Software Quality and Testing

Assignment 1

Group	The Unit Testers
Leith Hobson	gushoble@student.gu.se
Chrysostomos Tsagkidis	gustsach@student.gu.se
Krasen Anatoliev Parvanov	gusparkr@student.gu.se

Program:	Software Engineering and Management, BSc
Course:	DIT635: Software Quality and Testing
Date:	2020/02/16
Number of pages:	26

Table of contents

1. Problem 1

The following properties emerge during the development of the software for an Automatic Teller Machine (ATM). For each of the following, identify whether it is a correctness, robustness, or safety property. Briefly justify your decision.

1. If the network connection is interrupted, the session shall be immediately terminated with an appropriate error message. No funds shall be dispensed, and no changes shall be made to a user's account.

Robustness - When the network connection is interrupted, despite the fact that part of the software's operation has failed, no funds were dispensed to the user and thus the software still remained "correct", but instead, it failed "gracefully".

2. The account identifier of the account loaded by the utility shall exactly match the account identifier of the input debit card.

Correctness - In the event that the card reader incorrectly reads the card information (such that it does not exactly match the account identifier of the input card), this is simply incorrect, and not partially correct, therefore, this is a correctness property.

3. The amount of money requested by the user shall only be debited from their account following a confirmation from the ATM of the successful withdrawal of physical funds from the machine.

Safety - If the physical funds are not dispensed due to a technical, or other error, the transaction must not be recorded as successful (a hazard), and the funds must not be debited from the user's account. This is a safety property, and, if this condition were to be violated, it could open the ATM operator to legal and other costs to compensate the user.

4. If no physical money remains in the unit following a completed transaction, the utility shall cease standard operation and display an error message until a manual override is initiated.

Robustness - This feature adds to the ATM robustness since the requirement above takes into consideration an event where the system can stop to function as it was designed to and presents a countermeasure in the form of a new state. That ensures "graceful degradation" of the ATM service.

2. Problem 2

Under what circumstances can making a system safer also make it less reliable? Briefly
explain with an example.

There are many examples where increasing the safety of software can make it less reliable, because increasing safety can decrease the performance. Adding safety functionality in the form of requirements and controls to the system can reduce its availability and reliability. An example of this would be in the case of a self-driving car, where the main purpose of the car is to get the occupant from point A to point B, however, increasing obstacle avoidance sensitivity, could ultimately prevent the car from operating efficiently, and fulfilling its main purpose.

3. Problem 3

- reliability
  NFR1: The system should identify new aircraft in the controlled airspace within 5 seconds, in 99% of cases, and within 10 seconds in 99.99% of cases.

- availability
  NFR2: The system will implement redundancy on all levels, so as to maintain system availability of 99.999%.

- performance
  NFR3: The response time from the radar network should be less than 1s even under extreme workloads (50% more than the baseload of 100 flights per hour).

- scalability
  NFR4: The software shall be able to retain its average communication response time with the pilots lower than 2 seconds, therefore in the event of 10% increase of the number of concurrent aeroplanes that it needs to keep track of, it needs scale to maintain acceptable response times.

- security
  NFR5: A safety warning functionality shall be included in the system, providing a warning alert within 5 seconds if two or more aeroplanes are in close distance of each other.

Quality Scenarios:

Reliability

Overview: The system must continue recognising aircraft entering the airspace, even in extreme peak workload conditions.

System State: The system is lightly loaded at the current system load.

System Environment: The environment is operating normally, and planes are entering the monitored airspace steadily, at a rate of approximately 100 planes per hour. The system is recognising all-new aircraft in under 5 seconds.

External Stimuli: Suddenly, the radar network reports an steady increase of 100 new planes entering the airspace within 5 minutes interval.

System Response: The system indicates the increase to the air-traffic-controller with notification, the system increases the refresh and update intervals of displaying the list of all airplanes in the controlled space, and continues to operate normally.

Response Measure: The system recognises 99% of the planes within 5 seconds of their arrival, and all planes within 10 seconds of their arrival in the airspace.

Availability

Overview: Check if the system continues functioning and being available after one of the monitoring aircraft components fails.

System State: The system is functioning normally. The monitoring components are sending data to the air traffic control.

System Environment: The system is in a normal and functional state with monitoring components tracing 110 planes.

External Stimuli: One of the air traffic control monitoring components fails.

System Response: An error signal is sent to the maintenance team. A backup component is activated within 10s in place of the one gone down. The air-traffic-monitoring display is refreshed within 1s after receiving a signal from the new component. The system continues to function properly.

