TESTING

Methoden der Software-Prüfung

Definition	Testen ist ein wesentlicher Teil im Qualitätsmanagement von Projekten der Softwareentwicklung
Ziel	Messen der Qualität des Softwaresystems Um den wirtschaftlichen Aspekten klar zu werden.

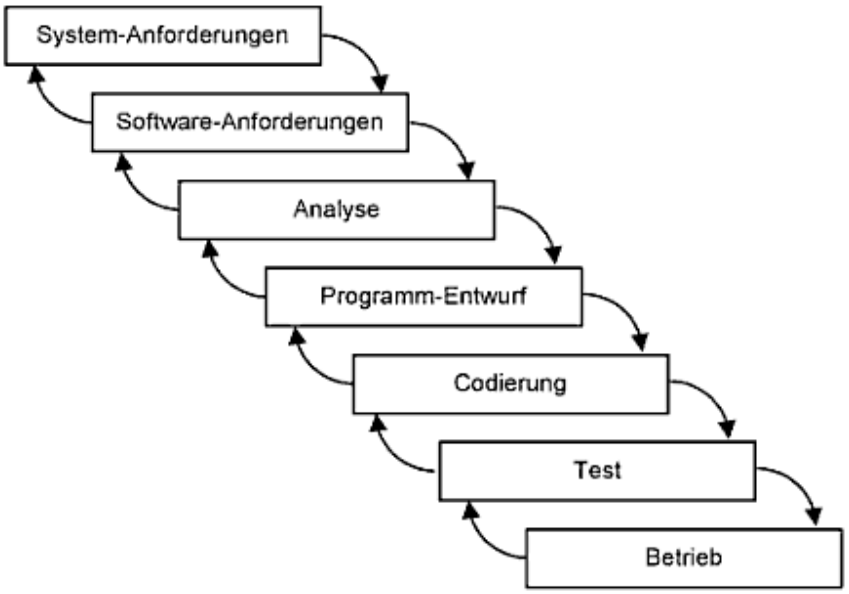
Begriffe

Softwaretest	prüft und bewertet Software, einzelne Testmassnahme
Test, Testen	Gesamtheit der Massnahmen zur Überprüfung der Softwarequalität

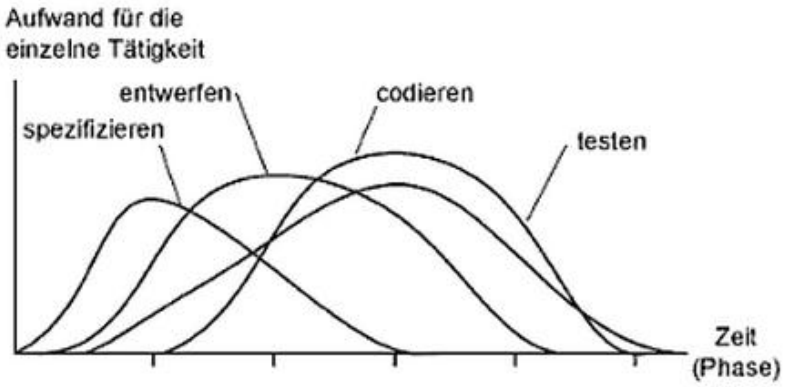
Eigenschaften

- immaterieller Charakter (also Entwicklung, nicht Fertigung)
- fehlende Stetigkeit (Sprunghafte Änderungen, nicht Verschleisseffekte)
- Geistige Nähe hindert uns, unsere Schwächen zu erkennen
- Nachweis, dass keine Fehler vorhanden sind, ist unmöglich

