

**Test-Cases**

**Burgstaller Raphael,   
Haider Sebastian,   
Serschen Anton**

**5BHIFS**

**Qualitätsmangement**

**2017/2018**

Inhalt

[Abstract 3](#_Toc501092419)

[Einleitung 4](#_Toc501092420)

[Test-Levels 4](#_Toc501092421)

[Black- und White-Box-Testing 5](#_Toc501092422)

[Black-Box-Testing 5](#_Toc501092423)

[White-Box-Testing 6](#_Toc501092424)

[Testverfahren 7](#_Toc501092425)

[Module-Test (Unit-Test, Komponenten-Test) 7](#_Toc501092426)

[GUI-Testing 7](#_Toc501092427)

[Module-Integration-Test 8](#_Toc501092428)

[System-Integration-Test 9](#_Toc501092429)

[Test-Cases 10](#_Toc501092430)

[Inhalt 10](#_Toc501092431)

[Best Practice 11](#_Toc501092432)

[Quellen 12](#_Toc501092433)

[Ausgearbeitet von 12](#_Toc501092434)

[Burgstaller 12](#_Toc501092435)

[Haider 12](#_Toc501092436)

[Serschen 12](#_Toc501092437)

# Abstract

Software tests check the quality of a software and evaluate if it meets certain requirements. The findings are used to detect and correct bugs to make the finished application as error free as possible.

Tests are divided into four levels, which are unit testing, integration testing, system testing and acceptance testing. The unit testing checks a specific section of code, usually one or a few classes. Integration testing verifies the interfaces between the modules, while system testing evaluates the connection between different (computer-) systems. In acceptance-testing the customer can test the software.

Black and white box testing are two common used testing techniques. Black box testing checks the functionality of a program, but the outcome only shows if there are errors and not where they actually are. White box testing is used to test the internal structure of a program. It evaluates if the code fulfills the quality criteria.

GUI testing is the process of testing the graphical user interface. The test designer prepares test cases to test all possible GUI functionalities. There are three problems within this process: there are many possible operations, some can only be tested after a sequence of GUI operations; the GUI could change, but the level underneath not. To get rid of these problems, artificial intelligence is used sometimes. There are two methods to test the GUI: mouse position capture and event capture. While in mouse position capture the outcome is compared to bitmaps prepared by the test case designers, the events are saved in logs in event capture.

To test a certain part of a system under certain condition test cases are used to check if the part works correctly. The test cases are saved in an archive and can be found in the system to find a solution to errors. Best practice for test cases are that they should only test one possible problem; all negative and positive scenarios should be covered, they should be written in a clear and easy understandable language.

# Einleitung

Ein Softwaretest prüft und bewertet Software auf Erfüllung der für ihren Einsatz definierten Anforderungen und misst ihre Qualität. Die gewonnenen Erkenntnisse werden zur Erkennung und Behebung von Softwarefehlern genutzt. Tests während der Softwareentwicklung dienen dazu, die Software möglichst fehlerfrei in Betrieb zu nehmen.

# Test-Levels

Das Testen von einem System wird in drei Schichten unterteilt. Sie werden von unten nach oben Module-Schicht, Module-Integration-Schicht und System-Integration-Schicht genannt.

Die Module-Schicht kümmert sich um die kleinstmöglichen und sinnvollen Module einer Software. Dies kann zum Beispiel eine Methode, eine Klasse oder mehrere Klassen zusammen sein, falls diese Klassen keinen Sinn machen, solange die anderen Klassen nicht vorhanden sind.

Die Module-Integration-Schicht kümmert sich um das Zusammenspiel aller Module innerhalb eines einzelnen Rechners.

Und die System-Integration-Schicht kümmert sich um den Datenaustausch zwischen verschiedenen Rechnern, damit das System voll funktionstüchtig ist.

Die Akzeptanz-Schicht kümmert sich um alle Use-Cases und den Abnahmetest.

