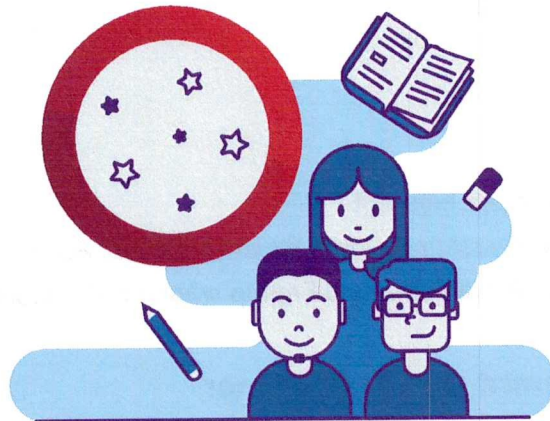




Seminare und Weiterbildung

ISTQB® Certified Tester Foundation Level

Ausbildung nach ISTQB® Lehrplan 2018 v3.1
Version 1.0t

0. Allgemeines / Einführung »
1. Organisatorisches



Referenten:

- **Arjan Brands**
arjan.brands@trendig.com
- **Jenny Doering**
jenny.doering@trendig.com
- **Werner Lieblang**
werner.lieblang@trendig.com
- **Oliver Rupnow**
oliver.rupnow@trendig.com
- ...



trendig technology services GmbH
Kleiststrasse 35
10787 Berlin

training@trendig.com
Telefon: 49 30 747628-0



Dieses Seminar bildet die Grundlage für die Prüfung zum „Certified Tester Foundation Level“ nach ISTQB®*

• Hauptziele der Ausbildung zum Certified Tester:

- Kenntnisse im Softwaretesten stehen auf einer fundierten Grundlage
- Softwaretests können projektspezifisch konzipiert werden
- Prüftechniken und Prüfziele können adäquat ausgewählt werden
- Werkzeuge zur Testunterstützung können angemessen bewertet und genutzt werden

• Das Begleitmaterial besteht aus:

- Einem kompletten Foliensatz
- Einem Satz Übungsunterlagen mit Lösungen
- Literaturliste

Benutzte Icons



Definition aus dem Glossar



Nicht im Syllabus oder sonstige Ergänzungen



Übung

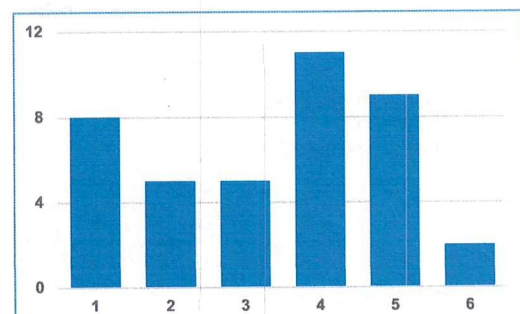
* ISTQB = International Software Testing Qualifications Board

Der Foliensatz wurde nach dem ISTQB®-Lehrplan für den „Certified Tester Foundation Level“, Version 2018, erstellt

- Im Anschluss an die Veranstaltung kann die Prüfung zum **Certified Tester Foundation Level** abgelegt werden
- Die **Abnahme** erfolgt durch Vertreter eines unabhängigen **Prüfungsinstituts**
- Die Prüfung besteht aus einem **60-minütigen Multiple-Choice-Test**
- Es sind **40 Fragen** zu beantworten. **26 Fragen (65%)** müssen korrekt beantwortet werden, um zu bestehen.

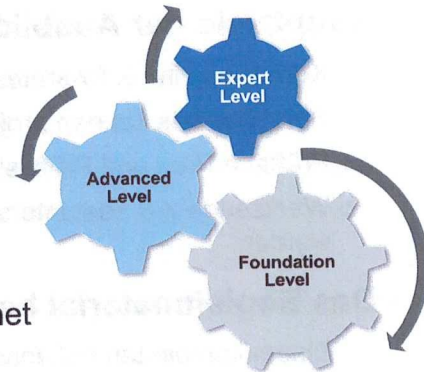
Lernziele / kognitive Stufen des Wissens:

- **K1:** (remember) sich erinnern
- **K2:** (understand) verstehen
- **K3:** (apply) anwenden
- **K4:** (analyze) analysieren



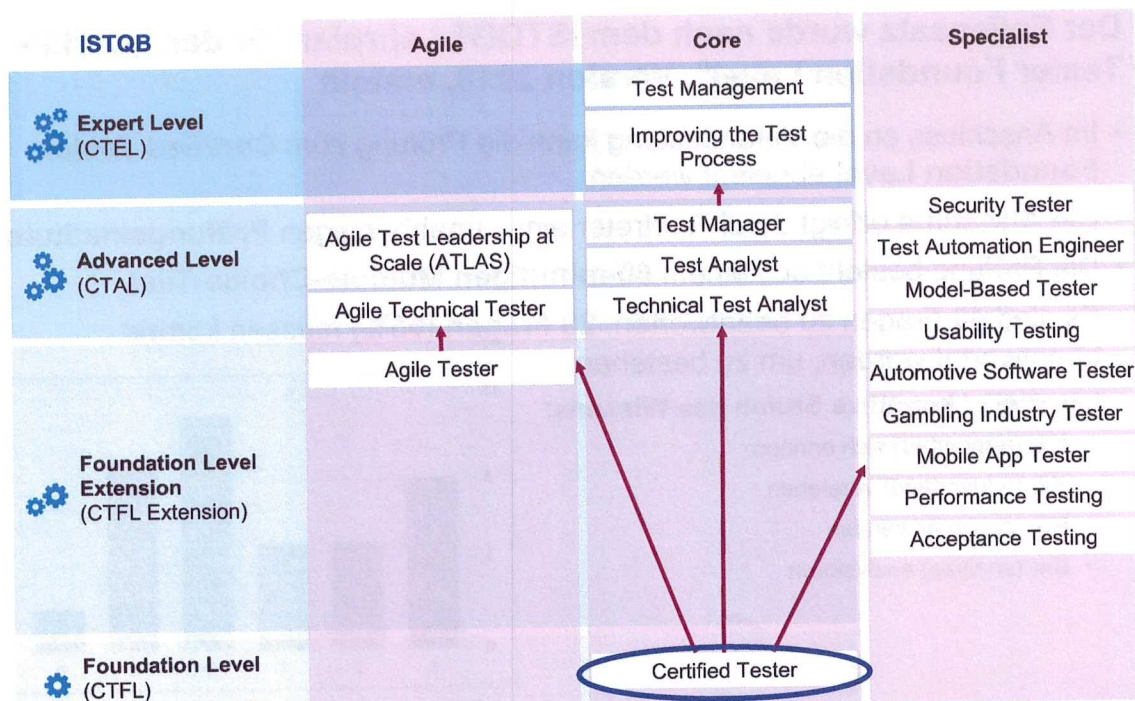
Qualifizierungsprogramm des ISTQB®

- 1998 wurde in Großbritannien ein mehrstufiges Qualifizierungsprogramm definiert
- Die Grundlagen zum Softwaretest sind im Lehrplan für den *Foundation Level* beschrieben, Version 2018 v3.1
- Seit 2004 werden Zertifikate für den *Advanced Level* (*Test Manager*, *Test Analyst*, *Technical Test Analyst*) angeboten
- Informationen zum Expert Level stehen im Internet zur Verfügung



Zielgruppe

- Angesprochen sind Software-Tester, Entwickler und Projektleiter in Software- und Industrieunternehmen, die ihre Kenntnisse auf eine fundierte Grundlage stellen wollen



German Testing Board (GTB)

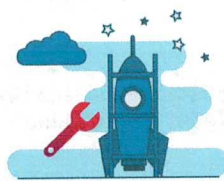
- Es sind landesspezifische *Testing Boards* etabliert worden, die gemeinsam das *International Software Testing Qualifications Board* bilden
- In Deutschland: das German Testing Board GTB
- In Österreich: das Austrian Testing Board (ATB)
- Begriffsdefinitionen, die dem Glossar (informativ) des GTB entsprechen, sind in **Fett-Kursiv-Schrift** dargestellt
<http://www.german-testing-board.info/lehrplaene/istqbr-certified-tester-schema/glossar/>
- Zu anderen Definitionen ist entweder die Quelle angeführt (z. B. ISO 29119) oder die Erklärungen stellen informelle Hintergrundinformationen dar