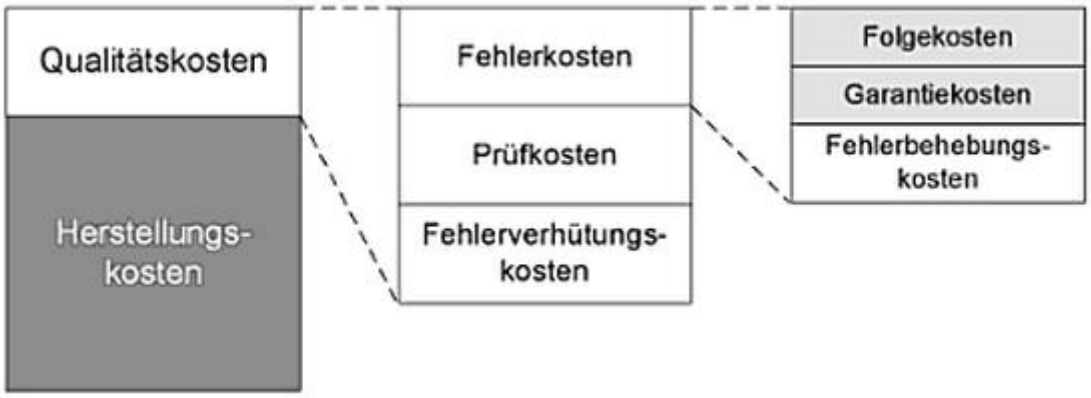
Software-Lebenslauf



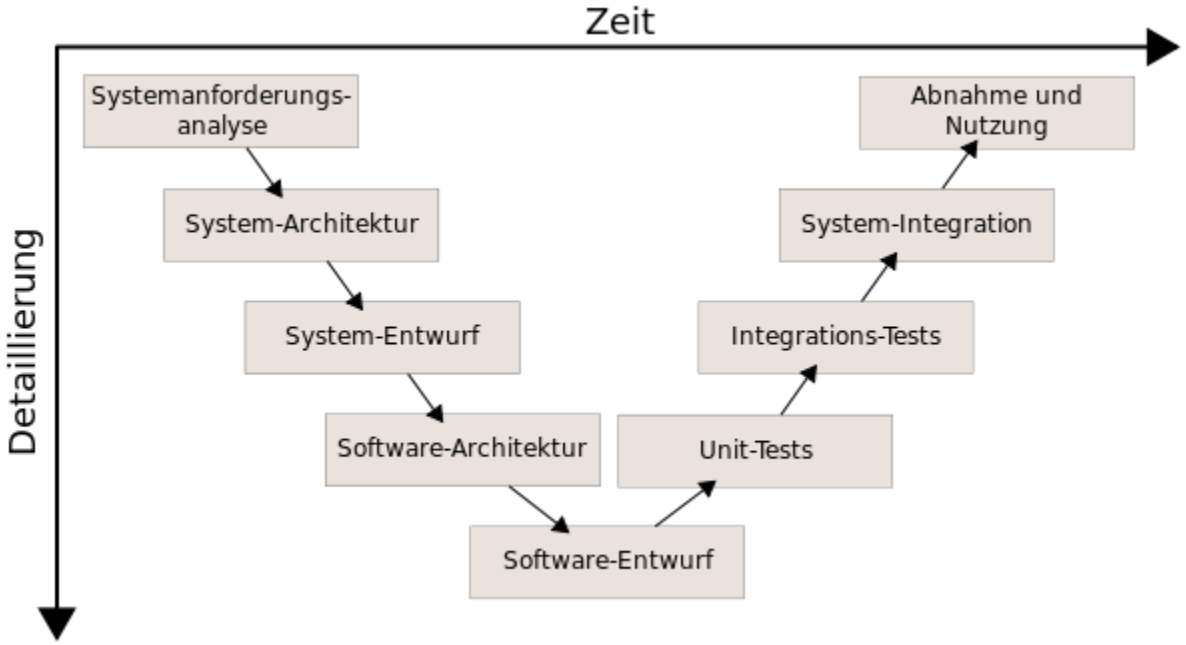
Kostenmodell



		M1	M2	M3	M4	M5
spezifizieren	Anforderungsdokument	V1	V2		V3	
entwerfen	Architekturbeschreibung		V1	V2	V3	
entwerfen	Entwurfsbeschreibung			V1	V2	
codieren	Quellcode				V1	V2
testen	Testvorschrift, -bericht			V1	V2	V3
schreiben	Benutzerhandbuch		V1			V2
Tätigkeit	Arbeitsergebnis	Versionen (Vi) der Arbeitsergebnisse an den Meilensteinen (Mj)				