Bevor ein Test auf einer Schicht ausgeführt wird, sollte man überprüfen, dass alle Tests in den unteren Schichten ausgeführt wurden und keine Fehler gefunden worden sind.

# Black- und White-Box-Testing

Das Black- und White-Box-Testing sind Testprinzipe die bei jedem Testverfahren mehr oder weniger verwendet werden. Währenddessen Black-Box-Testing die Prinzipien eines Systems überprüft, analysiert das White-Box-Testing die Korrektheit der internen Struktur des Systems.

## Black-Box-Testing

Vom Blackbox-Testing spricht man, wenn man Testfälle alleine aus der Spezifikation des zu testenden Objektes (Produkt, Schüler, Buch, usw.) ableitet. Dieses Testverfahren kann nicht die Abwesenheit von Fehlern nachweisen. Sondern nur die Existenz von Fehlern beweisen.   
Anders gesagt per Blackbox-Testing kann man nicht sagen, wo der Fehler ist, aber dass ein Fehler existiert. Dadurch hat man eine hohe Wahrscheinlichkeit schnell Fehler aufzudecken, solange Fehler vorhanden sind.

Beim Blackbox-Testverfahren unterscheidet man:

* Äquivalenzklassenbasiertes Testen
* Grenzwertbasiertes Testen
* Fehlerbasiertes Testen
* Zustandsbasiertes Testen
* Testen mit Entscheidungstabellen

Beim **äquivalenzklassenbasierten Testen** beachtet man Wertebereiche, in denen sich das Produkt bzw. Software äquivalent verhalten muss. Nehmen wir zum Beispiel, einmal das Geburtsdatum eines Kunden. Beim Äquivalenzklassenbasiertes Testen überprüft man, ob das Geburtsdatum des Kunden stimmen kann.

* Geburtsdatum stimmt, wenn der Kunde noch lebt und im Intervall von heute bis heute vor 120 Jahren ist.
* Ungültiges Geburtsdatum, wenn das Geburtsdatum in der Zukunft liegt.
* Ungültiges Geburtsdatum, bei lebenden Kunden die älter als 120 Jahre sind (Wenn wir davon ausgehen das unsere Kunden nicht älter als 120 Jahre werden).

Das **zustandsbasierte Testen** verwendet man, wenn neben den Eingabewerten auch der bisherige Ablauf eines Systems Einfluss auf das Systemverhalten oder den Ausgabewert hat. Das Testobjekt kann ein komplettes System mit unterschiedlichen Zuständen, aber auch eine Klasse mit verschiedenen Zuständen in einem objektorientierten System sein.  
Das Testobjekt lässt sich in Form eines [Zustandsdiagramms](https://de.wikipedia.org/wiki/Zustandsdiagramm) modellieren. Von einem Startzustand ausgehend, kann das Testobjekt verschiedene Zustände einnehmen. Auslöser für Zustandsänderungen können Ereignisse oder erfüllte Bedingungen sein.

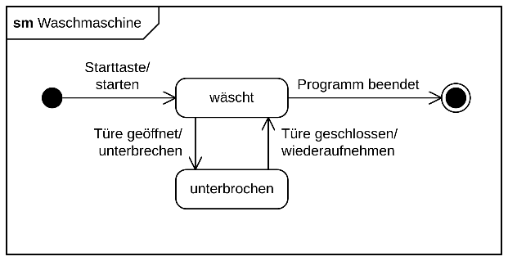


Abbildung 1 Zustandsdiagramm https://de.wikipedia.org/wiki/Zustandsdiagramm (UML)

## White-Box-Testing

Im Gegensatz zum Black-Box-Text ist für diesen Test also ein Blick in den Quellcode gestattet, das bedeutet es wird am Code Codeprüft.

Ein Beispiel für einen White-Box-Test ist ablaufbezogenes Testen, bei welchem der Ablaufgraph im Vordergrund steht. Das Qualitätskriterium ist hier das Testfälle in Bezug auf die Überdeckung der Quellcodes gewisse Hinlänglichkeitskriterien erfüllen.

