitRequest() respectively.
* The issue tracker system will start up again
* The main module tells the scenario control module to initialize by calling initControl().
* The scenario module then tells the product and release module, requester, and change request and items module to initialize by calling initProduct() and initRelease(), initRequester(), and initItem() and initRequest() respectively.

# 4.0: Comparison with Alternate Design

This space is intentionally blank in this release of the architectural design document.

# 5.0: Initial Interface Design

Main calls a mid-level control module to carry out the program’s processes. The mid-level control module works with three lower level modules that can interact with the system’s database files.

Each of our 3 application modules (product and release, requester, and change request and items) requires the operations of reading and writing to their relevant files on the disk. Since the files must be accessible in the case of no instances and must not be duplicated for each instance, we use a static attribute to represent it. All functions that deal with opening, reading, writing to, and closing the files are static for the same reason.

# 6.0: Detailed Interface Design

The detailed interface design is not a part of this document but can be found in the separate main and header modules that have been released alongside.

# 7.0: Testing

## 7.1 Functional Test Case

We expect the most widely used feature of our program to be the creation of a new Change Request, since it will be customer service representatives and software engineers alike. It therefore seems fitting to begin our functional testing from there. Below is a test case for the same.

Preconditions:

1. You are in the directory that contains the main files and the makefile
2. The program contains at least one product release (which in turn necessitates the existence of a product), one requester, and one change item . For testing purposes, the product will be called **rBusy**, and the release **rBusy1.0**. There will be exactly 1 change requester (**John Doe**) and change item (**start-up glitch**) registered in the system.
3. The change request to be entered is not identical to any existing ones

To start the program, type “**make ITS**” in the terminal and then press **Enter (↵)** on your keyboard.

You will be presented with the main menu:

Main Menu:

[1] Handle Change Request

[2] Update Change Item

[3] Manage Products

[4] Generate Reports

[0] Exit

Type **1** followed by **Enter (↵),** and the process for creating a new change request will begin.

You will first be asked to choose the requester from our database (or create one if they don’t exist). You will be presented with this menu :

Handle Change Request:

Select Requester:

NAME: Phone: Email:

[1] JOHN DOE +1 123-456-1891 johndoe@gmail.com

[0] Back [C] next page [N] Create New Requester

Type **1** followed by **Enter (↵),** and your requester will be selected. You will then be

asked to select a product. You will be presented with a list of all products to choose from.

This is what you will be presented with:

Handle Change Request:

Select Product Needing Change:

Product Name

[1] rBusy

[0] Back [C] Next Page

Type **1** followed by **Enter (↵),** and the product will be selected. Once you’ve selected a product, you must select the release. You will be presented with all the releases for that product, and be asked to choose one. You will be presented with:

Handle Change Request:

Select Release Needing Change:

Release Name

[1] rBusy1.0

[0] Back [C] Next Page

Type **1** followed by **Enter (↵),** and the release will have been selected. You must now enter the date of the day you are making this request. For the purposes of this test, let that date be the **3rd of July, 2024**. You will be prompted to enter the date like this:

Handle Change Request:

Get Date:

[0] Back

Field Requirements:

(YYYY-MM-DD: Y, M, D, are numeric characters, include dashes)

Enter Date: **2024-07-03**

With that, the date will have been entered into the system. You will now be prompted to enter a description of the change you are requesting. You will be prompted to do so by:

Handle Change Request:

Get Description:

[0] Back

Field Requirements:

(1-30 alphanumeric characters)

Enter Description: **start-up glitch**

You will have successfully entered the description. You must now see if a change item with a similar description exists. If it does, you will select it and the request will have been created! If it does not, a new item will be created first. You will see this on your screen:

Handle Change Request:

Does a Change Item described by “start-up glitch” Exist?:

Description ID

[1] START-UP GLITCH 1

[0] Back [C] Next Page [N] Create New Change Item

Type **1** followed by **Enter (↵),** and the change item will be selected. Your request will now be added to the system and you will see a message on the screen indicating that before you are taken back to the main menu :

Change Request Added

Main Menu:

[1] Handle Change Request

[2] Update Change Item

[3] Manage Products

[4] Generate Reports

[0] Exit

End of test.

## 7.2 Performance Test Case

The goal of this test is to measure the response time for the generation of a report on all requesters that requested a specific change.

Preconditions:

1. You are in the directory that contains the main files and the makefile
2. A change item exists for which you wish to generate a list of all requesters
3. There are 50 change requests in the system. The system also has one product **rBusy** and 16 changeItems (with changeIDs **1 through 16**, description same as id, status **In-progress** and release **rBusy2.0**). There is one requester for the change item with id **5**, and the requester is **John Doe**.
4. You have access to a stopwatch (physical or software)

To start the program, type “**make ITS**” in the terminal and then press **Enter (↵)** on your keyboard.

You will be presented with the main menu:

Main Menu:

[1] Handle Change Request

[2] Update Change Item

[3] Manage Products

[4] Generate Reports

[0] Exit

Type **4** followed by **Enter (↵),** and the process generating the report will begin.

You will be first asked to select the product for which the change item is.

Generate Reports:

Select a Change Item From What Product?:

Product Name:

[1] rBusy

[0] Back [00] Back to Main Menu [C] Next Page

Type **1** followed by **Enter (↵),** and you will then be asked to select a change item form the list of all items for that product.

