

# LU02c - Testabdeckung

Siehe auch [BrowserStack - Test Coverage Techniques](#)



Testabdeckung bezeichnet das Verhältnis zwischen den möglichen und den tatsächlich getesteten Teilen einer Software. Als Beispiel misst die Codeabdeckung das Verhältnis zwischen allen vorhandenen Codezeilen und den Codezeilen die im Rahmen aller Testfälle ausgeführt wurden.

Testabdeckung kann als Metrik in jeder Teststufe eingesetzt werden. Beim Unit Test wird unter anderem der Prozentsatz der ausgeführten Codezeilen geprüft.

## Vorteile der Testabdeckung

Wenn wir die Testabdeckung unserer Applikation fortlaufend messen, ergeben sich einige Vorteile:

- **Frühes Erkennen:** Wir entdecken früher Lücken in unserer Applikation oder in den Testfällen. Je früher wir ein Problem erkennen, desto einfacher kann es beseitigt werden.
- **Redundanzen eliminieren:** Die Testabdeckung hilft uns Redundanzen zu erkennen und zu eliminieren.
- **Weniger Aufwand:** Eine bessere Testabdeckung bedeutet weniger Fehler in späteren Teststufen und in der Produktion. Dadurch verringert sich der Aufwand für die Fehlerbeseitigung und die Qualitätssicherung.

## Codeabdeckung (Code Coverage)

Die Codeabdeckung 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 Codeabdeckung erhöht die Wahrscheinlichkeit, dass Fehler früh erkannt und beseitigt werden. Zudem helfen die Kennzahlen der Codeabdeckung zu erkennen, welche zusätzlichen Testfälle benötigt werden.

Die Codeabdeckung 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.

<pre>num1 = int(input('Number &gt;')) if num1 &gt; 0:     print('valid')</pre>	<pre>num1 = int(input('Number &gt;'))     if num1 &gt; 0:         print('valid')     else:         print('invalid')</pre>
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.

<pre>num1 = int(input('Number &gt;')) if num1 &gt; 0:     print('valid')</pre>	<pre>num1 = int(input('Number &gt;'))     if num1 &gt; 0:         print('valid')     else:         print('invalid')</pre>
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.



Marcel Suter

From:

<https://wiki.bzz.ch/> - **BZZ - Modulwiki**

Permanent link:

<https://wiki.bzz.ch/modul/m450/learningunits/lu02/coverage?rev=1711631267>

Last update: **2024/03/28 14:07**