Auf diesen vier Feldern sind wir am stärksten



innovation

Design Thinking
Design Sprints
Gamification
User Centered Design



engineering

Requirements Engineering
Software Development
Testing Services
Test Automation
Web Security



training

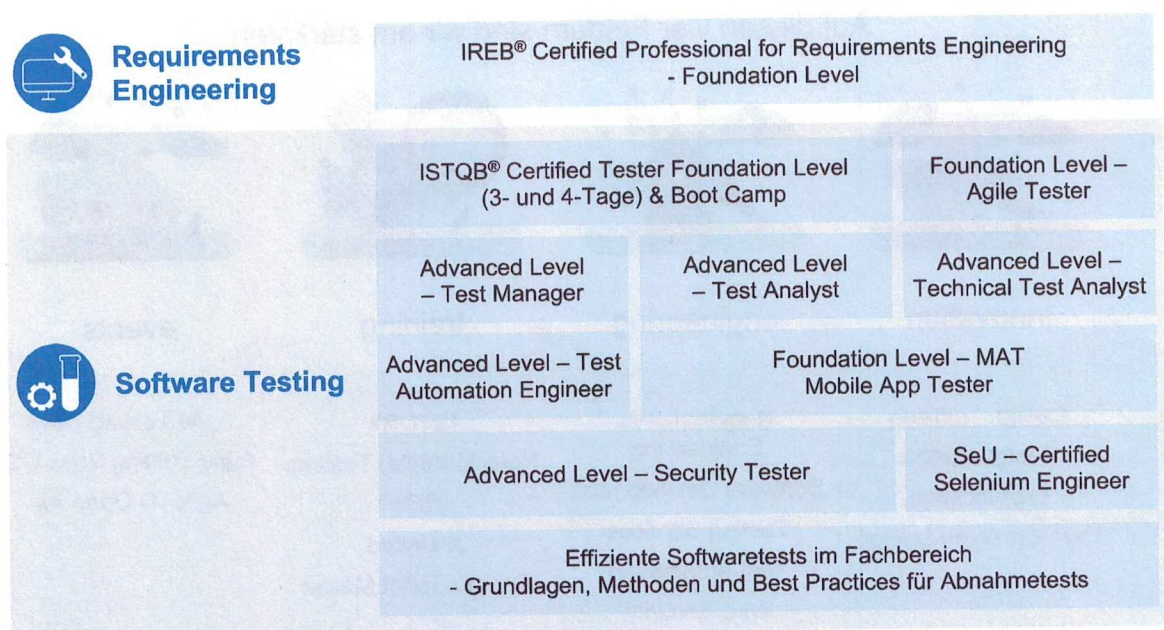
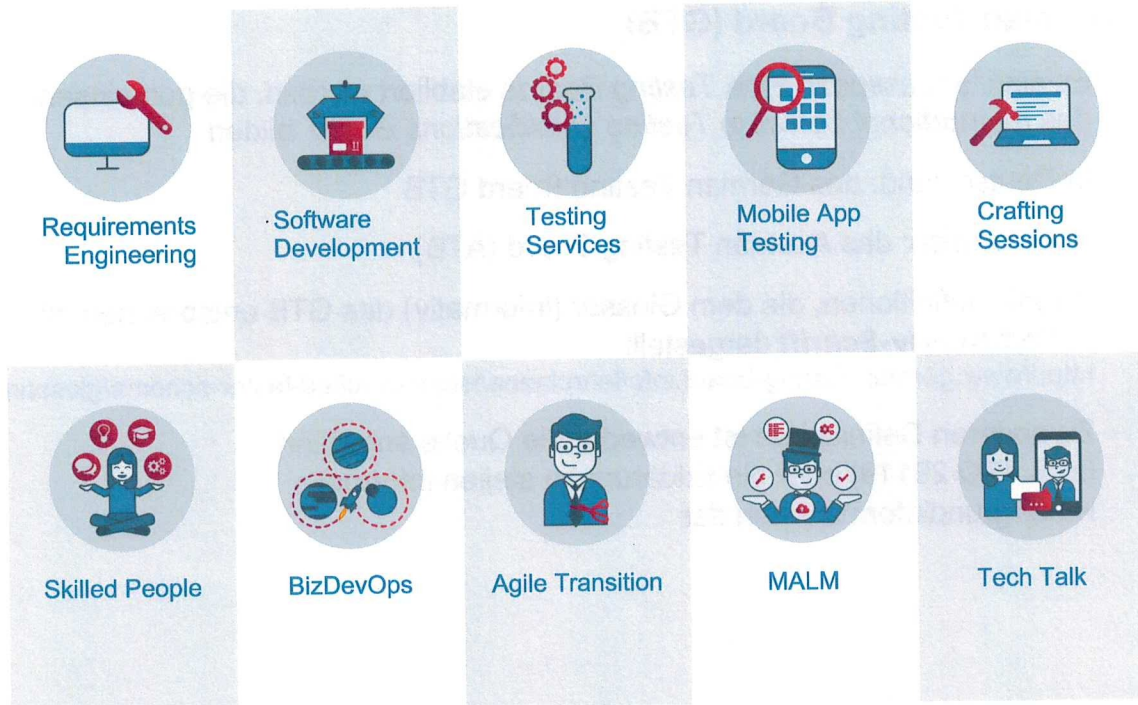
ISTQB®
Holistic (Agile) Testing
IREB®
X-United
Design Sprint Master

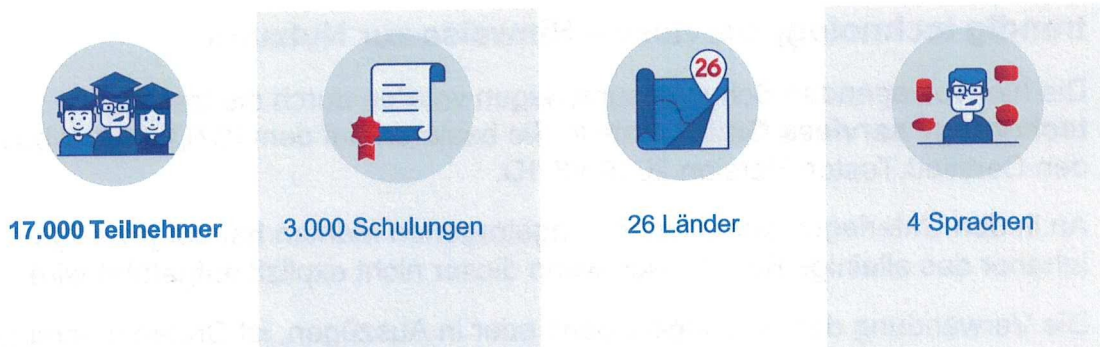
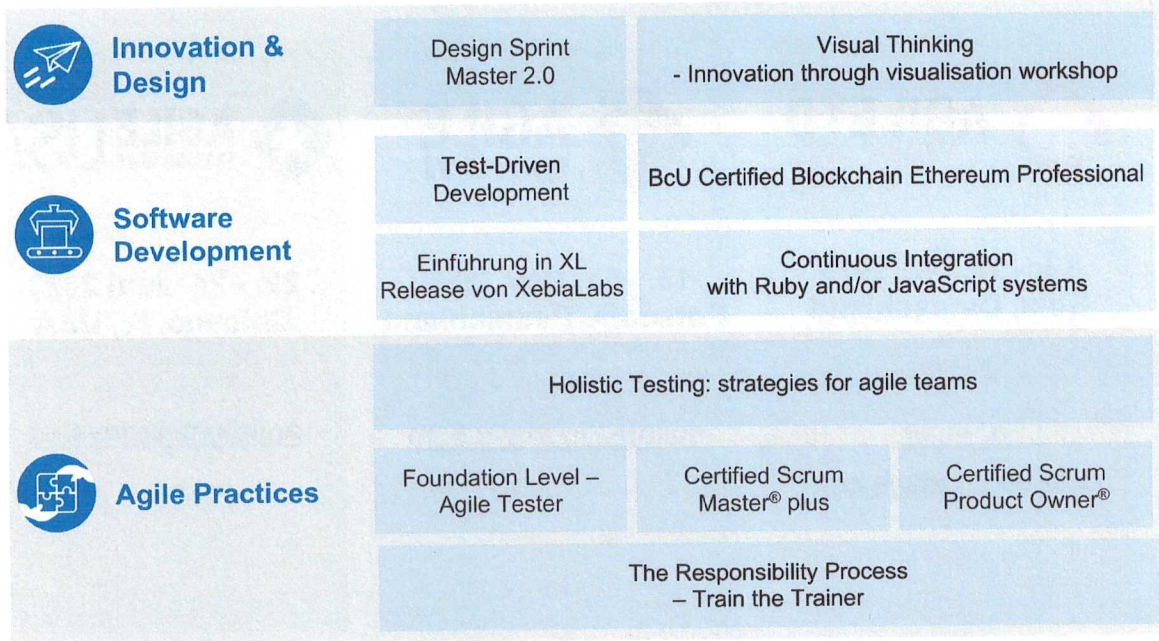


events

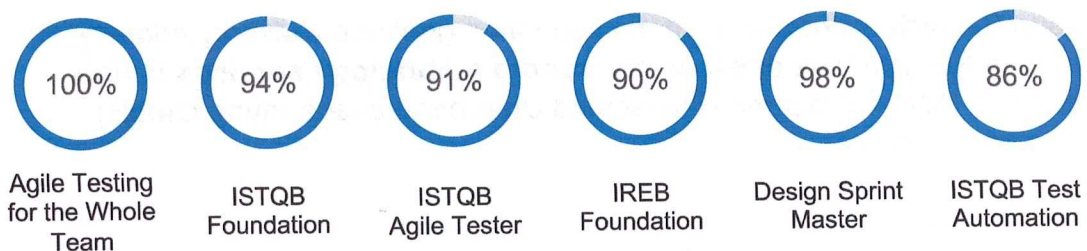
Agile Testing Days
Agile Testing Days US
AgileTD Open Air

Am liebsten begleiten wir Dich schon beim Start Deiner Projektreise und sind noch nach der Ankunft an Deiner Seite.





Unsere weltweiten Bestehensquoten in Prozent %



 AGILETD <i>Open Air</i> 13. - 15. Mai 2023 Köln, Deutschland openair. agiletestingdays.com	 AGILE TESTING DAYS 13. - 16. Nov. 2023 Potsdam, Deutschland agiletestingdays.com	 AGILE USA TESTING DAYS 22. - 24. Juni 2023 Chicago, IL, USA agiletestingdays.us
---	---	---

trendig technology services – Hinweise zur Nutzung

Die hier vorliegenden Schulungsunterlagen wurden durch die **trendig technology services GmbH** erstellt. Sie basieren auf dem ISTQB® Syllabus für den Certified Tester, Version 2018 V3.1D.