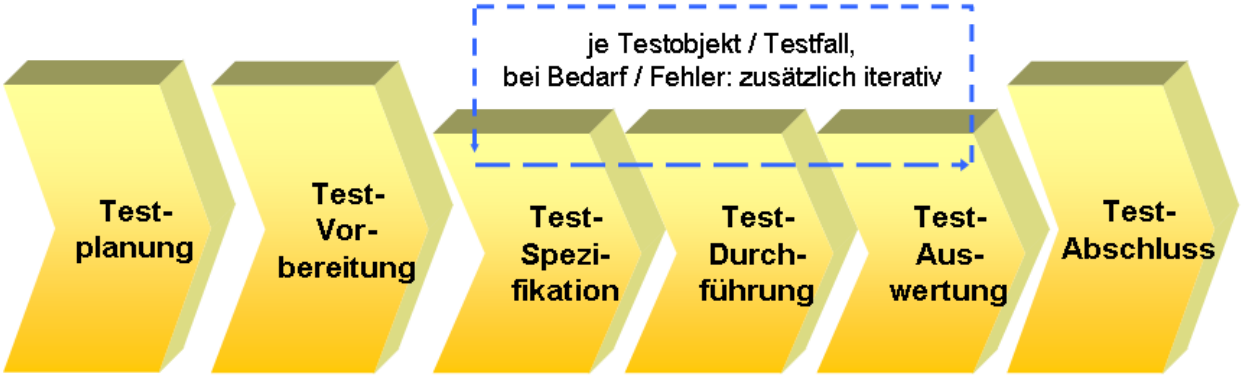


V-Modell / Badewannekurve



Komponententest / Modultest	Funktionalität innerhalb der einzelnen Module
Integrationstest / Interaktionstest	Zusammenarbeit von abhängigen Komponenten (Schnittstellen)
Systemtest	Gesamtes System auf Anforderungen testen
Abnahmetest / Verfahrenstest (UAT)	Test durch Auftraggeber

Phasenmodell



Quelle: Pol, Koomen, Spillner

Das Modell ist 'generisch', d.h.:

Es wird in mehreren Ebenen, mit unterschiedlichen Schwerpunkten angewendet.

a) Projekt-Ebene, Masterplanung:

b) Je Teststufe:

c) Je Testobjekt / Testfall:

Statischer Test
(ohne Programmausführung)

Review (Strukturierte Gruppenprüfung)

Technisches Review
Informelles Review
Walkthroug
Inspection

Statische Code-Analyse

Überprüfung gegen Richtlinien,
Datenflussanalyse, Kontrollflussanalyse

Dynamischer Test
(während Programmausführung)
festgelegte Eingabe- und Ausgabedaten
Fehlerfindung durch vergleichen

Strukturorientierter Test

Kontrollflussorientierte Tests
Datenflussorientierte Tests

Funktionsorientierter Test

Äquivalenzklassenbildung
Grenzwertanalyse
Pairwise-Methode
Zustandsbasierte Testmethoden
Ursache-Wirkungs-Analyse