Response Measure: After failure an error signal is sent within 3s, and a back-up component is activated and fully functional within 15s.

Performance

Overview: Radar's network response time in the event of 50% increase in the current number of aeroplanes being monitored.

System State: The system is functioning normally. The monitoring components are sending data to the air traffic control system, and the system has a low load state.

System Environment: The system has planes arriving steadily at 90 planes per hour, into the monitored airspace.

External Stimulus: Over the course of 20 minutes, the number of planes arriving into the monitored airspace increases to 150 planes per hour, which is then sustained for an hour, before reducing to 100.

System Response: The system recognizes the higher load on the radar's network based on the extra processing load that has been created by the respective increase of the planes entering the airspace, therefore a second load balancer steps in. In that case, there will be two load balancers in Active/Active mode, in order to handle and distribute the respective requests.

Response Measure: The latency between the air traffic control system, and the radar network will be measured continuously and the response time will remain within 1s.

Scalability

Overview: How the system behaves in the event of an unexpected increase in the operating load.

System State: The system is functioning normally. The monitoring components are sending data to the air traffic control system, and the system has a low load state.

System Environment: The environment is operating normally, and planes are entering the monitored airspace steadily, at a rate of approximately 100 planes per hour. The operator's average response time is 1s.

External Stimulus: The number of aeroplanes entering the airspace starts to steadily increase by 10%. The increase continues for a period of 24 hours.

System Response: The system detects the heightened system load, and creates an additional processing node to handle the increased load.

Response Measure: The system recognizes the demand for more processing units in order to respond to the higher demand for resources, therefore it creates an additional processing node successfully. Given the increase of the aeroplanes, after scaling, the communication response time stays within 2s.

Security

Overview: Check the system's response in the event of a possible impending collision between two close aeroplanes.

System State: The system is under heavy load. The monitoring components are sending data to the air traffic control system at an increased frequency.

System Environment: The system is keeping track of 40% more planes than the normal baseload.

External Stimulus: Suddenly, there is an indication coming after the analysis of the radar's data that there are two planes in very close distance between each other.

System Response: The system manages to recognize the pattern behind a possible collision between two planes that are close to each other, therefore a safety warning is being shown at the operator's screens, in order for them to prioritize their decision making over that incident. In addition to that, warning notifications are being sent to the pilots of the respective planes. The system refresh frequency is increased.

Response Measure: The safety warning message was shown under the threshold of 5 seconds, since the time that the system recognized the pattern behind a possible collision. Pilots warning notifications are being sent within the same threshold.

4. Problem 4

4.1 Test Plan

4.1.1 Introduction and overview

The testing team will be testing the CoffeeMaker software. CoffeeMaker is intended to keep track of coffee recipes and the machine's inventory. The software further allows for coffee to be ordered, and then dispensed by the CoffeeMaker hardware.
In order to carry out this testing, the testing team will evaluate the CoffeeMaker software created by the CSC department at North Carolina State University (NCSU) through a series of automated, and exploratory tests, as laid out in this testing document.

4.1.2 Scope

The scope of this testing case will cover the verification of the main functionality of each individual module. Therefore, the respective components should be tested in terms of both their core functionality and also to verify the absence of any possible out-of-boundary cases, such as user input data validation.

Outside the scope will be any hardware related cases, such as any possible compatibility issues, as well as requirement's validation. Stress test and load test of the software to find how it performs under huge load will not be covered as well. Furthermore, outside the scope are integration, system and acceptance test.

4.1.3 Test phases

The testing shall be performed in the following phases:

Static analysis: Main goal is to analyse the code and to verify that it follows Java Conventions and Standards. Try to find programming errors.

Unit test: Main goal is to write, automate and perform tests on the individual modules and components in the system. In order to verify the software functionality and find as many bugs as possible.

Functionality and Exploratory Black box test: During this phase the team shall conduct black box manual functionality tests in order to verify the main software features and to find potential bugs. Exploratory sessions are to be performed as well by the end of the testing phase.

Out of scope are Integration, System and Acceptance testing.

4.1.4 Test Strategy and Design

In order to verify the functionality of the Coffee Maker software and to be able to locate as many errors as possible the team will design and perform testing on each module of the application. All possible input values are to be taken into consideration while designing the test data.