An in den Unterlagen enthaltenen, eingetragenen Marken hat der jeweilige Inhaber das alleinige Recht, auch wenn dieser nicht explizit aufgeführt wird.

Die Verwendung der Unterlagen, ganz oder in Auszügen, ist Dritten untersagt. Kunden der **trendig technology services GmbH** dürfen die Unterlagen intern, für nicht kommerzielle Zwecke, nutzen.

Wenn Sie nicht sicher sind, ob die von Ihnen geplante Nutzung zulässig ist, wenden Sie sich bitte direkt an die **trendig technology services GmbH** (formerly D&H – Díaz und Hilterscheid Unternehmensberatung GmbH)
www.trendig.com

Berlin, 11.01.23

I. Grundlagen des Softwaretestens » Agenda

Kapitel I – Grundlagen des Testens

- I/1 Was ist Testen?
- I/2 Warum ist Testen notwendig?
- I/3 Sieben Grundsätze des Softwaretestens
- I/4 Testprozess
- I/5 Die Psychologie des Testens

Lernziele für die Grundlagen des Softwaretestens

1 von 2

1.1 Was ist Testen?

- FL-1.1.1 (K1) Typische Ziele des Testens identifizieren können
- FL-1.1.2 (K2) Testen von Debugging unterscheiden können

1.2 Warum ist Testen notwendig?

- FL-1.2.1 (K2) Beispiele dafür geben können, warum Testen notwendig ist
- FL-1.2.2 (K2) Die Beziehung zwischen Testen und Qualitätssicherung beschreiben können und Beispiele dafür geben können, wie Testen zu höherer Qualität beiträgt
- FL-1.2.3 (K2) Zwischen Fehlhandlung, Fehlerzustand und Fehlerwirkung unterscheiden können
- FL-1.2.4 (K2) Zwischen der Grundursache eines Fehlerzustands und seinen Auswirkungen unterscheiden können

1.3 Sieben Grundsätze des Testens

- FL-1.3.1 (K2) Die sieben Grundsätze des Softwaretestens erklären können

Lernziele für die Grundlagen des Softwaretestens

2 von 2

1.4 Testprozess

- FL-1.4.1 (K2) Die Auswirkungen des Kontexts auf den Testprozess erklären können
- FL-1.4.2 (K2) Die Testaktivitäten und zugehörigen Aufgaben innerhalb des Testprozesses beschreiben können
- FL-1.4.3 (K2) Arbeitsergebnisse unterscheiden können, die den Testprozess unterstützen
- FL-1.4.4 (K2) Die Bedeutung der Pflege der Verfolgbarkeit zwischen Testbasis und Testarbeitsergebnissen erklären können

1.5 Die Psychologie des Testens

- FL-1.5.1 (K1) Die psychologischen Faktoren identifizieren können, die den Erfolg des Testens beeinflussen
- FL-1.5.2 (K2) Den Unterschied zwischen der für Testaktivitäten erforderlichen Denkweise und der für Entwicklungsaktivitäten erforderlichen Denkweise erklären können

Kapitel I – Grundlagen des Testens

- I/1 Was ist Testen?
- I/2 Warum ist Testen notwendig?
- I/3 Sieben Grundsätze des Softwaretestens
- I/4 Testprozess
- I/5 Die Psychologie des Testens

Testen ist ein komplexer Prozess

- Testen ist mehr als nur Testdurchführung, sprich Software ausführen und Ergebnisse prüfen
- Der Testprozess (Kapitel 1.4) besteht aus unterschiedlichen Phasen
 1. **Testplanung, Testüberwachung und -steuerung** (durch Testmanager)
 2. **Testanalyse**
 3. **Testentwurf**
 4. **Testrealisierung**
 5. **Testdurchführung**
 6. **Testabschluss** (durch Testmanager)

(durch Testanalyst, Tester u.ä)
- Die Testaktivitäten umfassen
 - Das Prüfen von Arbeitsergebnissen (statische Tests, Verifizierung, Spezifikation)
„Wird das Produkt richtig erstellt?“
 - Das Ausführen von Software (dynamische Tests, Validierung, einsatzzweckgeeignet)
„Wird das richtige Produkt entwickelt?“



Begriffe

1 von 2

- **Anforderung** (requirement)
 - (1) Eine Vorschrift, die zu **erfüllende Kriterien** enthält. [ISO 24765]
 - (2) Eine Anforderung ist eine Aussage über die notwendige Beschaffenheit oder **Fähigkeit, die ein System oder Systemteile erfüllen** oder besitzen muss, um einen Vertrag zu erfüllen oder einer **Norm**, einer Spezifikation oder anderen, formell vorgegebenen Dokumenten zu entsprechen.
- **Debugging**
Tätigkeit des **Lokalisierens**/Identifizierens, Analysierens und **Entfernens** der Ursachen von Fehlerwirkungen in der Software (i.d.R durch den Entwickler)
- **Qualität** (lat.: qualitas = Beschaffenheit, Eigenschaft, Zustand)
 - (1) Der Grad, in dem ein System, eine Komponente oder ein Prozess die **Kundenerwartungen** und -bedürfnisse **erfüllt**.
 - (2) Eine vom Benutzer benötigte **Eigenschaft** oder Fähigkeit, **die eine Software erfüllen** oder besitzen muss, um einen Vertrag, einen Standard, eine Spezifikation oder ein anderes formales Dokument zu erfüllen



Begriffe

2 von 2

- **Testziel** (test objective)
Ein Grund oder **Zweck** für den Entwurf und die **Ausführung von Tests**.
- **Testen** (testing)
Der **Prozess**, der aus allen **Aktivitäten** des Lebenszyklus besteht (sowohl statisch als auch dynamisch), die sich mit der Planung, Vorbereitung und **Bewertung** eines Softwareprodukts und dazugehöriger **Arbeitsergebnisse** befassen. Ziel des Prozesses ist sicherzustellen, dass diese allen festgelegten Anforderungen genügen und etwaige Fehlerzustände zu finden.
- **Testobjekt** (test object)
Die Komponente oder das System, welches getestet wird (Testelement).
- **Testbedingung** (test condition)
Ein **Aspekt der Testbasis**, der für die **Erreichung bestimmter Testziele** relevant ist. Ist definiert als eine Einheit, ein Ereignis, eine Transaktion, ein **Qualitätsmerkmal**, ein strukturelles Element, welches durch einen oder mehrere **Testfälle** überprüft werden kann.

Typische Ziele des Testens

- **Fehlervermeidung** durch die Bewertung von Arbeitsergebnissen
 - Anforderungen, User Stories, Architekturdesign, Code
- Sicherstellen, dass alle spezifizierten **Anforderungen erfüllt** sind (verifizieren)
 - Konformität der **vertraglichen, rechtlichen, regulatorischen** Anforderungen prüfen
 - Nachweis der Einhaltung von branchenüblichen Standards und Normen liefern
- Das Testobjekt auf **Vollständigkeit** prüfen
- Nachweis über die vom User erwartete **Funktionalität** liefern (validieren)
- Vertrauen in die **Qualität** der gelieferten Software erzielen
- Aufdecken von Fehlerzuständen und -wirkungen, um das **Risiko** ungenügender Softwarequalität **zu minimieren**
- Stakeholdern belastbare **Informationen** (u.a. Metriken) geben
 - Zur Entscheidungsfindung über den Stand und das Qualitätsniveau des Testobjekts

Kontextabhängige Testziele

Abhängig vom Softwareentwicklungslebenszyklus-Modell

Die Ziele des Testens variieren je nach eingesetztem Softwareentwicklungslebenszyklus-Modell und der zu testenden Teststufe

z.B. beim Komponententest

- So viele Fehlerwirkungen wie möglich erzeugen, um Fehlerzustände zu beheben
- Die Codeüberdeckung der Komponenten (Units, Module) erhöhen

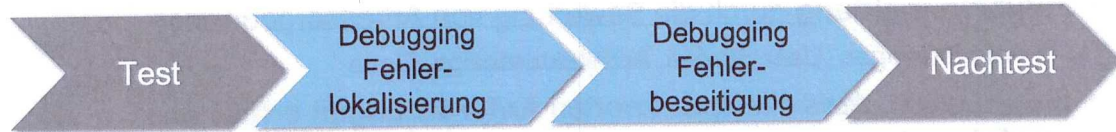
z.B. beim Abnahmetest

- Funktioniert das System wie erwartet?
- Wurden die Anforderungen erfüllt (Vertrauen in die Qualität)?
- Kann die Freigabereife durch die Stakeholder bewertet werden?

z.B. beim Wartungstest

- Ist die Altfunktionalität sichergestellt?
- Wurden versehentlich neue Fehlerzustände eingebaut?