Diversifiziernder Test

Back-to-Back-Test
Mutationen-Test
Regressionstest

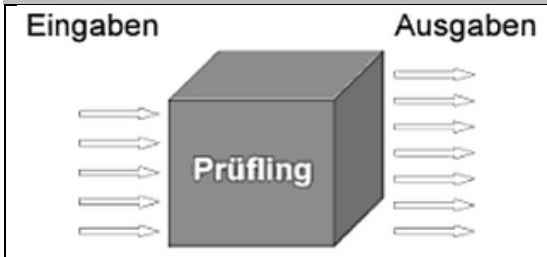
Alle Entwicklungsergebnisse kann man mit Reviews prüfen	Test ist reproduzierbar Aufwand kann mehrfach genutzt werden Zielumgebung wird mitgeprüft Systemverhalten wird sichtbar gemacht
menschliches Versagen (übersehen von Fehlern)	Aussagekraft wird überschätzt Prüfung nicht aller Programmeigenschaften Nicht alle Anwendungssituationen werden nachgebildet zeigt keine Fehlerursache

Begriffe	
Software	Rechner-Programme und Dokumentationen
Fehler	Diskrepanz zum theoretisch korrektem Wert
Prüfling	Software-Produkt, dass dem Test unterzogen wird
Autor	Urheber des Prüflings
Manager	Person, in deren Verantwortungsbereich der Prüfling erstellt wird
Kollege	Person, die unbeteiligt an der Erstellung ist, aber gewisse Aspekte zu beurteilen vermag

Teststrategien	
top-down	Haupt- vor Detailfunktionen testen; untergeordnete Routinen werden beim Test zunächst ignoriert
bottom-up	Detailfunktionen zuerst testen; übergeordnete Funktionen oder Aufrufe werden mittels "Testdriver" simuliert
hardest first	Situationsbedingt das Wichtigste zuerst
big-bang	Alles auf einmal

Dokumentation	
Testplan	Beschreibt Umfang, Vorgehensweise, Terminplan, Testgegenstände.
Testdesignspezifikation	Beschreibt die im Testplan genannten Vorgehensweisen im Detail.
Testfallspezifikationen	Beschreibt die Umgebungsbedingungen, Eingaben und Ausgaben eines jeden Testfalls.
Testablaufspezifikationen	Beschreibt in Einzelschritten, wie jeder Testfall durchzuführen ist.
Testobjektübertragungsbericht	Protokolliert, wann welche Testgegenstände an welche Tester übergeben wurden.
Testprotokoll	Listet chronologisch alle relevanten Vorgänge bei der Testdurchführung.
Testvorfallbericht	Listet alle Ereignisse, die eine weitere Untersuchung erforderlich machen.
Testergebnisbericht	Beschreibt und bewertet die Ergebnisse aller Tests.

Blackboxtest



Funktionsüberdeckung	Jede Funktion wird mindestens einem Testfall ausgeführt.
Eingabeüberdeckung	Jedes Eingabedatum wird in mindestens einem Testfall verwendet (jede Art)
Ausgabeüberdeckung	Jedes Ausgabedatum wrid in mindestens einem Testfall erzeugt (jede Art)
Test ohne Kenntnisse über die innere Funktionsweise Ursache bleibt unbekannt, 2 Fehler kännen sich aufheben!!	

bessere Verifikation des Gesamtsystems
Testen von semantischen Eigenschaften bei geeigneter Spezifikation
Protabilität von systematisch erstellten Testsequenzen auf plattformunabhänigige Implementierungen
grösserer organisatorischer Aufwand
zusätzlich eingefügte Funktionen bei der Implementierung werder nur durch Zufall getestet
testsequenzen einer unzureichendern Spezifikation sind unbrauchbar