4.1.5.1 Strategy

The following components are to be tested: Recipe, RecipeBook, CoffeeMaker, Inventory, Main. Each logical method in the components above is to be tested with white and black box testing techniques to ensure their functionality. Test cases are to be written in regards to every method that the components have, not only verifying the functionality but also testing with invalid data.
These white box test cases are to be automated and executed with the JUnit framework which will also provide a base for future regression testing.
For the black box approach the team will conduct functional testing by running the software in a controlled environment(IntelliJ) by which the testing team aims to verify the base functionality and find potential logical bugs missed by the unit testing. Exploratory testing will be performed in sessions of 30 minutes to give further depth and understanding of the software.
Since higher-level testing is out of the scope the testing approach will be focused on unit module functionality verification and correctness and complex testing scenarios are not to be considered.

4.1.5.2 Test case design

Test cases are to be designed for every single method in the application. Test data for the unit testing is to be created by analysing the source code and identifying. equivalent classes and performing boundary analysis. The test cases are to be written from the perspective of possible functionality of the system. The unit test cases are to be written and represented in Java using JUnit 5 framework. The results of the test cases are to be shown with the following format in Excel for traceability.

TC ID	Name Description	Test Object	Method to be tested	Status	Comment	Date

4.1.5 Risk analysis and failure classification

Risk analysis

Software related risks need to be assessed, with respect to their cause, their impact and the likelihood that they might happen. To begin with, a detailed evaluation of the code needs to be conducted, as well as specifying the relationships among the respective classes, in order to fully comprehend and understand the interactions of these components. The team acknowledges the potential risks of not having access to the hardware, in order to test the software in that regard and also not having a complete requirements specification document.

Failure classification

Test failures will be classified by the testing team into four categories, namely Critical, Severe, Minor and Cosmetic depending on the severity of the failure, and criticality of the function that is failing.

4.1.6 Start and End Criteria

Start criteria:

The testing will begin once the software has been delivered by the development team, as ready for FAT, and this testing document has been accepted by the client (the classroom).

Exit criteria:

The exit criteria for the test will be an actual run rate of 100% over tests with 100% method coverage.

4.1.7 Test deliverables

The testing team aims to deliver the following artefacts in order to support the developing process of the software in question:

Test plan - this document. Aims to give an overview of the testing process.

Test cases - Detailed description of the tests to be performed.

Test report - Detailed description of the results of the executed tests.

Test failure report - Detailed description of the faults found during the test execution.

Unit test suite - The automize test unit suite. Written by the test team, hosted on GitHub.

4.1.9 Testing Tools

To be able to plan, trace and execute the tests described in the test suite the following software tools are to be used by the testing team:

JUnit 5

Java 11

Maven

Git for version control

IntelliJ

Excel

4.1.10 Testing Environment

The tests are to be executed in a controlled environment. The team is going to use IntelliJ to perform unit white-box tests and black box exploratory test sessions.

4.2 Test Cases and Report

4.2.1 Overview

All the unit tests and their respective results are represented in the table in 4.2.2, every system module was tested to verify its functionality. While constructing the test cases the team took into consideration both positive and negative input values and outcomes, in order to find as many bugs as possible as well as to verify the application. All failures were analysed in order to find the root coast and the failure report is seen in 4.2.3. All unit tests are available in the repository test suite submitted in addition to this document.

A total of 91 test cases were designed and performed during the unit test cycle with Intellij and JUnit 5 as testing environment. All the unit tests were written with JUnitTest, and have @test annotation. In order to run a test, you can open one of the ClassNameTest files, and then click on the 'bubble' in the margin, next to the method signature. In order to run all the CoffeeMaker unit test, you should right-click on the edu.ncsu.csc326.coffeemaker package, and then select "Run 'Tests in 'edu.ncsu.csc326.coffeemaker'' with Coverage''

The total coverage achieved by the automated unit tests is as follows:

Coverage Summary for Package: edu.ncsu.csc326.coffeemaker			
Package	**Class, %**	**Method, %**	**Line, %**
edu.ncsu.csc326.coffeemaker	80% (4/ 5)	80.4% (45/ 56)	60.1% (217/ 361)
Class	**Class, %**	**Method, %**	**Line, %**
CoffeeMaker	100% (1/ 1)	100% (8/ 8)	100% (23/ 23)
Inventory	100% (1/ 1)	100% (16/ 16)	100% (89/ 89)
Main	0% (0/ 1)	0% (0/ 11)	0% (0/ 142)
Recipe	100% (1/ 1)	100% (16/ 16)	97.5% (78/ 80)
RecipeBook	100% (1/ 1)	100% (5/ 5)	100% (27/ 27)

Main menu functionality was tested manually, hence the zero percentage coverage for the main class in the table above.

