UNIVERSITÄT
LEIPZIG

Softwaretechnik 2024/25 – Übung 08

Prof. Dr. Norbert Siegmund
M.Sc. Stefan Jahns

Aufgabe 1: Grundlagen

a) Welche Fehlerklassen gibt es, geben Sie für drei der Fehlerklassen Beispiele.

Fehlerklasse	Beschreibung
Transient	Tritt nur bei bestimmten Eingaben auf
Permanent	Tritt bei allen Eingaben auf
Recoverable	System erholt sich ohne Intervention eines Nutzers
Unrecoverable	Nutzerintervention ist benötigt zur Wiederherstellung des Systems
Non-corrupting Fehler	korrumpiert nicht die Daten
Corrupting	Fehler korrumpiert die Daten

Aufgabe 1: Grundlagen

b) Was verbirgt sich hinter dem Begriff Regression Testing?

- Nach jeder Änderung werden alle Tests ausgeführt
- Stellt sicher, dass alles, was vor der Änderung funktioniert hat, auch nach der Änderung noch funktioniert
- Voraussetzung: Tests müssen deterministisch und wiederholbar sein

Aufgabe 1: Grundlagen

c) Nennen Sie die entscheidenden Vor- und Nachteile von

I. Testing

II. Model Checking.

Model Checking

System wird als Modell in einer formalen Sprache beschrieben

- Vollständige Überprüfung gegen das Modell
- Beweise möglich
- Modelle sind in der Erstellung sehr komplex und zeitaufwändig

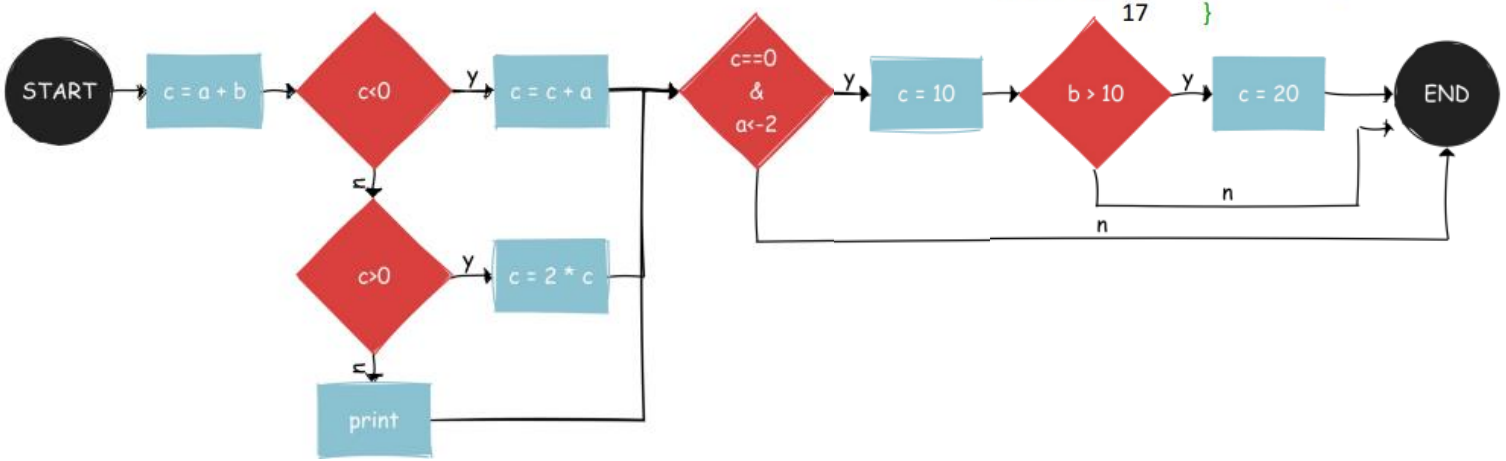
Testing

- Einfacher umzusetzen als Model Checking
- Fehler können gefunden, aber korrekte Funktionsweise nicht bewiesen werden

Aufgabe 2: Kontrollflussgraph

a) Zeichnen Sie den Kontrollflussgraphen für folgende Java-Methode.

```
1  public static double compute(double a, double b) {
2          double c = a + b;
3          if (c < 0) {
4              c = c + a;
5          } else if (c > 0) {
6              c = 2 * c;
7          } else {
8              java.lang.System.out.print("c is 0!");
9          }
10         if (c == 0.0 && a < -2.0) {
11             c = 10.0;
12             if (b > 10.0) {
13                 c = 20.0;
14             }
15         }
16         return c;
17     }
```



Aufgabe 2: Testabdeckung

b) Wie viele Tests werden für einen C0-Test benötigt? Geben Sie ein minimales Test-Set an.

Mindestens 3 verschiedene Inputs:

1. a+b < 0
2. a+b > 0
3. a+b = 0 ^ a < -2 ^ b > 10

Bspw:

1. a=-50, b= 5
2. a= 50, b= 500
3. a=-50, b= 50

```
1  public static double compute(double a, double b) {
2          double c = a + b;
3          if (c < 0) {
4                  c = c + a;
5          } else if (c > 0) {
6                  c = 2 * c;
7          } else {
8                  java.lang.System.out.print("c is 0!");
9          }
10         if (c == 0.0 && a < -2.0) {
11                 c = 10.0;
12                 if (b > 10.0) {
13                         c = 20.0;
14                 }
15         }
16         return c;
17 }
```

Aufgabe 2: Testabdeckung

b) Wie viele Tests werden für einen C1-Test benötigt? Geben Sie ein minimales Test-Set an.

Mindestens 4 verschiedene Inputs:

1. a+b < 0
2. a+b > 0
3. a+b = 0 ^ a < -2 ^ b > 10
4. a+b = 0, a < -2, b <= 10

Bspw:

1. a=-50, b= 5
2. a= 50, b= 500
3. a=-50, b= 50
4. a=-10, b= 10

```
1  public static double compute(double a, double b) {
2          double c = a + b;
3          if (c < 0) {
4                  c = c + a;
5          } else if (c > 0) {
6                  c = 2 * c;
7          } else {
8                  java.lang.System.out.print("c is 0!");
9          }
10         if (c == 0.0 && a < -2.0) {
11                 c = 10.0;
12                 if (b > 10.0) {
13                         c = 20.0;
14                 }
15         }
16         return c;
17 }
```

Aufgabe 3: Testabdeckung

```
public class Date {
        /**
         * @param year a valid year
         * @param month a month position from 1 to 12
         * @param day a valid day position within the month
         * @throws InvalidDateException if date is invalid
         */
        public Date(int year, int month, int day) throws InvalidDateException        { ... }


        /**
         * A method to modify dates without crossing month boundaries.
         *
         * @param numDays the number of days to add to the current date - should        work with
positive and negative values, probably maybe.
         * @return a new Date object representing the result
         * @throws InvalidDateException if resulting date is not in the same month
         */
        public Date modifyDaysWithinMonth(int numDays) throws InvalidDateException
        { ... }

        /**
         * @return true if the current year is a leap year
         */
        public boolean isLeapYear()
        { ... }
}
```

<u>Äquivalenzklassentests</u>

Date(2024,12,10) ok
Date(2024,13,10) -> InvDate
Date(2024,12,10).mod..(1) ok
Date(2024,12,10).mod..(100) -> InvDate
Date(2025,12,10).isLeapYear() -> False
Date(2024,12,10).isLeapYear() -> True

Domain-Wissen

Date(400,12,10).isLeapYear() -> True
Date(100,12,10).isLeapYear() -> False
Date(2025,02,29) -> InvDate
Date(2024,02,29) ok