Testen und Debugging (testing, debugging)



Test und Fehlernachtest sind Tester-Aktivitäten

Beim dynamischen Test können **Fehlerwirkungen** aufgezeigt werden, die auf Fehlerzuständen in der Software basieren. Das **Nachtesten** durch den Tester stellt sicher, dass die ursprünglichen Fehlerzustände behoben wurden

Debugging ist Entwickler-Aktivität

Beim Debugging (Fehlerlokalisierung und -beseitigung) werden die **Fehlerzustände** lokalisiert, analysiert und entfernt. Entwickler führen i.d.R. die hierzu gehörigen Komponententests bzw. Komponenten-Integrationstests (kontinuierliche Integration) durch

Beispiel: Im agilen Umfeld können auch Tester am Debugging und im Komponententest beteiligt sein

Zusammenfassung

- **Testen** selbst ist ein **Prozess** innerhalb des Entwicklungsprozesses:
 1. **Testplanung, Testüberwachung und -steuerung**
 2. **Testanalyse**
 3. **Testentwurf**
 4. **Testrealisierung**
 5. **Testdurchführung**
 6. **Testabschluss**
- **Testen** hat unterschiedliche **Ziele**, z. B. das Messen der aktuellen Qualität oder das Finden von Fehlerzuständen und -wirkungen
- **Frühes** Testen anstreben, denn **spät** entdeckte Fehlerwirkungen sind teuer
- **Dynamische** Tests führen Programme aus, **statische** Tests nicht
- Beim dynamischen Test suchen Tester **Fehlerwirkungen** und protokollieren diese (**testen**), Entwickler suchen **Fehlerzustände** und beseitigen diese (**debuggen**)

Kapitel I – Grundlagen des Softwaretestens

- I/1 Was ist Testen?
- I/2 Warum ist Testen notwendig?
- I/3 Sieben Grundsätze des Softwaretestens
- I/4 Testprozess
- I/5 Die Psychologie des Testens

I. Grundlagen des Softwaretestens » 2. Warum ist Testen notwendig?

Der Beitrag des Testens zum Erfolg und zur Wertschöpfung der Unternehmen!

Senkung des Risikos von Fehlerzuständen und -wirkungen (beim Einsatz)

Gründliches Testen von Komponenten und Systemen sowie der Dokumentation hilft, das Risiko von Fehlerwirkungen zu mindern, die im Betrieb auftreten können.

Steigerung der Qualität

Das Finden von Fehlerzuständen und Fehlerwirkungen sowie in Folge deren Behebung tragen zur Verbesserung der Qualität von Komponenten oder Systemen bei.

Erfüllung von Auflagen

Tests können durch den Auftraggeber oder Gesetzgeber vorgeschrieben werden oder erforderlich sein, um Industrienormen zu erfüllen (z.B. BAIT des BaFin, GoBD, TÜV).

Beispiel 1: Mars Climate Orbiter



Inch und Fuß statt Meter

Man möchte fast an einen Aprilscherz glauben: Die NASA gab gestern bekannt, warum ihre Sonde Mars Climate Orbiter beim Anflug auf den Mars verloren ging: Bei der Kommunikation zwischen zwei Kontrollteams wurden unterschiedliche Maßeinheiten verwendet.

„Menschen machen manchmal Fehler“, meinte Dr. Edward Weiler von der NASA und das Problem sei nicht der Fehler an sich gewesen, sondern die Tatsache, dass dieser von den internen Kontrollmechanismen nicht entdeckt wurde. Das Ergebnis der ersten Untersuchungen, warum die Sonde Mars Climate Orbiter vor einer Woche verloren ging, ist relativ banal: Beim Datenaustausch zwischen dem Missions-Navigationsteam in Kalifornien und dem Mars Climate Orbiter-Team in Colorado wurden unterschiedliche Maßeinheiten verwendet: Ein Team benutzte englische Einheiten wie Inches und Fuß, das andere das metrische System. So wurden die Daten für den Anflug auf den roten Planeten falsch berechnet. (...)

Schon am 3. Dezember soll der Mars Polar Lander auf dem roten Planeten aufsetzen. Und einen weiteren Fehlschlag könnte sich die NASA wohl kaum erlauben.

Quelle: www.astronews.com/news/artikel/1999/10/9910-01.shtml

Beispiel 2: Tödliche Röntgenstrahlen

Der nächste Fehler stammt aus dem medizinischen Bereich. Dort gab es Probleme mit dem Therac 25, ein Gerät, welches zur Strahlentherapie eingesetzt wurde.

Das Gerät kann sowohl Beta- als auch Gammastrahlen erzeugen. Letztere werden dadurch erzeugt, dass man Elektronen mit sehr hoher Geschwindigkeit auf einen Metallblock schießt.

Nachdem Therac 25 mehrmals eine tödliche Strahlendosis abgeschossen hatte, bemerkte man, dass dieser Effekt auftritt, wenn man die Daten zu schnell in das Gerät eingibt. Dieser startete dann sofort, ohne dass sich der Metallblock in Position befand. Beim Vorgänger funktionierte alles noch elektro-mechanisch. Beim Nachfolger kam die Software zum Einsatz.

Quelle: www.giga.de/features/storyarchiv/00127860_die_schlimmsten/seite_4.html

Testen als Erfolgsfaktor

1 von 2

- **Problematische Inbetriebnahmen vermeiden**
 - Die Software enthält noch Fehlerzustände und erzeugt Fehlerwirkungen
 - Die Bedürfnisse der Stakeholder werden nicht erfüllt und das Vertrauen schwindet
- **Auswahl der Testverfahren durch Testkompetenz**
 - Einsatz geeigneter Testverfahren reduzieren Fehlerzustände
 - Die richtigen Testverfahren passend zu Teststufe und Zeitpunkt auswählen
- Fehlerzustände in **Anforderungen** aufdecken
 - Tester früh einbinden und Reviews (siehe Kapitel 3) durchführen
 - z.B. schon bei Anforderungsreviews und User-Stories-Refinements

Testen als Erfolgsfaktor

2 von 2

- **Enge Zusammenarbeit der Tester und Entwickler – Verständnis erzielen**
 - Entwurfsfehler durch Zusammenarbeit mit den Systementwicklern vermeiden
 - Fehlerzustände im Code durch Zusammenarbeit mit den **Entwicklern** vermeiden
- **Verifizierung und Validierung**
 - Prüfung der Umsetzung der Anforderungen und der Bedürfnisse der Stakeholder
 - Verifizierung und Validierung der Software durch Tester vor der Freigabe



Begriffe

- **Qualitätssicherung** (quality assurance)
Teil des Qualitätsmanagements, das darauf gerichtet ist, Vertrauen in die Erfüllung der Qualitätsanforderungen zu erzeugen. [ISO 9000]
- **Verifizierung** (verification)
Bestätigung durch Bereitstellung eines objektiven Nachweises, dass festgelegte Anforderungen erfüllt worden sind. [ISO 9000]
- **Validierung** (validation)
Bestätigung durch Bereitstellung eines objektiven Nachweises, dass die Anforderungen für einen spezifischen beabsichtigten Gebrauch oder eine spezifische beabsichtigte Anwendung erfüllt worden sind. [ISO 9000]

Qualitätssicherung vs. Testen

Qualitätsmanagement (organisationsübergreifend)

- Umfasst alle Aktivitäten zur Leitung und Steuerung von Qualität
- Qualitätssicherung und Qualitätssteuerung sind wichtige Bestandteile

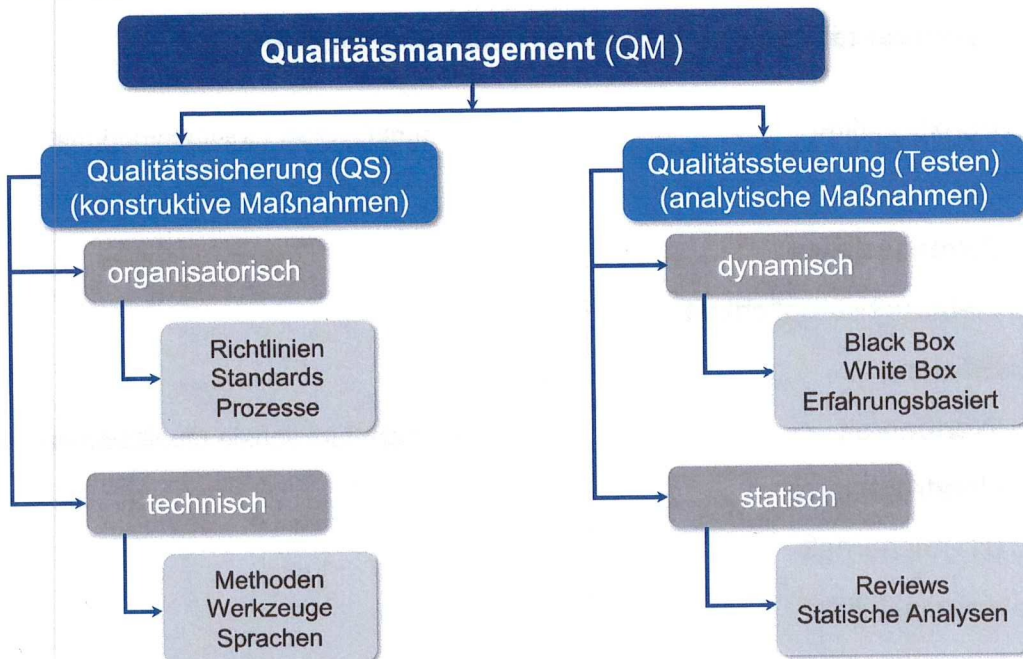
a) Qualitätssicherung (Quality Assurance)