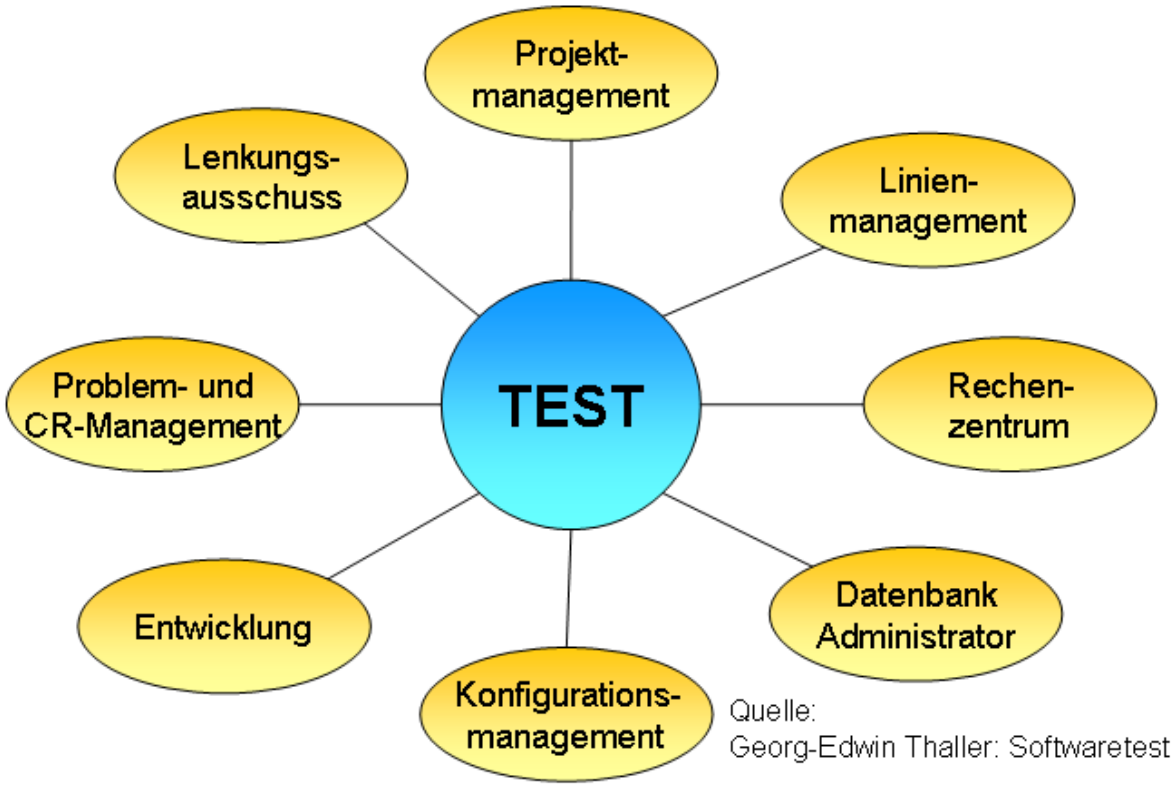
Auswahl der Testfälle: Grenzwerte, spezielle Werte, Äquivalenzklassenmethode, Klassifikationsbaum-Methode, Entscheidungstabellen, zustandsbezogene Tests, use case Tests, Ursache- und Wirkungsgrad, Auffinden von Robustheits- und Sicherheitsproblemen Fuzzing, Risikoanalyse

Glass-Box-Test (White-Box-Test)

Zeilenüberdeckung	Ausführung aller Quellcode-Zeilen
Anweisungsüberdeckung	Ausführung aller Anweisungen
Zweigüberdeckung / Kantenüberdeckung	Durchlaufen aller möglichen Kanten von Verzweigungen des Kontrollflusses
Bedingungsüberdeckung / Termüberdeckung	Durchlaufen aller möglichen ausschlaggebenden belegungen bei logischen Ausdrücken in Bedingunen
Pfadüberdeckung	Betrachtung der Pfade durch ein Modul