4.2.2 Test Cases and Results

For simplicity of the table below the preconditions and the exact test data used are not included. Those details can be seen in the test suite itself. Where each unit test has a respective test case ID (TC ID).

TC ID	Name	Test Object	Method Tested	Expected Result	Actual Result	Status
1	Get amount of chocolate with positive input value	Recipe class	getAmtChocolate	Integer of value 5	The return value is the same as the expected result value	PASSED
2	Set amount of chocolate with positive input value	Recipe class	setAmtChocolate	Integer of value 12	The set value is the same as the expected one	PASSED
3	Set amount of chocolate with negative input value	Recipe class	setAmtChocolate	RecipeException	Exception was thrown as expected, due to an invalid input value	PASSED
4	Set amount of chocolate with invalid input value	Recipe class	setAmtChocolate	RecipeException	Exception was thrown as expected, due to an invalid input value	PASSED
5	Set amount of chocolate with no input value	Recipe class	setAmtChocolate	RecipeException	Exception was thrown as expected, due to an invalid input value	PASSED
6	Get amount of coffee with positive input value	Recipe class	getAmtCoffee	Integer of value 10	The return value is the same as the expected result value	PASSED
7	Set amount of coffee with positive input value	Recipe class	setAmtCoffee	Integer of value 17	The set value is the same as the expected one	PASSED
8	Set amount of coffee with negative input value	Recipe class	setAmtCoffee	RecipeException	Exception was thrown as expected, due to an invalid input value	PASSED
9	Set amount of coffee with invalid input value	Recipe class	setAmtCoffee	RecipeException	Exception was thrown as expected, due to an invalid input value	PASSED
10	Set amount of coffee with no input value	Recipe class	setAmtCoffee	RecipeException	Exception was thrown as expected, due to an invalid input	PASSED

					value	
11	Get amount of milk with positive input value	Recipe class	getAmtMilk	Integer of value 8	The return value is the same as the expected result value	PASSED
12	Set amount of milk with positive input value	Recipe class	setAmtMilk	Integer of value 9	The set value is the same as the expected one	PASSED
13	Set amount of milk with negative input value	Recipe class	setAmtMilk	RecipeException	Exception was thrown as expected, due to an invalid input value	PASSED
14	Set amount of milk with invalid input value	Recipe class	setAmtMilk	RecipeException	Exception was thrown as expected, due to an invalid input value	PASSED
15	Set amount of milk with no input value	Recipe class	setAmtMilk	RecipeException	Exception was thrown as expected, due to an invalid input value	PASSED
16	Get amount of sugar with positive input value	Recipe class	setAmtSugar	Integer of value 7	The return value is the same as the expected result value	PASSED
17	Set amount of sugar with positive input value	Recipe class	setAmtSugar	Integer of value 11	The set value is the same as the expected one	PASSED
18	Set amount of sugar with negative input value	Recipe class	setAmtSugar	RecipeException	Exception was thrown as expected, due to an invalid input value	PASSED
19	Set amount of sugar with invalid input value	Recipe class	setAmtSugar	RecipeException	Exception was thrown as expected, due to an invalid input value	PASSED
20	Set amount of sugar with no input value	Recipe class	setAmtSugat	RecipeException	Exception was thrown as expected, due to an invalid input value	PASSED
21	Get the name of the recipe	Recipe class	getName	String of value "name"	The return value is the same as the expected result value	PASSED
22	Set the name of the recipe	Recipe class	setName	String of value "New"	The set value is the same as the expected one	PASSED