- **Konstruktive** Maßnahmen zur **Fehlervermeidung**
- Einhaltung **gültiger Prozesse** zur Erreichung der angemessenen Reifegrade
- **Vermeidung** von **Fehlerzuständen** in Arbeitsergebnissen mit Hilfe der Prozesse
- Grundursachenanalyse zur Feststellung und Behebung von Fehlerzuständen
- **Prozessverbesserungen** durch Umsetzung der Befunde aus Retrospektiven
- Korrektes Testen durch korrekte Ausführung des gesamten Prozesses

b) Qualitätssteuerung (Quality Control)

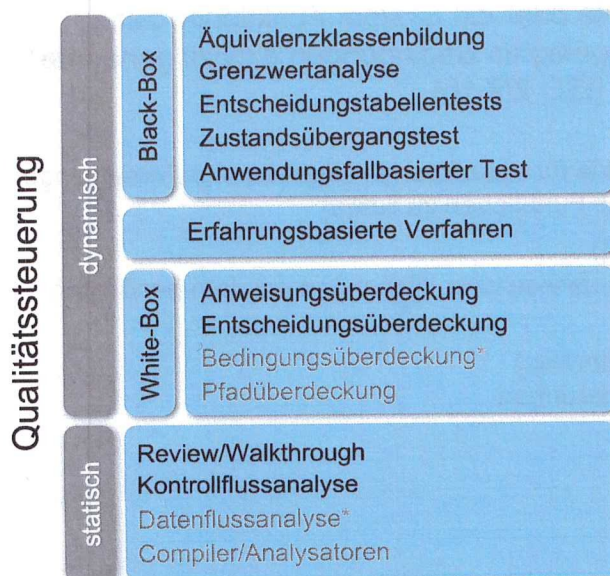
- **Aktivitäten (Testen und Reviews)** zur Erreichung von Qualitätsgraden
- **Testaktivitäten** als Bestandteil des SW-Entwicklungs- und -wartungsprozesses zwecks **Fehlerfindung**

Qualitätssicherung vs. Testen



Qualitätssteuerung (~Testen)

Zielt auf die Erhöhung der Produktqualität **mittels Prüf- und Testverfahren** ab.



Leitmotive

- Fehler sollen möglichst früh gefunden werden
- **Statische Tests** führen das Testobjekt nicht aus, finden Fehlerzustände
- **Dynamische Tests** führen das Testobjekt aus, finden Fehlerwirkungen

* Wird jetzt im Advanced Level behandelt

Softwarequalität

Softwarequalität setzt sich nach ISO 25010 zusammen aus

- Funktionalität
 - Zuverlässigkeit
 - Gebrauchstauglichkeit
 - Effizienz
 - Wartbarkeit
 - Übertragbarkeit
 - (IT-)Sicherheit
 - Kompatibilität
- funktionales Qualitätsmerkmal
- nicht-funktionale Qualitätsmerkmale



Funktionales Qualitätsmerkmal

- **Funktionale Eignung** (*functional suitability*)
Der Grad, zu dem eine Komponente oder ein System Funktionen zur Verfügung stellt, welche unter festgelegten Bedingungen explizit genannte und implizite Bedürfnisse erfüllen. [ISO 25010]
 - **Angemessenheit** (*suitability*)
Die Fähigkeit eines Softwareprodukts für spezifizierte Aufgaben und Zielsetzungen der Benutzer einen geeigneten Satz Funktionen zu liefern
 - **Korrektheit** (*functional correctness*)
Die richtigen oder vereinbarten Ergebnisse oder Wirkungen mit dem benötigten Grad an Genauigkeit zu liefern
 - **Vollständigkeit** (*functional completeness*)
Vollständige Umsetzung der Anforderungen



Nicht- funktionale Qualitätsmerkmale

1 von 3

- **Zuverlässigkeit** (reliability)

Der Grad, zu dem eine Komponente oder ein System seine spezifizierten Funktionen unter den festgelegten Bedingungen während einer bestimmten Zeitspanne ausführt. [ISO 25010]

Reife (maturity), Verfügbarkeit (availability), Fehlertoleranz (fault-tolerance), Wiederherstellbarkeit (recoverability)

- **Gebrauchstauglichkeit** (usability)

Der Grad, zu dem eine Komponente oder ein System durch bestimmte Benutzer in einem bestimmten Nutzungskontext genutzt werden kann, um festgelegte Ziele effektiv, effizient und zufriedenstellend zu erreichen. [ISO 9241-11]

Erlernbarkeit (learnability), Operabilität / Bedienbarkeit (operability), Barrierefreiheit (accessibility)



Nicht-funktionale Qualitätsmerkmale

2 von 3

- **Effizienz** (performance efficiency)

Eingesetzte Mittel im Verhältnis zu dem Ausmaß, in dem Benutzer spezifische Ziele erreichen. [ISO 9241-11]

Antwortzeitverhalten (time behaviour), Ressourcenverbrauch (resource utilisation), Kapazität (capacity)

- **Wartbarkeit** (maintainability)

Der Grad, zu dem eine Komponente oder ein System von den dafür vorgesehenen Personen gewartet werden kann.

Modularität (modularity), Wiederverwendbarkeit (reusability), Analysierbarkeit (analysability), Modifizierbarkeit (modifiability), Prüfbarkeit (testability)

- **Übertragbarkeit** (portability)

Die Einfachheit, mit der eine Software von einer Hardware- oder Software-umgebung in eine andere übertragen werden kann.

Anpassungsfähigkeit (adaptability), Installierbarkeit (installability), Austauschbarkeit (replaceability)



Nicht-funktionale Qualitätsmerkmale

3 von 3

- **(IT-)Sicherheit** (security)

Der Grad, zu dem eine Komponente oder ein System Informationen und Daten schützt, so dass Personen oder andere Komponenten oder Systeme nur einen solchen Grad an Zugriff erhalten, der ihrer Berechtigungsart und -stufe entspricht.

Vertraulichkeit (confidentiality), Integrität (integrity), Nachweisbarkeit (non-repudiation), Verantwortlichkeit (accountability), Authentizität (authenticity)

- **Kompatibilität** (compatibility)

Der Grad, zu dem eine Komponente oder ein System Informationen mit anderen Komponenten oder Systemen austauschen kann.

Interoperabilität (interoperability), Koexistenz (co-existence)

Fehlhandlungen, Fehlerzustände und Fehlerwirkungen

1. Fehlhandlungen erzeugen einen Fehlerzustand in einem Arbeitsergebnis

Gründe für Fehlhandlungen

Zeitdruck, menschliche Fehlbarkeit, Ungenauigkeit, Unerfahrenheit, Missverständnisse, Komplexität, fehlerhafte Anforderung, neue Technologie

2. Fehlerzustände in Software und Systemen können Fehlerwirkungen erzeugen, müssen aber nicht

3. Fehlerwirkungen durch Umgebungsbedingungen

z.B. im Betrieb der Software geänderte Hardware-Umgebung, Hardware-Defekte, Strahlungs- oder Temperatureinflüsse

Ausgetauschte Nachbarsysteme, Netzwerküberlastungen, Plattencrashes, Stromschwankungen, elektromagnetische Felder

„**Falsch positive**“ **Test-Ergebnisse** berichtete Fehlerzustände, die keine sind, z.B. Fehlerzustände in Testdaten, Testmitteln, Testumgebung (nicht im Testobjekt)

„**Falsch negative**“ **Test-Ergebnisse** Tests, die Fehlerzustände nicht entdecken, die sie aber hätten entdecken sollen

Fehlerzustände, Grundursachen und Wirkungen

Grundursachen

Ziel ist es, das Auftreten von ähnlichen Fehlerzuständen zu erkennen, zu analysieren und durch geeignete Maßnahmen, z.B. durch Schulungen, zu mindern. Die Konzentration auf die bedeutendsten Grundursachen bringt den meisten Erfolg

Grundursachenanalyse

Als Bestandteil der kontinuierlichen Testprozessverbesserung (KVP) kann die Grundursachenanalyse gefolgt von geeigneten Maßnahmen als Wirkung des Auftretens einer Vielzahl von zukünftigen Fehlerzuständen verhindern (Expert Level Test Management)

Beispiel:

Eine Häufung von falschen Zinsberechnungen, die durch Kundenreklamationen aufgefallen waren, beruhte auf einer Ungenauigkeit in einer User-Story, resultierend aus unzureichendem Wissen des Product Owners. Lösung: Schulung des Product Owners



Begriffe

- **Fehlhandlung** (error) [IEEE 610]
Die menschliche Handlung, die zu einem falschen Ergebnis führt.
- **Fehlerzustand / Defekt** (defect / bug)
Eine Unzulänglichkeit oder ein Mangel in einem Arbeitsergebnis, sodass es seine Anforderungen oder Spezifikationen nicht erfüllt.
- **Fehlerwirkung** (failure)
Ein Ereignis in welchem eine Komponente oder ein System eine geforderte Funktion nicht im spezifizierten Rahmen ausführt. [ISO 24765]
- **Grundursache** (root cause)
Die Ursache eines Fehlerzustands. Wenn man sie behebt, dann wird das Vorkommen der Fehlerart reduziert oder eliminiert. [CMMI]
- **Fehlschlag** (failure)
Ein Test(-fall) schlägt fehl, wenn das aktuelle Ergebnis nicht mit dem vorausgesagten Ergebnis übereinstimmt.