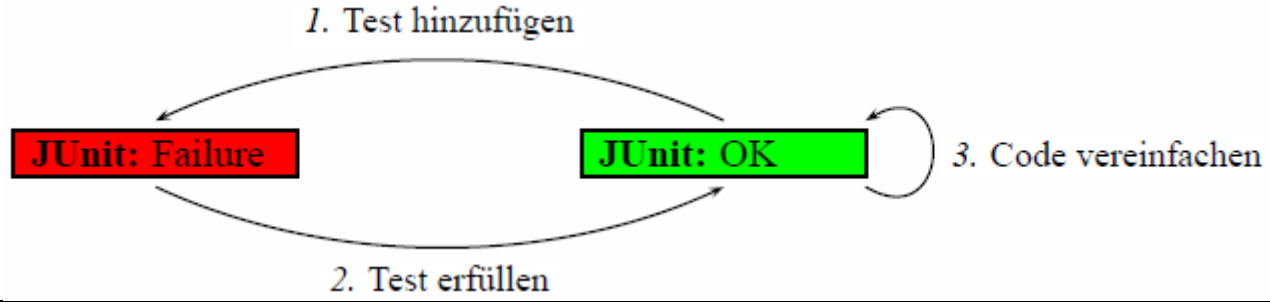
Test mit Kenntnissen über die innere Funktionsweise

Schnittstellen



Quelle:
Georg-Edwin Thaller: Softwaretest

Testen mit JUnit 4

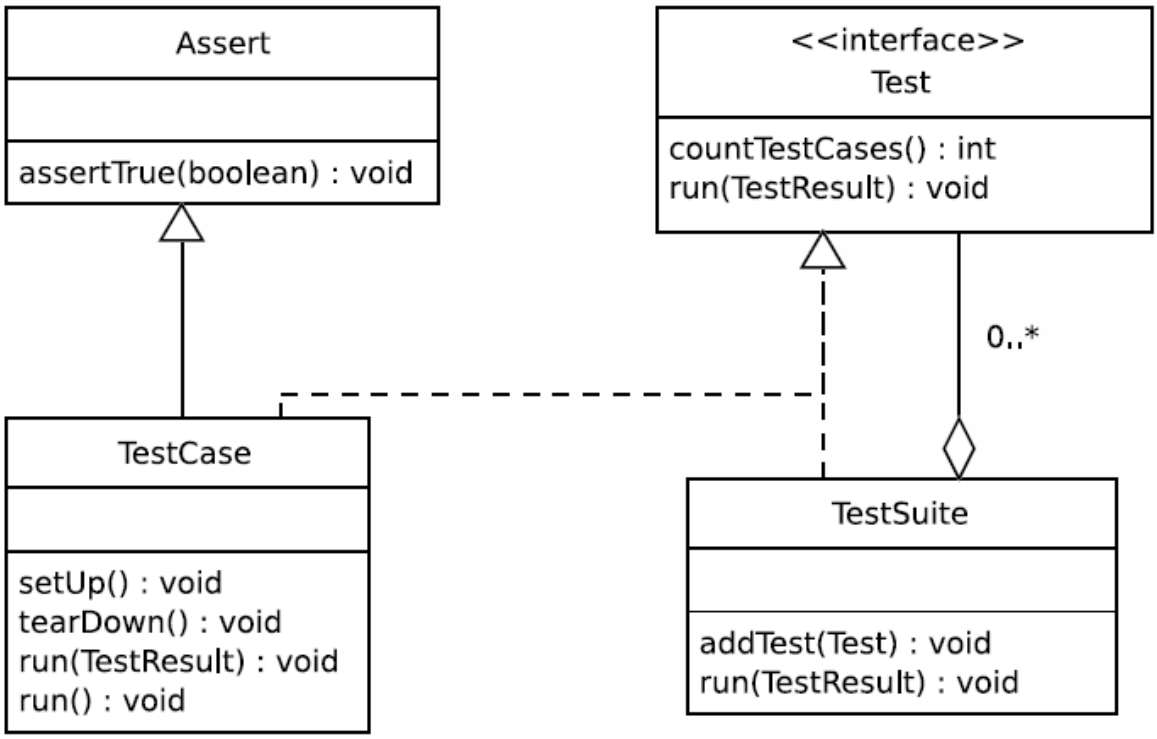
Prinzip	Testen ist eine Absicherung (Beispiel Klettern). Viele Kleine Tests ergeben weniger Fehler am Ende als ein grosser Test am Schluss. Reihenfolge der Testfälle spielt keine Rolle (Ergebnisse werden protokolliert)
Muster	Erstellen eines Tests; Test fehlschlagen lassen; genau soviel Code hinzufügen, wie zum Erfüllen des Tests notwendig ist 

