Aufgabe 10

Nathan Ritter
5566519
Lorenzo Tecchia
5581906
2023.06.27

Contents

Listings

Chapter 1

Task 1

(a) In essence the process of Validation is asking yourself the question: ”Am I building the right product?” where the process of Verification is asking yourself: ”Am I building the product right?”. Verification involves checking that the software conforms to its specification. The aim of validation is more general, it is to ensure that the software system meets the customer’s expectations. [**Sommerville:2004aa**]

(b) A test case is a set of actions performed on a system to determine if it satisfies software requirements and functions correctly.[**techtargetWhatTest**]

(c) An error is usually an human mistake that resides inside a software system, a defect is the imperfection in the software that happens when the human mistake is made, finally a failure is what a software incurs when trying to execute with a defect. [**toolsqaWhatDifference**]

(d) Solution:

1. The review is the process of examining the code of a Software also relating the code to its requirements and documentation; the structural testing or White Box testing is a process in which the the structure of often time just units in the software, it is like the process of review an inspection of the source code, but it doesn’t have any relation to the documentation and to the requirements. [**Sommerville:2004aa**][**wikipediaWhiteboxTesting**]
2. Load testing is usually used, when testing a software, to find the threshold for the software to still work under a certain amount of load of work. Instead stress testing is used to examine how the software reacts to an excessive amount of load, so when the threshold is passed.[**bmcPerformanceTesting**]
3. Testing is the process of verifying and validating that a software or application is bug-free, meets the technical requirements as guided by its design and development, and meets the user requirements effectively and efficiently by handling all the exceptional and boundary. Debugging is the

process of fixing a bug in the software. It can be defined as identifying, analysing, and removing errors.[**geeksforgeeks**]

4. Function testing is the process of verifying the functionalities of the software, acceptance testing is the process whereby the system is tested using customer data to check that it meets the customer's real needs.[**Sommerville:2004aa**]
5. Top-down testing is a type of incremental testing approach in which testing is done by integrating or joining two or more modules by moving down from top to bottom through control flow of architecture structure. Bottom-up is the inverse process: testing single modules and then going up putting together said modules.[**geeksTopBottom**]

Chapter 2

Task 2

(a) OCL of klassifiziereDreieck

```
context Dreieck::klassifiziereDreieck(seite1 : int, seite2 : int, seite3 : int)
 pre: seite1 > 0 and seite2 > 0 and seite3 > 0 and not seite1.oclIsUndefined() and not seite2.oclIsUndefined() and not seite3.oclIsUndefined()
```

(b) Test Cases in Java

`@Test` `public void testgleichseitig(){` `DreiecksArt result = klassifiziereDreieck(4,4,4);` `assertEquals("Pruefe ob es Gleichseitigkeit erkennt", DriecksArt.Gliechseitig, result);` `}`	Need to check if it can recognize equallateral triangles.
`@Test` `public void testrechtwinklig(){` `DreiecksArt result = klassifiziereDreieck(3,4,5);` `assertEquals("Pruefe ob es Rechtwinkligkeit erkennt", DriecksArt.Rechtwinklig, result);` `}`	Need to check if it can recognize right triangles.

```java
@Test
public void testgleichschenklig(){
   DreiecksArt result = klassifiziereDreieck(4,4,5);
   assertEquals("Pruefe ob es Gleichschenkligkeit erkennt", DriecksArt.Gleichschenklig, result);
}
```

Need to check if it can recognize isosceles triangles.

```java
@Test
public void testnulleingabe(){
   DreiecksArt result = klassifiziereDreieck(0,4,5);
   assertEquals("Pruefe ob es Schlechte Laengen erkennt", DriecksArt.Normal, result);
}
```

Need to check if it can recognize invalid lengths.

```java
@Test
public void testnegativeingabe(){
   DreiecksArt result = klassifiziereDreieck(3,-4,5);
   assertEquals("Pruefe ob es Schlechte negative Laengen erkennt", DriecksArt.Normal, result);
}
```

Need to check if it can recognize invalid lengths that are negative.

```java
@Test
public void testNaneingabe(){
   DreiecksArt result = klassifiziereDreieck(3,4,NAN);
   assertEquals("Pruefe ob es Schlechte Laengen erkennt die keine Zahlen sind", DriecksArt.Normal, result);
}
```

Need to check if it can recognize lengths that are not Numbers.

```java
@Test
public void testNaneingabe(){
   DreiecksArt result = klassifiziereDreieck(3,null,5);
   assertEquals("Pruefe ob es Schlechte Laengen erkennt die keine Werte haben", DriecksArt.Normal, result);
}
```

Need to check if it can recognize lengths that are null.

```java
@Test
public void testgleichseitig(){
   DreiecksArt result = klassifiziereDreieck(null,null,null)
       ;
   assertEquals("Pruefe ob es Gleichseitigkeit bei
       Falscheingabe erkennt", DriecksArt.Normal, result);
}
```

Need to check if it will recognize equallateral triangles even if it has bad input.