23	Set null as the name of the recipe	Recipe class	setNamel	The name is not null	The set value was null, thus it wasn't accepted	PASSED
24	Get the price of the recipe	Recipe class	getPrice	Integer of value 50	The return value is the same as the expected result value	PASSED
25	Set the price for the recipe	Recipe class	setPrice	Integer of value 55	The set value is the same as the expected one	PASSED
26	Set a negative number for the price of the recipe	Recipe class	setPrice	RecipeException	Exception was thrown as expected, due to an invalid input value	PASSED
27	Set invalid input for the price of the recipe	Recipe class	setPrice	RecipeException	Exception was thrown as expected, due to an invalid input value	PASSED
28	Set empty input for the price of the recipe	Recipe class	setPrice	RecipeException	Exception was thrown as expected, due to an invalid input value	PASSED
29	Compare the toString output with the return of getName method of the Recipe class	Recipe class	toString	The name of the recipe	The toString output is the same as the expected result value	PASSED
30	Verification of the hashCode implementation	Recipe class	hashCode	The comparison returns TRUE	The hashCode implementation shows the same results as the expected ones	PASSED
31	Compare two different recipies if they are equal	Recipe class	equals	The two recipes are equal to each other	Two different cases with valid input data were equal to the expected value	PASSED
32	Compare a recipe with a null one	Recipe class	equals	The comparison returns FALSE	The recipe was not equal to a null one	PASSED
33	Compare the names of two recipes if they are equal	Recipe class	equals	The comparison returns FALSE	The comparison between two different names of the respective recipes returns false	PASSED
34	Compare the recipe with another object	Recipe class	equals	The comparison	The comparison of equality	PASSED

				returns FALSE	between a recipe object with one of different type returns false	
35	Add recipe to the recipebook array	RecipeBook class	addRecipe	Recipe is added to the recipe book array	Recipe is added to the recipe book array	PASSED
36	Adding a recipe that is already in the list	RecipeBook class	addRecipe	Recipe is not added to the recipe book array	Recipe is not added to the book	PASSED
37	Retrieve all the recipes from the book array	RecipeBook class	getRecipe	The recipe book are returned	The recipe book are returned	PASSED
38	Remove recipe from the list	RecipeBook class	deleteRecipe	A recipe was removed	A recipe was removed	FAILED
39	Delete a recipe that is empty	RecipeBook class	deleteRecipe	Null value is returned	Null value returned	PASSED
40	Test the edit recipe funcution	RecipeBook class	editRecipe	The recipe is edited/ updated	After the update the name of the updated recipe was empty string	FAILED
41	Test editing recipe that is null	RecipeBook class	editRecipe	Null value is returned	Null value returned	PASSED
42	Test delete recipe at positon that doesn't exist	RecipeBook class	deleteRecipe	Null value is returned	Index out of bound exception	FAILED
43	Editing a recipe with a null value	RecipeBook class	editRecipe	Null value is returned	Null point exception	FAILED
44	Test that the all recipe are retrived	CoffeeMaker class	getRecipes	All recipes are returned	All recipes are returned	PASSED
45	Delete a recipe	CoffeeMaker class	deleteRecipe	A recipe was removed	A recipe was removed	FAILED
46	Test delete a recipe that is null	CoffeeMaker class	deleteRecipe	Null value is returned	Null value returned	PASSED
47	Test add recipe functunality	CoffeeMaker class	addRecipe	Recipe is added	Recipe is added	PASSED
48	Adding a recipe that is already added	CoffeeMaker class	addRecipe	Recipe is not added	Recipe was not added	PASSED
49	Editing a recipe with a new recipe values	CoffeeMaker class	editRecipe	Recipe is updated	Recipe name was not updated it was a empty string.	FAILED
50	Edit a recipe that doesn't exist/ is empty	CoffeeMaker class	editRecipe	No recipe found and null is returned	No recipe was found and null was returned	PASSED
51	Check values in	CoffeeMaker	checkInvento	Values in list	Values were as	PASSED

	inventory list	class	ry	are the same as pre-defined. Milk 10; Chocolate 10; Coffe 10; Sugar 10	expected. Milk 10; Chocolate 10; Coffe 10; Sugar 10	
52	Check if the values in the inventory are null	CoffeeMaker class	checkInventory	Values in the inventory are not null	Values in the inventory are not null	PASSED
53	Check if the inventory can be added	CoffeeMaker class	addInventory	Values were as expected. Milk 20; Chocolate 20; Coffe 20; Sugar 20	Values were as expected. Milk 20; Chocolate 20; Coffe 20; Sugar 20	FAILED
54	Check if the inventory can be updated with invalid input	CoffeeMaker class	addInventory	Given an invalid combination of input the inventory is not updated and throws an exception	Sugar was updated with a negative value	FAILED
55	Check if the inventory can be updated with a Zero value	CoffeeMaker class	addInventory	Expected that given 0 the inventory should not be updated	The inventory values were set to 0	PASSED
56	Check if make coffee function works and returns change	CoffeeMaker class	makeCoffee	50 is returned as change given 100 as input and 50 as price	Change of 50 was returned	PASSED
57	Check if all change is returned if there is no recipe	CoffeeMaker class	makeCoffee	Given 100 as a input and with no recipe a change of 100 is returned.	Change of 100 was returned	PASSED
58	Try making a coffee from a recipe that is not in the list	CoffeeMaker class	makeCoffee	Given an input of 100 and trying to purchase recipe at a position that does not exist a change of 100 is returned	Index out of bounds exception	FAILED
59	Try to purchase a	CoffeeMaker	makeCoffee	Given an	The change is -3	FAILED