Assert	
`assertTrue(boolean cond)`	überprüft ob true
`assertFalse(boolean cond)`	überprüft ob false
`assertNotNull(Object object)`	überprüft ob nicht null
`assertEquals(Object expected, Object actual)`	überprüft ob zwei gleiche Objekte (equals)
`assertEquals(double exp, double act, double delta)`	überprüft ob zwei gleiche Objekte mit Toleranz delta (auch für float)
`assertSame(Object expected, Object actual)`	überprüft ob zwei Variablen auf dasselbe Objekt zeigen
`assertNotNull("Must not be null", stack);`	Eigener Fehlertext
TestCase (Textmethoden)	
`public void test...() { ... }`	Grundstruktur
1. Eine Unterklasse von TestCase definieren 2. Instanzvariablen deklarieren, die den Zustand des Fixture speichern 3. Diese Variablen durch Überschreiben der `setUp()`-Methode initialisieren 4. Nach dem Test durch Überschreiben von `tearDown()` den Speicher leeren, Datenbankverbindungen beenden usw.	Ablauf eines Testfalls
`public void testException() { try { //Code, der zur Ausnahme fuehrt fail("Ausnahme doch nicht aufgetreten"); } catch (Exception expected) { //In Ordnung, Ausnahme aufgetreten } }`	Erwartete Ausnahmen
mittels **throws** weiterreichen	Unerwartete Ausnahmen
TestSuite	

`import junit.framework.*;`
`public class AllTests {`
`  public static Test suite() {`
`  TestSuite suite = new TestSuite();`
`  suite.addTestSuite(StackTest.class);`
`  //Weitere Tests hinzufuegen`
`  return suite;`
`  }`
`}`
	Klasse AllTests

Assert stellt statische Methoden zur Verfügung, mit denen tatsächliche Werte mit erwarteten verglichen werden können
TestCase ist die Klasse, von der alle Tests zwingend erben.
Mittels einer **TestSuite** ist es möglich, mehrere Tests zu einer Sammlung von Tests zusammenzufassen

UML-Diagramm



Änderungen durch JUnit4

- Testklassen müssen nun nicht mehr von TestCase erben
- Die Test-Methoden werden mit der Annotation @Test markiert
- Die Namenskonvention test... entfällt
- Die bisher als setUp() und tearDown() bekannten Methoden werden nun durch die Annotationen @Before bzw. @After realisiert.
- Bei @Before-Initialisierungen werden also zunächst die der Superklasse(n) ausgeführt, dann die der erbenden Klasse, bei den @After-Methoden verhält es sich genau umgekehrt.
- @BeforeClass und @AfterClass um Code nur einmal pro Testklasse auszuführen
- mittels @Ignore kann die Ausführung einzelner Testmethoden zeitweise unterbunden werden.
- für zeitkritische Aufgaben gibt es die Angabe eines Zeitlimits in der Form @Test(timeout = n).

Code Coverage mit EclEmma (Eclipse Plug-in)

Messen, durch welche Klassen, Methoden, Bläcke und Zeilen Code die Abarbeitung lief.
Das Ergebnis lässt auf die Güte dieser Modultests schließen und somit auf die technische Qualität der Software
Erlaubt auch die Messung der Testabdeckung ohne vorhergehende Instrumentierung des Codes.
Erkennung von totem Code oder nicht benötigte Programmteile

Logging mit Logback

Quelle der Fehlerquelle herausfinden
Meldungen flexibel ein und ausschalten

Module

- logback-core lays the groundwork for the other two modules
- logback-classic you can readily switch back and forth between logback and other logging frameworks
- logback-access integrates with Servlet containers, such as Tomcat and Jetty, to provide HTTP-access log functionality