(c) There are 4 branches in the code:

1. if all if-statements are false it executes int quad1 = seite1 * seite1; , int quad2 = seite2 * seite2; , int quad3 = seite3 * seite3; and return DreieckArt.Normal;
2. if all sides are the same length it executes return DreieckArt.Gleichseitig;
3. if two sides are the same length it executes return DreieckArt.Gleichschenklig;
4. if the triangle is a right triangle it executes return DreieckArt.Rechtwinklig; after executing int quad1 = seite1 * seite1; , int quad2 = seite2 * seite2; and int quad3 = seite3 * seite3;

```java
@Test
public void testgleichseitig(){
   DreiecksArt result = klassifiziereDreieck(4,4,4);
   assertEquals("Pruefe ob es Gleichseitigkeit erkennt",
       DriecksArt.Gliechseitig, result);
}
```

testing branch 2

```java
@Test
public void testrechtwinklig(){
   DreiecksArt result = klassifiziereDreieck(3,4,5);
   assertEquals("Pruefe ob es Rechtwinkligkeit erkennt",
       DriecksArt.Rechtwinklig, result);
}
```

Testing Branch 4

```java
@Test
public void testgleichschenklig(){
   DreiecksArt result = klassifiziereDreieck(4,4,5);
   assertEquals("Pruefe ob es Gleichschenkligkeit erkennt",
       DriecksArt.Gleichschenklig, result);
}
```

Testing branch 3

```java
@Test
public void testgleichschenklig(){
   DreiecksArt result = klassifiziereDreieck(2,4,5);
   assertEquals("Pruefe ob es Gleichschenkligkeit erkennt",
       DriecksArt.Gleichschenklig, result);
}
```

Testing branch 1

(d) The Programm all of the tests successfully, execpt the one that checks what it returns if all lenghts are bad values but the same type of bad value and the one that checks if it recognize negative lengths correctly. The Program doesn't handel bad inputs very well, it returns that a triangle is normal even when one of the inputs is null, Nan or 0 for a side of the triangel, or if all the inputs are the same even if they are bad values it will return that the triangle is equallateral or if two sides have the same bad value it will return that the triangle is an isosceles triangle which could be misleading and even if the sides of the triangel are negative it will still recognize a right triangle.

(e) Yes, The Coverage criteria C0 and C1 were achieved by the tests because all Code was executed once and all conditions were true and false at least once. The Coverage criteria C1 and C0 don't seem very suitable for this type of problem because the code doesn't handel all possible conditions with conditional statements.

(f) The structure test also don't seem very suitable because the code didn't anticipate all types inputs it could get. The Functionstest on the otherhand was suitable because it did cover all possible inputs that the code needs to be able to handel.

(g) If null and Nan are the inputs for 2 sides e.g klassifiziereDreieck(Nan,null,5) it is likely that the program will crash when it tries to do arithmetic on these values.

Chapter 3

Task 3

(a) When working with multiple people on a bigger Software project, role assignment is crucial. Often the developers team gets to be split into to part, the part that does the actual developing and the testing team. The testing team develops automated test cases to adapt to the code keeping up with the other team every step of the developing process. Actuating a sort of ”divide and conquer” process. Testing your own code would mean losing this ”divide and conquer” principle so that the developing process accompanied by the testing process would rather take double the time. Moreover developers often carry on the ”how to make this” mindset, when instead a tester often uses the mindset: ”how to break this”. Developers are not that much closer to the client so when it comes to properly and finally test a software someone else much more closer to the client, like the QA team, should instead test the Software. This rules makes even more sense if we consider that often times developers cannot detach themselves from the idea of how they coded the software, and therefore would be bias on how they would test the code (testing the code exactly how it was developed). [**stackexchangeDevelopersTest**][**Sommerville:2004aa**]

(b) Test Driven Development (TDD) is the process of interleaving development with test. You develop the code incrementally, along with a set of tests for that increment. You don't start working on the next increment until the code that you have developed passes all of its tests. Test-driven development was introduced as part of the XP agile development method. However, it has now gained mainstream acceptance and may be used in both agile and plan-based processes. There are four fundamentals steps of TDD:

1. Identification of the increment of functionality required
2. Writing the test for said incremental functionality and implement it into an automated test
3. Running the test to fail to verify the branch reachability.
4. Implementing the function for which the test was developed and run the test.

5. Repeat for the next functionality.

TDD helps the developers to change hats on every step of the process, since the testing gets implemented before the actual coding is done, the bias of the developer is almost null and therefore force the teams to think both ways. Also TDD help decrease the phenomenon of Regression Testing, in which the functionalities before implemented have to be tested again when the new functionalities get implemented into the project, thus reducing the amount of time for testing in general. It is also claimed that use of TDD encourages better structuring of a program and improved code quality. [**Sommerville:2004aa**]