IWS2 Projects A.A. 2019-2020

Andrea Graziani
Università degli Studi di Roma "Tor Vergata"
Laurea Magistrale in Ingegneria Informatica
Rome, Italy
andrea.graziani93@outlook.it

***Index Terms*—component, formatting, style, styling, insert**

I. Introduction

II. Adequacy Criteria

A. Statement coverage

Our test set T is able to cover up to 85 statements out of a total of 89, reaching, according to Sonarcloud report, a *statement coverage* equal to **0,955** (**95,5%**).

However we believe that `DiskChecker` class contains several statements that we can consider as **unreachable** because they fall on an *infeasible path*, that is a path that would never be reached by our test set T with any type of input data; to be more precise, we believe that `setDiskSpaceThreshold(float, float)` method contains up to 4 unreachable statements.

Generally, in order to identify unreachable statements, drawing the flow-graph of the developed code and finding out the path that would never be reached is required, however we can consider as unreachable all statements of `setDiskSpaceThreshold(float, float)` method owing to following reasons:

1) `setDiskSpaceThreshold(float, float)` method is **never used** in the project, that is it is never called by any other method.
2) `setDiskSpaceThreshold(float, float)` method has **no access modifier** (*package private*) which means, according to Java language specification, that it is only accessible within classes in the same package, therefore that method is not visible by any test set (we are assuming that test code, generally located into `/test` directory, is always included into a different package respect to application code, which is conversely located into `/main` directory)

Therefore, we can conclude by stating that:

$$\textbf{Statement Coverage} = \frac{|S_c|}{|S_e| - |S_i|} = \frac{85}{89 - 4} = 1 = 100\% \tag{1}$$

Since statement coverage of T is 1, we can consider T as **adequate with respect to the statement coverage criterion**.

B. Decision coverage (Branch decision coverage)

According to [1], a *decision* is considered covered if the flow of control has been diverted to all possible destinations that correspond to this decision, i.e. all outcomes of the decision have been taken. This implies that, for example, the expression in the `if` or `while` statement has evaluated to `true` in some execution and to `false` in the same or another execution. Note that each `if` and each `while` contribute to *one* decision whereas a `switch` may contribute to more than one.

According to our analysis, `DiskChecker` class contains 14 possible decisions, therefore $|D_e| = 14$ where D_e is the set of decisions in the program. Our test set T can cover all decision, therefore $|D_c| = 14$, where D_c is the set of decisions covered.

$$\textbf{Decision Coverage} = \frac{|D_c|}{|D_e| - |D_i|} = \frac{14}{14 - 0} = 1 = 100\% \tag{2}$$

Since decision coverage of T is 1, we can consider T as **adequate with respect to the decision coverage criterion**.

C. Condition coverage

Unlike decision coverage, condition coverage ensures that each simple condition within a compound condition has assumed both values `true` and `false`.

Le be given:

- C_e the set of simple conditions in the program.
- C_c the set of of simple conditions covered by our test set T.
- C_i the set infeasible simple conditions.

According to `JaCoCo` and sonarcloud reports, our test set T covers up to 46 simple conditions, out of a total of 46, while $|C_i| = 0$. Therefore we can say that:

$$\textbf{Condition Coverage} = \frac{|C_c|}{|C_e| - |C_i|} = \frac{46}{46 - 0} = 1 = 100\% \tag{3}$$

Since condition coverage is equal to 1, our test set T is adequate with respect to the condition coverage criterion.

References

[1] A. Spillner, T. Linz, and H. Schaefer, *Software Testing Foundations: A Study Guide for the Certified Tester Exam*, 3rd ed. Rocky Nook, 2011.

TABLE I
EQUIVALENCE CLASSES AND REPRESENTATIVES OF DISKCHECKER METHOD

Parameter	Equivalence Classes		Representatives
threshold	vEC_1:	$0 < \texttt{warnThreshold} \leq x < 1$	$0, 5$
	iEC_1:	$x \leq 0$	0
	iEC_2:	$x \geq 1$	1
	iEC_3:	$0 < x < \texttt{warnThreshold} < 1$	1
warnThreshold	vEC_1:	$0 < x \leq \texttt{threshold} < 1$	$0, 5$
	iEC_1:	$x \leq 0$	0
	iEC_2:	$1 > x > \texttt{threshold} > 0$	0
	iEC_3:	$x \geq 1$	1

TABLE II
EQUIVALENCE CLASSES AND REPRESENTATIVES OF CHECKDIR METHOD

Parameter	Equivalence Classes		Representatives
dir	vEC_1:	Valid `File` object representing a valid (existent, readable and writeable) directory.	`new File("/home")`
	iEC_1:	Valid `File` object which not represents a directory, that is it can represent regular file, symbolic link, character device file etc.	`new File("./regularFile.txt")`
	iEC_2:	Valid `File` object representing a not existent and make-able directory.	`new File("./root")`
	iEC_3:	Valid `File` object representing a not existent and not make-able directory.	`new File("/root/")`
	iEC_4:	Valid `File` object representing an existent, readable, not writeable directory.	`new File("/root")`
	iEC_5:	Valid `File` object representing an existent, not readable, writeable directory.	`new File("/root")`
	iEC_6:	A `null` object.	`null`

TABLE III
EQUIVALENCE CLASSES AND REPRESENTATIVES OF GETTOTALDISKUSAGE, GETTOTALFREESPACE, GETTOTALDISKSPACE METHODS

Parameter	Equivalence Classes		Representatives
dir	vEC_1:	A valid non-empty `List<File>` object containing `File` objects every of which represents a valid (existent, readable and writeable) directory.	-
	iEC_1:	A validgfdg non-empty `List<File>` object containing `File` objects every of which represents a not existent directory.	-
	iEC_2:	A valid non-empty `List<File>` object containing `File` objects every of which represents an existent, not readable and not writeable directory.	-
	iEC_3:	A valid non-empty `List<File>` object containing `File` objects every of which not represents a directory, that is it can represent regular file, symbolic link, character device file etc.	-
	iEC_4:	A valid empty `List<File>` object.	`new ArrayList<>()`
	iEC_5:	A `null` object.	`null`