Generate Reports:

Generate Report for Which Change Item?:

ID Description Status Release

[1] 1 1 IN PROGRESS rBusy2.0

[2] 2 2 IN PROGRESS rBusy2.0

[3] 3 3 IN PROGRESS rBusy2.0

[4] 4 4 IN PROGRESS rBusy2.0

[5] 5 5 IN PROGRESS rBusy2.0

[6] 6 6 IN PROGRESS rBusy2.0

[7] 7 7 IN PROGRESS rBusy2.0

[8] 8 8 IN PROGRESS rBusy2.0

[9] 9 9 IN PROGRESS rBusy2.0

[10] 10 10 IN PROGRESS rBusy2.0

[11] 11 11 IN PROGRESS rBusy2.0

[12] 12 12 IN PROGRESS rBusy2.0

[13] 13 13 IN PROGRESS rBusy2.0

[14] 14 14 IN PROGRESS rBusy2.0

[15] 15 15 IN PROGRESS rBusy2.0

[16] 16 16 IN PROGRESS rBusy2.0

[0] Back [00] Back to Main Menu [C] Next Page

Enter **5** and press **Enter (↵),** and you will be asked to confirm the change item you selected

Generate Reports:

Change Item: “1”

Product: “rBusy”

Status: “In Progress”  
Description: “1”

Priority: “1”

Release: “rBusy2.0”

Release Date: “2024-07-03”

Create a Report for Change Item “ChangeID”?

[y/n]: **y**

Press **Enter (↵)** enter to your response at the exact same time as when you start your stopwatch.

You will see:

Release Date: “RELEASE DATE” Product: “PRODUCT” Release: “RELEASE”

Name Phone Email

John Doe +1 123-456-7890 johndoe@ca

[00] Main Menu [C] Next Page [N] New Report

Stop the stopwatch once this is displayed.

This should tell you how long it takes to go through all requests and find the requester information for the ones that are valid in terms of product and change item.

The response time should be 0.2 seconds

End of Test.

## 7.3 Stress Test Case

This test will look at the program’s behavior when an invalid input has been entered. We will test this through the trying to create a new product.

Precondition:

1. You are in the directory that contains the main files and the makefile

To start the program, type “**make ITS**” in the terminal and then press **Enter (↵)** on your

keyboard.

You will be presented with the main menu:

Main Menu:

[1] Handle Change Request

[2] Update Change Item

[3] Manage Products

[4] Generate Reports

[0] Exit

Type **3** followed by **Enter (↵),** and the you will be taken to a sub menu

Manage Products:

[1] Create Product

[2] Create Product Release

[0] Back

Type **1** followed by **Enter (↵),** and the product creation process will begin

Add a Product:

Define Product Name:

[0] Back  
  
Field Requirements:  
(1-10 alphanumeric characters)

Enter Product name: **remarkablyBusy**

The product name entered goes beyond the 10 character field requirement and will cause

the program to throw an error message and will ask you if you wish to try again or go

back to the main menu.

The value that you have entered exceeds the bounds set by the data format requirements.

Enter Y to try again. Enter N to return to the Main Menu.

Enter **N** and return back to the main menu

End of Test

# 8.0: Coding Convention

#### 8.1

The module will be organised as so. The module name, description, and version history will be written at the top of each module file. Then include statements, then variables, then structures, and finally prototypes. In a class public variables, structures, and prototypes, will be written preceding private variables, structures, and prototypes

They will be written as:

/\* ModuleName.filetype

description:

...

version history:

...\*/

//==================

#include <this>

#include “that.h”

//==================

var1 var1Name;  
var2 var2Name;

//==================

returntype functionPrototype1(

...

);

#### 8.2

The version history will follow these conventions:

* Will be written in descending chronological order; the most recent revision will be written first..
* Entries will be separated by a newline.
* The first line of each entry will contain the version number, the author, and the date of that version’s release as YY/MM/DD.
* Each change will be written on its own line singly indented, the reason for each change will be written below on its own line doubly indented.

They will be written as

version history:

ver2 -YY/MM/DD, modified by authorB

-made change

-made change because..., or to...

-also made change

-”...”

ver1 -YY/MM/DD, original by authorA

#### 8.3

Function prototypes will follow these conventions:

* The first line will contain the function name, keywords (const, static, etc.), and return type.
* Each variable will be written on its own line with a multi-line comment written above, describing that variable, and stating whether that variable is used as an input, output, or input and output of that function.
* The function will have a multiline comment below the prototype providing a description of the function, the preconditions, the postconditions, and information about the exceptions the function raises.

They will be written as:

keywords returntype functionName(

/\* variable description:

...

used for: (input | output | input/output) \*/

var v1,

/\* variable description

...

... \*/

var v2,

...

/\* ...

...

... \*/

var vN

);

/\* description of the function behaviour

...

preconditions:

...

postconditions:

...

exceptions raised:

... \*/

#### 8.4

* All sections of the module and functions will be divided by a line in the form of the comment “//==================” : 2 forward slashes and 18 equal signs. This comment will be preceded by an empty line.
* Function names and variables will be written in camelCase, constant variables will be written in ALL\_CAPS\_WITH\_UNDERSCORES.
* Spaces will surround operators e.g., “int b = a \* 12 + 1”
* In classes, public members will be written before private members.
* Code block indentations will be four spaces wide.
* Curly braces are on their own line. If, else, then, and loop control structures must have curly braces.