4.1 Unit-Testen Praxis

1 Überlegen Sie sich eine Strategie um die Klasse zu testen und notieren Sie sich Ihre Vorgehensweise (Wie gehen Sie vor um gut zu testen?, Welche Strategien wenden Sie an?, Welche Verfahren eignen sich hier gut?,...).

Um die Klasse ordentlich durchzutesten, gehe Ich erstmal die gesamte Klasse ordentlich durch, um ein Grundverständnis der gesamten Klasse zu erhalten. Als nächstes sehe ich mir die Methoden der Klasse an, da wir diese durchtesten wollen. Je nachdem um was für eine Methode es sich handelt gehe Ich unterschiedlich an die Sache ran.

Als erstes Beispiel die Methode isEmpty. isEmpty ist vom Datentyp boolean also kann es zwei Ergebnisse ausgeben. True und False, und das teste ich anschließend.

Die Methode Size ist vom Datentyp int und gibt die Anzahl an Elementen im Buffer aus. Die teste ich mit einem Leeren Buffer und wenn mein Buffer in diesem Fall voll ist, also 5 Elemente.

Als nächstes teste ich die Methode Push. Ich verwende push zwar davor bereits, teste es hier aber explizit mit einem Mittelwert ich habe hierfür mich für 2 Elemente entschieden. wird ein Item zu viel gepusht wird eine Exception geworfen da der Buffer bereits voll ist.

Pop ist als nächstes dran. Hier gibt es mehr Möglichkeiten als bei Push. Erst einmal das alles nach Plan verläuft und das der Buffer kleiner wird indem sich der count verkleinert. Dies überprüfen wird mit size. Die Möglichkeit das pop verwendet wird obwohl sich kein Object im Buffer befindet und eine Exception geworfen wird und, dass das gepopte Object auch richtig returned wird.

Kommen wir nun zum Iterator. hasNext überprüfen wir erstmal auf seine Funktion einmal auf True und auf False, je nachdem ob die Bedingung i < count gegeben ist oder nicht.

Remove hat nur eine Funktion und diese ist es eine Exception zu werfen, wenn sie verwendet wird, da pop für das Rauswerfen eines Objects zuständig sein soll.

Und als letztes next welche wir chronologisch erstmal auf die Exception prüfen also wenn hasnext nicht erfüllt ist. sollten sich allerdings Elemente im RingBuffer befinden und dadurch next durchlaufen wird es auf die richtige Reihenfolge der Ausgabe (wegen Return wert) und auf die Bewegung der nächsten Position getesten.

In diesem Fall wurde durch das gründliche Durchtesten der Methoden bereits eine volle Line Coverage und Mutation Coverage als auch Test Strength erreicht wodurch ein extra FullCoverage- und FullMutationScoretest nicht nötig ist.

5. Wie hängen Code-Coverage Metriken und Mutation Score zusammen?

Code-Coverage und Mutation Score hängen nur indirekt voneinander ab. Zwar kann durch eine hohe Code-Coverage ein einigermaßen hoher Mutation Score erzielt werden, allerdings muss das nicht direkt was damit zu tun haben. Da bei der Mutation der Code so verändert wird, das aus true, false wird aus +, - oder * ein :, Daraufhin wird getestet ob das Programm immer noch weiterhin funktioniert obwohl diese Änderungen stattgefunden haben.