Die Qualitätskriterien sind:

* Zeilenüberdeckung: Ausführung aller Quellcode-Zeilen
* Anweisungsüberdeckung bzw. Knotenüberdeckung: Ausführung aller Anweisungen
* Zweigüberdeckung bzw. Kantenüberdeckung: Durchlaufen aller möglichen Kanten von Verzweigungen des Kontrollflusses
* Bedingungsüberdeckung bzw. Termüberdeckung (mehrere Varianten): Durchlaufen aller möglichen ausschlaggebenden Belegungen bei logischen Ausdrücken in Bedingungen
* Pfadüberdeckung (mehrere Varianten): Betrachtung der Pfade durch ein Modul

Auch wenn das Programm nach genannten Kriterien überprüft wurde, bedeutet das noch nicht das es auch Fehlerfrei ist. Dies kann den Grund haben, dass zwar die Korrektheit des Softwaresystems selbst über den Code überprüft wird, aber nicht die Semantik selbst.

White-Box-Testing reicht alleine zum Testen eines Systems nicht aus, es sollte zusammen mit Black-Box-Testing kombiniert werden.

# Testverfahren

## Module-Test (Unit-Test, Komponenten-Test)

Beim Modultest (auch Komponententest oder in Englisch Unit-Test genannt) werden die kleinsten Software-Einheiten getestet, die sinnvoll isoliert zu testen sind. Dabei kann es sich nur von Eingabewerten abhängigen Funktionen oder Methoden, um eine einzelne Klasse, oder sogar um eine Reihe von sehr eng interagierenden Klassen handeln.

Die Modultests beschäftigen sich in der Regel hauptsächlich mit funktionalen Eigenschaften und wenden deswegen alle Möglichkeiten des Black-Box- bzw. White-Box-Tests an.

## GUI-Testing

In der Softwareentwicklung „grafical user interface testing“ ist der Prozess des Testens eines Produkts ob es die Grafischen Nutzer Interface Spezifikationen auch erfüllt.

Test Case Generation

Um ein Set von Test Cases zu generieren versuchen die Test Designer alle Funktionalitäten der GUI auszutesten.

Es gibt drei Probleme:

1.) Nicht wie ein CLI (Command Line Interface) System, hat eine GUI mehrere Operationen die getestet werden müsse. Als Beispiel WordPad hat 325 mögliche GUI Operationen.

2.) Manche Events können nur ausgelöst bzw. überprüft werden durch eine Sequenz von GUI Operationen. Wie z.B. das Speichern. Was die Schwierigkeit Exponentiell steigert.

3.) Regressions Testing kann auch ein Problem werden beim GUI Testen. Es ist möglich, dass die GUI sich signifikant ändert jedoch das darunter nicht.

Es werden heutzutage schon Künstliche Intelligenzen benutzt um GUIs intelligent zu testen um diese Probleme zu umgehen.

Es gibt 2 Methoden die GUIs zu testen, entweder:

1. Mouse position capture

Diese häufig gebrauchte Methode benutzt das Capture and Playback Verfahren. Man speichert zuerst alle Ausgänge mit Bitmap Grafiken und speichert das Testverfahren um es wiederholen zu können und anschließend die Grafiken zu vergleichen.

1. Event Capture

Benutzt die GUI interaction data um die Events abzufangen und in logs zu speichern welche gefiltert werden.

Um die Funktionalität von mehreren Systemen, die im Verbund in Betrieb gehen, zu garantieren ist ein systematischer System-Integrationstest unabdingbar. Bei diesem Test muss das Zusammenspiel der Systeme unter einander sowie ihre Kommunikation nach außen über die gemeinsame spezifizierte Schnittstelle erprobt und bestätigt werden. Für Unternehmen, die solche Verbundsysteme betreiben, ist der System-Integrationstest ein essentieller qualitätssichernder Faktor für die ordnungsgemäße Funktionalität ihrer IT und ein Garant für die Zuverlässigkeit, Sicherheit und Performanz ihrer Geschäftsprozesse.

