LU02c - Test Coverage

Siehe auch BrowserStack - Test Coverage Techniques

Testabdeckung bezeichnet das Verhältnis zwischen den möglichen und den tatsächlich getesteten Teilen einer Software. Beispielsweise misst die Codeabdeckung das Verhältnis zwischen allen vorhandenen Codezeilen und den Codezeilen, die in allen Testfällen ausgeführt wurden.

Testabdeckung kann als Metrik in jeder Teststufe verwendet werden. Beim Unit-Test wird unter anderem der Prozentsatz der ausgeführten Codezeilen überprüft.

Vorteile von Testabdeckung

Wenn wir die Testabdeckung unserer Anwendung kontinuierlich messen, ergeben sich einige Vorteile:

- **Früheres Erkennen**: Wir erkennen Lücken in unserer Anwendung oder in den Testfällen früher. Je früher wir ein Problem erkennen, desto einfacher ist es, es zu beheben.
- **Redundanzen eliminieren**: Die Testabdeckung hilft uns, Redundanzen zu erkennen und zu beseitigen.
- **Weniger Aufwand**: Eine bessere Testabdeckung bedeutet weniger Fehler in späteren Teststufen und in der Produktion. Dadurch verringert sich der Aufwand für die Fehlerbehebung und die Qualitätssicherung.

Code-Abdeckung (Code Coverage)

Code Coverage ist für das White Box Testing in Unit- und Integrationstests von großer Bedeutung. Mit Hilfe von Werkzeugen wird gemessen, wie vollständig der Code getestet wurde. Eine hohe Code Coverage erhöht die Wahrscheinlichkeit, dass Fehler frühzeitig erkannt und behoben werden. Darüber hinaus helfen Kennzahlen zur Codeabdeckung zu erkennen, welche zusätzlichen Testfälle benötigt werden.

Die Code Coverage setzt sich aus mehreren Ebenen zusammen.

Anweisungsabdeckung (Statement Coverage)

Die Anweisungsabdeckung misst, wieviel Prozent der Anweisungen im Sourcecode ausgeführt wurden.

Zweigabdeckung (Branch Coverage)

Die Zweigabdeckung misst, ob bei jeder Entscheidung innerhalb des Codes (if, while, until, switch, ...) jeder Zweig einmal durchgeführt wurde.

`num1 = int(input('Number >'))`
`if num1 > 0:`
`  print('valid')`
	`num1 = int(input('Number >'))`
	`if num1 > 0:`
	`  print('valid')`
	`else:`
	`  print('invalid')`

Da es im Code nur einen Branch für `if` gibt, reicht ein einziger Testfall um eine Branch Coverage von 100% zu erreichen.	In diesem Beispielcode brauchen wir 2 Testfälle für `if` und `else`.

Bedingungsabdeckung (Condition Coverage)

Die Bedingungsabdeckung misst, ob jede (Teil-)Bedingung einmal mit `true` und einmal mit `false` ausgewertet wurde.

`num1 = int(input('Number >'))`
`if num1 > 0:`
`  print('valid')`
	`num1 = int(input('Number >'))`
	`if num1 > 0:`
	`  print('valid')`
	`else:`
	`  print('invalid')`

Auch wenn es im linken Beispiel keinen `else`-Zweig gibt, müssen wir für die Decision Coverage beide möglichen Ergebnisse für die Bedingung prüfen. Beide Beispiele benötigen also 2 Testfälle.	

```
num1 = int(input('Number >'))
if num1 > 0 and num1 < 100:
    print('valid')
```

In diesem Beispiel haben wir eine verknüpfte Bedingung. Somit müssen wir alle 4 möglichen Kombinationen von `true` und `false` der Teilbedingungen testen:

num1 > 0	**num1 < 100**
true	true
true	false
false	true
false	false

Funktionsabdeckung (Function Coverage)

Die Funktionsabdeckung misst die Anzahl der ausgeführten Funktionen im Verhältnis zur Anzahl aller Funktionen.

M450-LU02

Marcel Suter

From:
https://wiki.bzz.ch/ - **BZZ - Modulwiki**

Permanent link:
https://wiki.bzz.ch/modul/m450/learningunits/lu02/coverage?rev=1727772848

Last update: **2024/10/01 10:54**