	coffee with a negative amount	class		input of -3 the change is 0		
60	Try to purchase a coffee with a smaller than price amount	CoffeeMaker class	makeCoffee	Given an input of 10 and price being 50 a change of 10 is returned	Change of 10 was returned	PASSED
61	Try to purchase a coffee with no inventory	CoffeeMaker class	makeCoffee	Given an input 100 and no inventory left a change of 100 is returned	Change of 100 was returned	PASSED
62	Get the amount of chocolate in the inventory	Inventory class	getChocolate	The return value is equal to the valid one that was specified as test data	The return value is the same as the expected result value	PASSED
63	Set amount of chocolate with valid input	Inventory class	setChocolate	The set value is equal to the random generated one	The set value is the same as the expected one	PASSED
64	Add the specified amount of chocolate units in the inventory to the current one	Inventory class	addChocolat e	The random generated amount of chocolate was added to the previous value one	The amount of chocolate was added to the inventory	PASSED
65	Add a negative amount of chocolate units in the inventory to the current one	Inventory class	addChocolat e	InventoryExc eption should be thrown	An exception was thrown as expected, due to an invalid input value	PASSED
66	Add an invalid amount of chocolate units in the inventory to the current one	Inventory class	addChocolat e	InventoryExc eption should be thrown	An exception was thrown as expected, due to an invalid input value	PASSED
67	Get the amount of coffee in the inventory	Inventory class	getCoffee	The return value is equal to the valid one that was specified as test data	The return value is the same as the expected result value	PASSED

68	Set amount of coffee with valid input	Inventory class	setCoffee	The set value is equal to the randomly generated one	The set value is the same as the expected one	PASSED
69	Add the specified amount of coffee units in the inventory to the current one	Inventory class	addCoffee	The randomly generated amount of coffee was added to the previous value one	The amount of coffee was added to the inventory	PASSED
70	Add a negative amount of coffee units in the inventory to the current one	Inventory class	addCoffee	InventoryException should be thrown	An exception was thrown as expected, due to an invalid input value	PASSED
71	Add an invalid amount of coffee units in the inventory to the current one	Inventory class	addCoffee	InventoryException should be thrown	An exception was thrown as expected, due to an invalid input value	PASSED
72	Get the amount of milk in the inventory	Inventory class	getMilk	The return value is equal to the valid one that was specified as test data	The return value is the same as the expected result value	PASSED
73	Set amount of milk with valid input	Inventory class	setMilk	The set value is equal to the randomly generated one	The set value is the same as the expected one	PASSED
74	Add the specified amount of milk units in the inventory to the current one	Inventory class	addMilk	The random generated amount of milk was added to the previous value one	The amount of milk was added to the inventory	PASSED
75	Add a negative amount of milk units in the inventory to the current one	Inventory class	addMilk	InventoryException should be thrown	An exception was thrown as expected, due to an invalid input value	PASSED
76	Add an invalid amount of milk units in the inventory to the current one	Inventory class	addMilk	InventoryException should be thrown	An exception was thrown as expected, due to an invalid input value	PASSED

77	Get the amount of sugar in the inventory	Inventory class	getSugar	The return value is equal to the random generated one	The return value is the same as the expected result value	PASSED
78	Set amount of sugar with valid input	Inventory class	setSugar	The set value is equal to the random generated one	The set value is the same as the expected one	PASSED
79	Add the specified amount of sugar units in the inventory to the current one	Inventory class	addSugar	The randomly generated amount of milk was added to the previous value one	InventoryException was thrown given a positive integer	FAILED
80	Add a negative amount of sugar units in the inventory to the current one	Inventory class	addSugar	InventoryException should be thrown in the event of negative provided input	Negative amount of sugar was added to the inventory	FAILED
81	Add an invalid amount of sugar units in the inventory to the current one	Inventory class	addSugar	InventoryException should be thrown	An exception was thrown as expected, due to an invalid input value	PASSED
82	Check if there are enough ingredients to make the beverage	Inventory class	useIngredients	True for enough amount of ingredients and False if the recipe exceeds the amount of ingredients	True for comparison of enough ingredients. Falce for the rest of the cases.	PASSED
83	Remove the respective ingredients for the recipe to be made	Inventory class	useIngredients	Expected the inventory to be decreased by 10 units per ingredient, And 5 units to be returned/left for every ingredient	25 units of chocolate was returned	FAILED
84	Return a string with	Inventory	ToString	The toString	The toString	PASSED