## Module-Integration-Test

Beim Module-Integration-Test wird das Zusammenspiel zwischen den einzelnen Modulen (Units, Komponenten) überprüft. Dabei muss sowohl die korrekte Interaktion, wie z.B.: der Austausch von Daten durch Nachrichten oder gemeinsam genutzten Speicher oder die Nutzung von Funktionalität durch Aufrufe von Schnittstellenfunktionen über einzelne Teile hinaus, überprüft werden als auch das Nicht-Auftreten unerwünschter Effekte. Typischerweise erkannte Fehler betreffen die falsche Nutzung von Schnittstellen, unerlaubte Parameterwerte, aber auch Blockierungen durch gemeinsam genutzte Ressourcen oder ‚*Race-Conditions*‘ durch das ungeordnete Ändern gemeinsam genutzter Daten bis hin zu inkonsistenten Datenzuständen. Integrationstests werden wegen des vornehmlich System-internen Fokus beim Ersteller der Software durchgeführt.

Um den Vorgang zu erleichtern und Fehler bestmöglich lokalisieren zu können, sollte man davon ausgehen, dass dem Module-Integration-Test nach allen Module-Tests ausgeführt wird. In der Regel werden die einzelnen Module schon als fehlerfrei angenommen, was bei notwendigen Änderungen zur Fehlerbehebung den Rückschritt auf die Ebene des Module-Tests erforderlich macht. Es werden nicht nur die funktionellen Eigenschaften überprüft, sondern auch die nicht-funktionalen Eigenschaften.

Es gibt 2 Vorgehensweisen beim Module-Integration-Test:

* bottom-up
* top-down

Bei bottom-up wird schrittweise von der Kombination einzelner Module oder Klassen über logisch eng gekoppelte Teilsysteme bis hin zum Gesamtsystem (innerhalb eines Computers) getestet. Um die Aufrufe in den zu testenden Teilen möglichst gut zu überdecken, werden Testtreiber genutzt, die über Zeit mit dem wirklichen Zielsystem ausgetauscht werden.

Bei top-down beginnt man mit der größten Abstraktionsschicht in Kombination mit Teststubs, die dann schrittweise durch reale Teilsysteme ersetzt werden. Damit kann man die Ursache eines auftretenden Problems leichter identifizieren.

## System-Integration-Test

Der System-Integrationstest soll dazu dienen Inkonsistenzen zwischen den verbundenen Systemen aufzudecken, Inkonsistenzen die erst ans Tageslicht treten wenn die Systeme erst zusammenwirken. Da die Entwickler eines bestimmten Subsystems oft nicht wissen, was die Entwickler eines anderen Subsystems annehmen, sind Missverständnisse aufgrund falscher Annahmen an der Tagesordnung. Jeder wird behaupten seine Ergebnisse sind korrekt aber in der Vereinigung mit den Ergebnissen der Anderen sind sie falsch. Deshalb ist das Risiko eines Fehlschlages beim „Scharfschalten“ der Systeme besonders groß. Kein Unternehmen kann es sich leisten, seine Systeme mit ihren vielen gegenseitigen Abhängigkeiten zueinander ohne einen System-Integrationstest in Betrieb zu nehmen. Bei diesem Test komme es darauf an, mehrere unabhängig voneinander entwickelte Software-Systemen mit einer Vielzahl gemeinsamer Schnittstellen zu erproben. Dabei können auch unterschiedlicher Technologien ins Spiel kommen, z.B. verschiedener Webbrowser. Es gilt zu bestätigen, dass diese Technologien wirklich in allen Aspekten gegenseitig verträglich sind.

# Test-Cases

Ein Test-Case ist die Überprüfung von einem bestimmten Systemteil unter bestimmten Bedingungen, um überprüfen zu können, ob das Systemteil korrekt funktioniert.