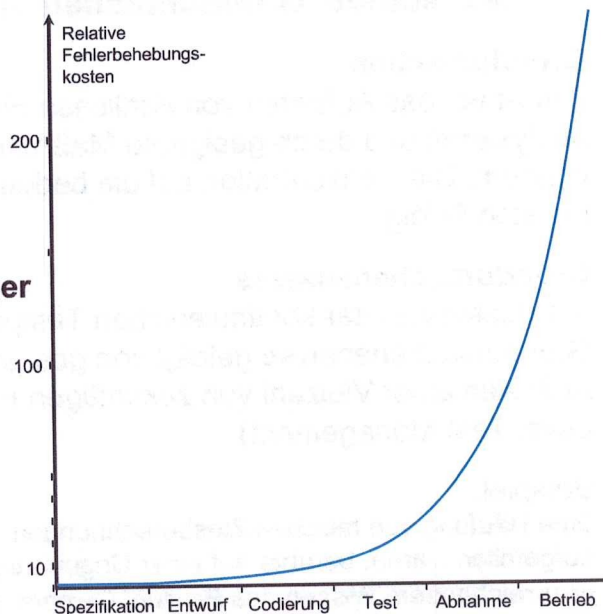


Fehlerkosten

Die **Fehlerbehebungskosten** steigen mit der Verweildauer des Fehlers im System

Je **früher** ein Fehler im Softwareentwicklungsprozess gefunden wird, desto **kostengünstiger** wird seine Behebung

Beispiel: Ein Spezifikationsfehler der erst im laufenden Betrieb gefunden und behoben wird, kann ein erneutes Durchlaufen aller dazwischenliegenden Phasen bedeuten!



Quelle: nach Boehm (1981)

Zusammenfassung

- **Softwarefehler** können verheerende Auswirkungen haben und sehr teuer sein
- Durch **Fehlhandlungen** können **Fehlerzustände / Defects** in das Testobjekt kommen, die sich mit **Fehlerwirkungen** zeigen
- Standard ISO 25010 regelt den Begriff der **Softwarequalität**. Funktionale und nicht-funktionale **Qualitätsmerkmale** bestimmen die Gesamtqualität
- Das Qualitätsmanagement umfasst **Qualitätssicherung (konstruktive Maßnahmen)** und **Qualitätssteuerung (analytische Maßnahmen)**
- Software QS mit Fehlerfindung, -korrektur und Qualitätssicherung
- Ähnliche Fehlerzustände, **Grundursachenanalyse** zur Prozessverbesserung und Fehlerverhinderung

Kapitel I – Grundlagen des Softwaretestens

- I/1 Was ist Testen?
- I/2 Warum ist Testen notwendig?
- I/3 Sieben Grundsätze des Testens
- I/4 Testprozess
- I/5 Die Psychologie des Testens

Grundsatz 1: Testen zeigt die Anwesenheit von Fehlerzuständen (Testing shows the presence of defects, not their absence)

- **Testen zeigt das Vorhandensein von Fehlerzustände**

Abweichung während des Testens zeigt Fehlerwirkung

- **Testen zeigt nicht die Abwesenheit von Fehlerzuständen**

Ausreichendes Testen verringert die Wahrscheinlichkeit des Vorhandenseins von unentdeckten Fehlerzuständen; mit Testen lässt sich jedoch nicht beweisen, dass ein Testobjekt frei von Fehlern wäre.

Keinen Fehlerzustand gefunden zu haben, bedeutet keinen Beweis für Korrektheit.

Hempels Paradox oder **Rabenparadox** ist ein nach dem Philosophen Carl Gustav Hempel benanntes Problem der Erkenntnistheorie. Das Paradoxon besteht darin, dass eine Allaussage über die Eigenschaft bestimmter Objekte scheinbar durch Beobachtungen beliebiger anderer Objekte ohne diese Eigenschaft bestätigt werden kann.



Grundsatz 2: Vollständiges Testen ist nicht möglich (Exhaustive testing is impossible)

- **Vollständiges Testen**

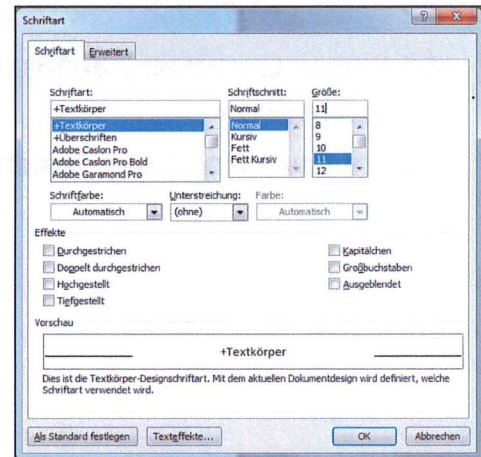
Ein vollständiger Test berücksichtigt alle möglichen Eingabewerte und deren Kombinationen mit allen denkbaren Vorbedingungen

- **Testfallexplosion**

Bezeichnung für den (mit der Anzahl der Parameter) exponentiell steigenden Aufwand von Tests bei einem vollständigen Test

- **Stichprobentest**

Durchführung eines Tests mit einer (methodisch oder zufällig ausgewählten) Stichprobe aller möglichen Eingabewerte als Testfälle



Tests sind immer nur **Stichproben** und der **Testaufwand** ist entsprechend Risikoanalyse, Testverfahren und Priorität festzulegen.

Grundsatz 3: Frühes Testen spart Zeit und Geld (Early testing saves time and money)

Je früher ein Fehlerzustand entdeckt wird, desto günstiger seine Behebung

Am Günstigsten ist die Fehlerbehebung **vor** Umsetzung des Fehlers

Auch **Konzepte**/Spezifikationen können bereits **getestet** werden

Unentdeckte **Konzeptfehler** können teure Folgefehler verursachen

Testen beginnt schon früh im Softwarelebenszyklus

Reduktion und Vermeidung von kostenintensiven, nachträglichen Änderungen

Testziele definieren und verfolgen

„**Früher anfangen**“ ist eine einfache, aber sehr wirkungsvolle Strategie zur **Risikoreduktion**.



Grundsatz 4: Häufung von Fehlerzuständen (Defect clustering)

Je Modul erfolgt eine Schätzung über die erwartete **Fehlerdichte**

Der Testaufwand ergibt sich proportional zur Schätzung

Die reale Fehlerdichte zeigt, welche Module die meisten Fehlerzustände beinhalten, ein wichtiger Bestandteil der Risikoanalyse

Oft enthalten wenige Module die meisten **Fehlerzustände** oder sind für die meisten **Fehlerwirkungen** im Betrieb verantwortlich

Wo ein **Fehler** (-zustand oder -wirkung) **gefunden wurde**, oft **stecken noch mehr!**

Fehler (-zustände oder -wirkungen) treten meist **gehäuft** auf, wie Pilze oder Kakerlaken ☺.



Grundsatz 5: Vorsicht vor dem Pestizid-Paradoxon (Beware of the pesticide paradox)

Das wiederholte Ausführen **immer gleicher Testfälle** zeigt irgendwann keine neuen Fehlerzustände auf*

Testfälle sind **regelmäßig** zu prüfen und ggf. zu modifizieren oder neue zu erstellen, um neue Fehlerzustände zu finden

Tests bei verändertem Testgegenstand

Wiederholungen von bereits durchgeführten Testfällen können jedoch sinnvoll sein, z. B. nach Programmiererweiterungen, Korrekturen, Betriebssystemwechsel

Bei automatisierten Regressionstests hat das Pestizid-Paradoxon einen vermeintlich positiven Effekt (geringe Anzahl Regressionsfehler)

Bei Testfällen, von denen angenommen wird, dass sie mehrfach wiederholt werden, kann sich **Testautomatisierung** lohnen

*) so wie Pestizide irgendwann nicht mehr wirksam gegen Insekten sind

Grundsatz 6: Testen ist kontextabhängig (Testing is context dependent)

Testen in Abhängigkeit des zu testenden Systems

- Unterschiedliche Testgegenstände sind **unterschiedlich zu testen**
- Motorsteuerung eines PKW braucht **andere Tests** als eine mobile E-Commerce-Applikation eines Versandhauses
- Sicherheitskritische Systeme werden anders getestet als Systeme, die z. B. Leib und Leben nicht gefährden
- Das Testen in agilen Projekten unterscheidet sich vom Testen bei sequentiellen Softwareentwicklungslebenszyklus-Modellen

Testumgebung vs. Produktionsumgebung

- Ein Test sollte nicht in der Produktionsumgebung, aber in einer **produktionsnahen** Umgebung durchgeführt werden
- Immer vorhandene Abweichungen zwischen den beiden Umgebungen beeinträchtigen die Aussagekraft von Tests

Grundsatz 7: Trugschluss: „Keine Fehler“ bedeutet ein brauchbares System (Absence-of-errors fallacy)

Alle Fehlerzustände wird das Testen in der Regel nicht finden (siehe Grundsätze 1 und 2). Aber die **wichtigsten** Fehler sollten entdeckt worden sein!

Das alleine ist aber noch keine Aussage über die Qualität der abgelieferten Software oder ein brauchbares System!