	the current contents of the inventory	class		should return the following: Coffee: 15 Milk: 15 Sugar: 15 Chocolate: 15	output is the same as the expected result value	
85	Testing main menu basic functionality - run and exit	Main class/ Main menu	Manual test	The system starts and exits by entering 0 without issues.	The system started and exited as expected, without any issues	PASSED
86	Testing add a recipe functionality in the main menu	Main class/ Main menu	Manual test	The system creates a new recipe successfully.	The system created a new recipe successfully.	PASSED
87	Testing delete a recipe functionality in the main menu	Main class/ Main menu	Manual test	The recipe is deleted by the system.	The values of the recipe are deleted, but not the recipe object, and so the recipe still exists.	FAILED
88	Testing edit a recipe functionality in the main menu	Main class/ Main menu	Manual test	The recipe is updated according to the new values that are entered by the user.	The recipe values were updated correctly, however, the recipe name was removed.	FAILED
89	Testing add inventory functionality in the main menu	Main class/ Main menu	Manual test	The inventory levels for each of the inventory items is increased according to the values entered.	The inventory levels for coffee and milk were increased, however, sugar and chocolate remained unchanged, despite the user input.	FAILED
90	Testing check inventory functionality in the main menu	Main class/ Main menu	Manual test	The system prints the respective values of the items that are available currently in the inventory.	The inventory of the coffee maker is being returned, along with each of the respective values of the items.	PASSED
91	Testing make coffee functionality in main menu	Main class/ Main menu	Manual test	The system should make the requested	The system made the requested coffee, and	FAILED

				coffee and return any change due to the user. Further, the inventory levels should be adjusted, according to the inventory that was used to produce the purchased coffee.	returned the change to the user. However, the level of coffee in the inventory was increased, and not decreased.	

4.2.3 Failure Report and Recommended fixes

After executing the tests above a total of 17 failures were found as seen in the table below. In addition to that after static analysis of the code the team found a potential issue in the constructor of the Inventory class where the inventory is set to have default values of 15 for every single attribute. The testing team believes that it will potentially cause issues and it recommended to have default values to be set to zero.

The team conducted an analysis in order to find the cause of the bugs and provide the developing team with feedback and recommendation of possible fixes.

Test with ID 42. Trying to delete a recipe at a position that doesn't exist throws an ArrayOutOfBounds exception.
The method deleteRecipe in Recipe book class does not verify if the position of recipe received is higher than the array length which can produce an Array out of bounds exception. A recommended fix for that is adding a if statement that verifies if the position received is higher than the array length and if true returning null.

Test with ID 43: When trying to update a recipe in recipeBook with a new value of null, it is expected that editRecipe will return null, however, this is not the case, and editRecipe instead throws a NullPointerException. A recommended fix for this bug is to replace the contents of the if statement on line 75 of RecipeBook with "recipeArray[recipeToEdit] != null && newRecipe != null" as this will result in the method checking for null values of newRecipe, and returning null in the event that this occurs.

Tests with ID 40, 49 and 88. After updating a recipe the name is set to an empty string instead of being updated.
That is seen at line 77 in RecipeBook class method editRecipe(). A recommended fix is to change newRecipe.setName(""); to newRecipe.setName(recipeName);

Tests with ID’s 89, 53, 54, 79 and 80 : When trying to either add a positive or a negative amount of sugar units in the inventory to the current amount one, an Inventory exception is thrown for the former ca 535353se while no exception is thrown for the latter, as it was expected. A recommended fix for those cases would be a change in the if statement that is responsible for the respective verification of whether the amount of sugar should be accepted as valid, in line 183 of *Inventory Class*, from “<=” as it is right now to “>=”.

Test with ID 58: While trying making coffee from a recipe that currently does not exists by calling the makeCoffee method of the class CoffeeMaker, the amount of mount that was requested for that transaction should be given as change back to the user, instead currently the method returns an “Index out of bounds exception”. A recommended fix for that case would be to try and catch the exception in the method and in that case return a message specifying the reason along with the respective line of code for returning the change back to the user (“change = amtPaid;”).