Test-Cases werden archiviert. Durch ihnen können Probleme im System gefunden werden und die benötigten Veränderungen zum Lösen des Problems herausgeleitet werden.

## Inhalt

Ein Test-Case kann diese Daten beinhalten:

* Test Suite ID
  + Die ID des Test Suites wohin dieser Test-Case gehört
* Test Case ID
  + Die ID dieses Test Cases
* Test Case Summary
  + Kurze Zusammenfassung worüber der Test-Case ist
* Related Requirement
  + Die IDs zu den Voraussetzungen die dieser Test-Case benötigt
* Prerequisites
  + Alle Grundvoraussetzungen die vorhanden sein müssen, damit der Test ausgeführt werden kann
* Test Procedure
  + Step-by-Step alle Schritte des Tests aufgelistet
* Test Data
  + Die Daten die verwendet werden um den Test auszuführen (oder links zu ihnen)
* Expected Result
  + Das zu erwartende Ergebnis von dem Test
* Actual Result
  + Das Ergebnis nach dem Ausführen des Testes
* Status
  + Der Status nach dem Ausführen des Test-Cases
  + Statusmöglichkeiten (Pass, Fail, Not Executed, Blocked)
* Remarks
  + Irgendwelche Kommentare zum Test-Case oder dessen Ausführung
* Created by
  + Name des Erstellers des Test-Cases
* Date of Creation
  + Datum der Erstellung des Test-Cases
* Executed by
  + Name des Ausführer des Test-Cases
* Date of Execution
  + Datum der Ausführung des Test-Cases
* Test Environment
  + Die Umgebung (Hardware/Software/Network) in welchem der Test-Case ausgeführt wurde

## Best Practice

* So gut wie es geht, sollen Test-Cases nur eine Sache überprüfen. Man sollte sie nicht mit anderen Test-Cases überlappen oder umsonst komplizierter machen.
* Probiere sicher zu gehen, dass alle positiven, als auch negativen Szenarien gedeckt sind.
* Sprache:
  + Schreib in einfacher und leicht zu verstehender Sprache
  + Verwende aktive Sprache: Tu dies, Tu das.
  + Verwende für Felder und Forms die exakten Namen und umschreib diese nicht.
* Eigenschaften eines guten Test-Cases:
  + Akkurat: genau zum Zweck
  + Sparsam: Keine unnötigen Wörter.
  + Zurückverfolgbarkeit: Kann zu den Anforderungen zurückverfolgt werden.
  + Wiederholbarkeit: Der Test-Case kann immer wieder und wieder ausgeführt werden
  + Wiederverwendbarkeit: Kann falls nötig wiederverwendet werden

# Quellen

<http://www.enzyklopaedie-der-wirtschaftsinformatik.de/lexikon/is-management/Systementwicklung/Hauptaktivitaten-der-Systementwicklung/Software-Implementierung/Testen-von-Software/Modultest>

<https://www.johner-institut.de/blog/iec-62304-medizinische-software/blackbox-testing/>

<https://de.wikipedia.org/wiki/Zustandsbezogener_Test>

<https://de.wikipedia.org/wiki/Zustandsdiagramm_(UML)>

<http://softwaretestingfundamentals.com/white-box-testing/>

<http://www.gm.fh-koeln.de/~winter/tav/html/tav30/TAV30P07_ANECON.pdf>

<http://www.enzyklopaedie-der-wirtschaftsinformatik.de/lexikon/is-management/Systementwicklung/Hauptaktivitaten-der-Systementwicklung/Software-Implementierung/Testen-von-Software/Integrationstest>

http://softwaretestingfundamentals.com/test-case/

# Ausgearbeitet von

## Burgstaller

* Einleitung
* GUI-Test
* System-Integration-Test

## Haider

* Test-Levels
* Black-Box-Testing
* Module-Test
* Module-Integration-Test
* Test-Cases (Inhalt, Best Practice)

## Serschen

* Englisch Abstract
* White-Box-Testing