Die **Bedürfnisse** und **Erwartungen** der Nutzer können vom Leistungsumfang der tatsächlich gelieferten Software abweichen

Die Qualität kann geringwertiger sein, als bei vergleichbaren Systemen

Qualität (im Sinne von „Erfüllen der Erwartungen“) **kann nicht** nachträglich in eine Software **hinein getestet werden**, sondern muss bereits vorne **hinein konstruiert werden!**



Zusammenfassung

- **Tests** können Fehlerzustände in der Software finden, aber **nicht Fehlerfreiheit** nachweisen
- **Vollständiges Testen** von (nicht-trivialen) Programmen ist unmöglich, daher ist i. d. R. eine Stichprobenbildung erforderlich
- **Frühes Testen** hilft Kosten zu sparen, denn je früher ein Fehlerzustand entdeckt wird, desto kostengünstiger ist seine Behebung
- Fehlerzustände treten **gehäuft** auf: wo einer ist, sind oft auch noch andere
- Die identische **Wiederholung** von Testfällen findet keine weiteren Fehlerzustände
- Die konkrete Ausgestaltung der Tests **hängt vom Umfeld ab**
- „Fehlerfreiheit“ bedeutet nicht automatisch **Gebrauchstauglichkeit**

Kapitel I – Grundlagen des Softwaretestens

- I/1 Was ist Testen?
- I/2 Warum ist Testen notwendig?
- I/3 Sieben Grundsätze des Testens
- **I/4 Testprozess**
- I/5 Die Psychologie des Testens

Testprozess im Kontext

Es gibt keinen universellen Softwaretestprozess!

- Es gibt aber gebräuchliche Testaktivitäten
- Abhängig von der vorgegebenen Situation können die Aktivitäten variieren
- Die Teststrategie (des Unternehmens) legt fest, wann und wie diese Aktivitäten im Testprozess durchzuführen sind

Kontextabhängige Faktoren mit Einfluss auf den Testprozess

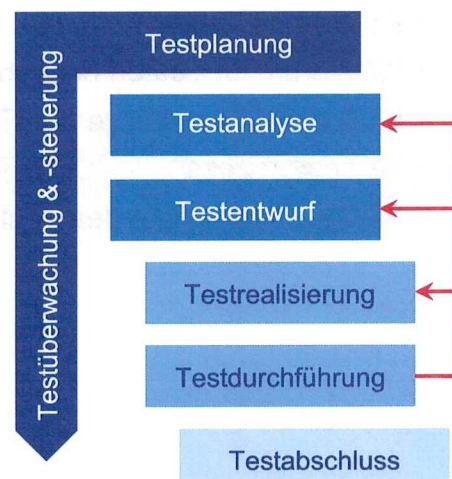
- Eingesetztes Softwareentwicklungslebenszyklus-Modell, **Projekumfang**
- **Produkt- und Projektrisiken** und die möglichen Teststufen und -arten
- Geschäftsbereich, Richtlinien des Unternehmens, interne und externe **Standards**
- Budget, Ressourcen, Fristen, Komplexität, regulatorische Anforderungen, **Verträge**

Die **Testbasis** benötigt messbare, definierte Überdeckungskriterien

- Nutzung als Key-Performance-Indicators (KPIs) zur Lenkung der Testaktivitäten
- Aufzeigen des Erreichens von Zielen des Softwaretests

Testaktivitäten und Aufgaben

- Der Testprozess besteht aus verschiedenen **Hauptgruppen** von Aktivitäten
- Jede Gruppe besteht aus mehreren **Einzelaufgaben**, die nach Projekt oder Release **variieren**
- Bei agiler Entwicklung finden auch die Testaktivitäten **iterativ** und **kontinuierlich** statt
- Aber auch bei sequentieller Entwicklung wird die **schrittweise, logische Abfolge** oft durch Überlappung, Kombination usw. verlassen. Daher ist die projekt- bzw. systemspezifische **Anpassung** der Hauptaktivitäten grundsätzlich **einzukalkulieren**





Begriffe zum Testprozess (~ in der Kompetenz des Test-Managers)

1 von 2

- **Testplanung** (test planning)
Eine Aktivität im Testprozess zur Erstellung und Fortschreibung des Testkonzepts.
- **Testüberwachung** (test monitoring)
Eine Aktivität des Testmanagers, die die Prüfung des Status der Testaktivitäten, das Identifizieren von Abweichungen vom geplanten oder erwarteten Status, und das Berichten über den Status an die Stakeholder beinhaltet.
- **Teststeuerung** (test control)
Die Managementaufgabe zur Entwicklung und Anwendung von Korrekturmaßnahmen, um in einem Testprojekt eine Abweichung vom geplanten Vorgehen zu beherrschen.
- **Testabschluss** (test completion)
Die Aktivitäten am Abschluss des Tests sammeln Daten aus beendeten Testaktivitäten, um Erfahrungen, Testmittel und andere relevante Informationen zu konsolidieren.



Begriffe zum Testprozess (~ in der Kompetenz des Testers)

2 von 2

- **Testanalyse** (test analysis)
Die Aktivität, die Testbedingungen durch eine Analyse der Testbasis identifiziert.
- **Testentwurf** (test design)
Die Aktivität, die Testfälle aus Testbedingungen ableitet und spezifiziert.
[Nach ISO 29119] *Siehe auch: Testentwurfsspezifikation.*
- **Testrealisierung** (test implementation)
Die Tätigkeit, die auf Basis der Testanalyse und des -Entwurfs die Testmittel vorbereitet, welche für die Testdurchführung benötigt werden.
- **Testdurchführung** (test execution)
Der Prozess der Ausführung eines Tests für eine Komponente oder ein System, der IST-Ergebnisse erzeugt.

Testplanung (test planning)

Definition der **Testziele**

- Ggf. unter Berücksichtigung der Qualitätsziele des Unternehmens

Festlegung der **notwendigen Testaktivitäten**

- Zum Erreichen des **Aufgabenumfangs** und
- Zum Erreichen der **Testziele**
- Unter Berücksichtigung der Beschränkungen aus dem **Umfeld**
- Festlegen geeigneter **Testverfahren/Aufgaben**
- Aufstellen eines Test(zeit)plan (test schedule) zur **Einhaltung der Termine / Meilensteine**
- Erstellen und Überarbeiten von Testkonzepten
- Planen von Überwachungs- und Steuerungsaktivitäten

Überwiegend in der Verantwortung des TM näheres dazu in Kapitel 5.2 (Testmanagement)



Begriffe zu Testplanung und Testanalyse

- **Testbasis** (test basis)
Alle Informationen, die als Basis für die Testanalyse und den Testentwurf verwendet werden können. [nach TMap*]
Die Dokumentation, auf der die Herleitung oder Auswahl der Testfälle beruht.
ANMERKUNG: Wenn ein Dokument nur über das formale Änderungsverfahren geändert werden kann, handelt es sich um eine festgelegte Testbasis. [Nach Tmap*]
- **Testobjekt** (test object)
Die Komponente oder das System, welches getestet wird. *Siehe auch: Testelement.*
- **Testbedingung** (test condition)
Ein Aspekt der Testbasis, der für die Erreichung bestimmter Testziele relevant ist.
- **Verfolgbarkeit** (traceability)
Der Grad, zu dem eine Beziehung zwischen zwei oder mehr Arbeitsergebnissen hergestellt werden kann. (auch **bidirektionale** Verfolgbarkeit genannt)

* TMap® ist ein Modell im Bereich des Testens und der Qualitätssicherung von Software, in der alle anfallenden Aspekte, das Umfeld und die Vorgehensweise strukturiert werden.

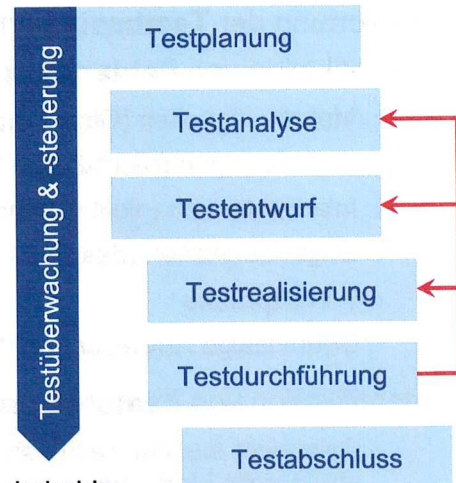
Testüberwachung und -steuerung

Testüberwachung (test monitoring)

- Prüfen des **tatsächlichen Fortschritts** gegen den Test(-zeit)plan per Überwachungsmetriken, definiert im Testkonzept
- **Messen und Analysieren** von Resultaten

Teststeuerung (test control)

- Überwachen und Fortschreiben des **Testfortschritts**, der erreichten **Testüberdeckung** und der **Endekriterien**
- Analyse der Testergebnisse und -protokolle
- Beurteilung des Grades der Systemqualität
- Treffen von **Entscheidungen** unter Einbezug der Stakeholder
- Anstoßen von **Korrekturmaßnahmen** (z.B. mehr Tests, da der Grad an Produktisikooberdeckung nicht erreicht werden kann)

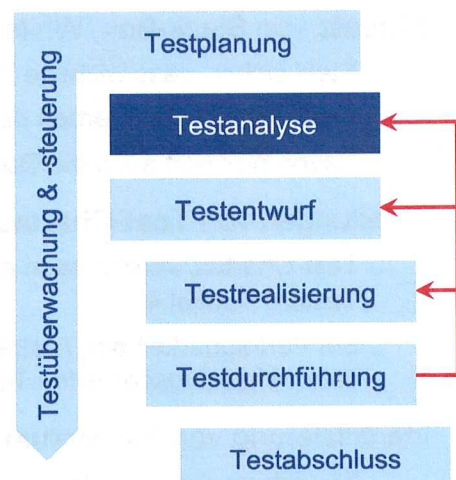


Testanalyse (test analysis) – „was ist zu testen“

1 von 3

Hauptaktivitäten Analyse der Testbasis

- Anforderungsspezifikationen, Fachkonzepte, User-Stories, Epics, Anwendungsfälle, funktionale und nicht-funktionale Anforderungen
- Systemarchitekturdiagramme, Aufrufdiagramme, Modelldiagramme (UML), Systemstruktur, Schnittstellenspezifikationen, etc.
- Realisierung der Komponente/des Systems, Code, Datenbankmetadaten, Datenbankabfragen, Schnittstellen
- Risikoanalyse über funktionale, nicht-funktionale, strukturelle Aspekte