Test with ID 59: When calling the makeCoffee method with a negative amtPaid, the expected result is that the machine does not make the coffee, and returns 0 change, however, the makeCoffee method returned a negative change, resulting in the failed test case. The recommended fix, in this case, is to add two lines “if(change<0)
change = 0;” at line 103 in CoffeeMaker.java (directly before the change is returned).

Tests with ID 83 and 91: When calling useIngredients, this should deplete each of the inventory levels by the respective amount, that is being utilised to produce the requested drink. However, this test case failed because the amount of coffee in the inventory is increased, rather than being depleted. A recommended fix for the base would be simply to change the line 221 in class Inventory “Inventory.coffee += r.getAmtCoffee();” to be “Inventory.coffee **-=** r.getAmtCoffee();” this way, the used ingredients will be removed from the inventory, and not erroneously added to the inventory.

Tests with ID 38, 87 and 45: When calling the delete recipe methods the recipe is deleted but instead of being set to null the position in the array is set to new Recipe() an empty object which shows at the menu when you run the application and can coast issues. A fix to that can be setting the deleted object in the array to null at line 60 in RecipeBook class instead of new Recipe().

TC ID	Name	Test Object	Method Tested	Expected Result	Actual Result	Status
38	Remove recipe from the list	RecipeBook class	deleteRecipe	A recipe was removed	A recipe was removed	FAILED
40	Test the edit recipe function	RecipeBook class	editRecipe	The recipe is edited/ updated	After the update, the name of the updated recipe was an empty string	FAILED
42	Test delete recipe at position that doesn't exist	RecipeBook class	deleteRecipy	A null value is returned	Index out of bounds exception	FAILED
43	Editing a recipe given a null value	RecipeBook class	editRecipe	Null value is returned	Null point exception	FAILED
45	Delete a recipe	CoffeeMaker class	deleteRecipe	A recipe was removed	A recipe was removed	FAILED
49	Editing a recipe with a new recipe values	coffeeMaker class	editRecipe	Recipe is updated	Recipe name was not updated, it was an empty string.	FAILED
53	Check if the inventory can be added	CoffeeMaker class	addInventory	Values were as expected. Milk 20; Chocolate 20; Coffe 20; Sugar 20	Values were as expected. Milk 20; Chocolate 20; Coffe 20; Sugar 20	FAILED
54	Check if the inventory can be updated with invalid input	coffeeMaker class	addInventory	Given an invalid combination of input, the inventory is not updated and throws exception	Sugar was updated with a negative value	FAILED
58	Try making a coffee from a recipe that is not in the list	coffeeMaker class	makeCoffee	Given an input of 100 and trying to purchase recipe at position that does not exist a change of 100 is returned	Index out of bound exception	FAILED
59	Try to purchase a coffee with a negative amount	coffeeMaker class	makeCoffee	Given a input of -3 the change is 0	The change is -3	FAILED

79	Add the specified amount of sugar units in the inventory to the current one	Inventory class	addSugar	The randomly generated amount of milk was added to the previous value one	InventoryException was thrown given a positive integer	FAILED
80	Add a negative amount of sugar units in the inventory to the current one	Inventory class	addSugar	InventoryException should be thrown in the event of negative provided input	Negative amount of sugar was added to the inventory	FAILED
83	Remove the respective ingredients for the recipe to be made	Inventory class	useIngredients	Expected the inventory to be decreased by 10 units per ingredient, And 5 units to be returned/left for every ingredient	25 units of chocolate was returned	FAILED
87	Testing delete a recipe functionality in main menu	Main class/ Main menu	Manual test	The recipe is deleted by the system.	The values of the recipe are deleted, but not the recipe object, and so the recipe still exists.	FAILED
88	Testing edit a recipe functionality in main menu	Main class/ Main menu	Manual test	The recipe is updated according to the new values that are entered by the user.	The recipe values were updated correctly, however, the recipe name was removed.	FAILED
89	Testing add inventory functionality in main menu	Main class/ Main menu	Manual test	The inventory levels for each of the inventory items is increased according to the values entered.	The inventory levels for coffee and milk were increased, however, sugar and chocolate remained unchanged, despite the user input.	FAILED

91	Testing make coffee functionality in main menu	Main class/ Main menu	Manual test	The system should make the requested coffee and return any change due to the user. Further, the inventory levels should be adjusted, according to the inventory that was used to produce the purchased coffee.	The system made the requested coffee, and returned the change to the user. However, the level of coffee in the inventory was increased, and not decreased.	FAILED