Identifizierung und Priorisierung der **Testbedingungen**
Bestimmung der messbaren **Überdeckungskriterien**

Testanalyse

2 von 3

Bewertung der **Testbasis** und der **Testelemente**

Identifikation von **Fehlerarten**

- Mehrdeutigkeiten (Glossar fehlt)
- Auslassungen (Nicht-Zweig fehlt)
- Inkonsistenzen (nicht gleiche Terminologie)
- Ungenauigkeiten (das System soll performant sein)
- Widersprüche
- Überflüssige Anweisungen (Systemkontext- / Abgrenzung fehlt)

Identifikation von **Features** und **Feature-Sets**, die getestet werden müssen:

- Abgeleitet aus der Testbasis, Priorisierung, funktionale, nicht-funktionale und strukturelle Merkmale berücksichtigen, Risikograd, fachliche und technische Faktoren
- **Bidirektionale** Verfolgbarkeit (Testbasis ↔ **Testbedingungen** (↔ Testfällen))

Testanalyse

3 von 3

Einsatz von Black-Box, White-Box und erfahrungsbasierten Testverfahren

- Stellt sicher, dass wichtige Testbedingungen identifiziert und berücksichtigt werden
- Testbedingungen werden dadurch präziser definiert
- Davon profitiert auch die Qualität der Testfälle

Ableitungen von **Test-Chartas** (als Testziele aus den Testbedingungen)

- Test-Chartas werden beim erfahrungsbasierten Testen eingesetzt (Exploratives Testen, Kapitel 4)
- Bei Verfolgbarkeit der Testziele auf die Testbasis, kann eine Überdeckung auch beim erfahrungsbasierten Testen gemessen werden

Identifizierung von **Fehlerzuständen** (als Review)

- Bewertung der Anforderungen (s. Fehlerarten) und der Kundenbedürfnisse
- Bei Verfahren wie BDD, ATDD
- Generierung der Testbedingungen aus User-Stories etc. erkennen Fehlerzustände in User-Stories und den Abnahmebedingungen

Testentwurf (test design) – „wie wird getestet“

1 von 2

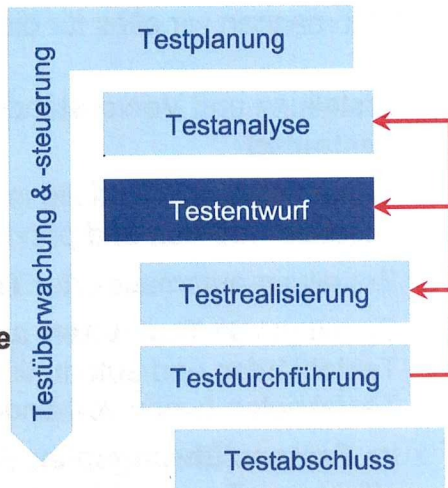
Ableitung von Testfällen aus Testbedingungen

- Ableiten und Priorisierung von (**abstrakten**) **Testfällen** (Testsets, etc.)
- Einsatz von Testverfahren (Kap. 4)
- Prüfung von Funktionen (**Positivtests**) und von Fehlersituationen (**Negativtests**)

Identifizierung benötigter Testdaten

Entwurf der Testumgebung und Testwerkzeuge

- Entwurf des Testumgebungsbaus
- Identifikation der benötigten **Infrastruktur** und Werkzeuge



Bidirektionale Verfolgbarkeit von **Testbasis** ↔ Testbedingungen ↔ **Testfällen** (↔ Testabläufen) erfassen

Identifizierung von Fehlerzuständen in der Testbasis während des Testentwurfs



Begriffe zum Testentwurf

2 von 2

• **Testfall** (test case)

Eine Menge von Vorbedingungen, Eingaben, Aktionen (falls anwendbar), erwarteten Ergebnissen und Nachbedingungen, welche auf Basis von Testbedingungen entwickelt wurden. [nach ISO 29119]

- Eindeutige Kennzeichnung
- Ausführungsvoraussetzungen (notwendige Vorbedingungen) (preconditions)
- Menge der Eingabewerte (input values)
- Menge der vorausgesagten Ergebnisse (expected results)
- Erwartete Nachbedingungen (postconditions)

• **Testdaten** (test data)

Die Daten, die erzeugt oder ausgewählt werden, um für die Durchführung eines oder mehrerer Testfälle Vorbedingungen zu erfüllen und Eingaben für die Durchführung bereit zu stellen. [Nach ISO 29119]

• **Testorakel** (test oracle)

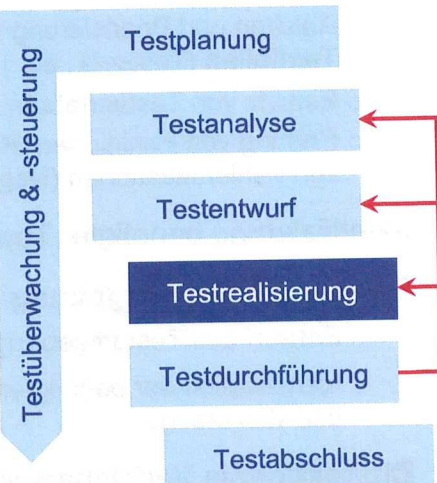
Eine Informationsquelle zur Ermittlung vorausgesagter Ergebnisse, um sie mit den tatsächlichen Ergebnissen eines Systems unter Test zu vergleichen. [Nach Adrion]

Testrealisierung (test implementation)

1 von 3

„Jetzt machen wir alles für die Durchführung bereit“

- Erstellung und Vervollständigung der **Testmittel**
- Reihenfolge der Testfälle in **Testabläufen** zusammenstellen und priorisieren
- Erstellung automatisierter **Testskripte**
- Erstellen von **Testsuiten** aus den Testabläufen und automatisierten Testskripten (wenn vorhanden)
- Im **Testausführungsplan** die Testsuiten zur effizienten Testdurchführung anordnen
- Aufbau der Testumgebung (Testrahmen, Simulatoren usw.) und Verifizierung, dass alles korrekt aufgesetzt wurde



Testrealisierung

2 von 3

- Vorbereitung von **Testdaten**
- Sicherstellen, dass sie **ordnungsgemäß** in die Testumgebung geladen sind
- Verifizierung der **bidirektionale** Verfolgbarkeit zwischen Testbasis ↔ Testbedingungen ↔ Testfällen ↔ **Testabläufen** ↔ **Testsuiten**
- Aufgaben des Testentwurfs und der Testrealisierung sind **oft kombiniert**
- Erfahrungsbasiertes Testen (exploratives Testen)
Testentwurf, -realisierung und -durchführung finden oft in direkter Folge statt. Basierend auf Test-Chartas werden diese Tests sofort ausgeführt, nachdem sie entworfen und realisiert wurden



Begriffe zur Testrealisierung

3 von 3

- **Testablaufspezifikation** (test procedure specification)
Die Spezifikation eines oder mehrerer Testabläufe. [nach ISO 29119] Auch bekannt als Testskript oder Testdrehbuch.

Die Testfälle werden in eine erforderliche sinnvolle Reihenfolge gebracht

- **Testsuite** (test suite)
Eine Menge von Testfällen oder Testabläufen, welche in einem bestimmten Testzyklus ausgeführt werden sollen.
- **Testausführungsplan** (test execution schedule)
Ein Zeitplan für die Ausführung von Testsuiten innerhalb eines Testzyklus.
- **Testmittel** (testware)
Die Arbeitsergebnisse resp. Artefakte, die während des Testprozesses erstellt werden und dazu gebraucht werden, um die Tests zu planen, zu entwerfen, auszuführen, auszuwerten und darüber zu berichten. [ISO 29119]
Dazu gehören: Dokumente, Skripte, Eingabewerte, erwartete Ergebnisse, Prozeduren zum Aufsetzen und Aufräumen von Testdaten, Dateien, Datenbanken, Umgebungen und weitere zusätzliche Software- und Dienstprogramme, die für das Testen verwendet werden [nach Fewster und Graham]

Testdurchführung (test execution)

1 von 3

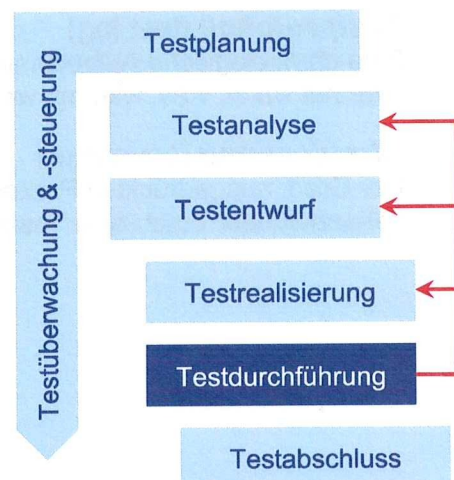
Der **Testausführungsplan** bestimmt die Reihenfolge des Ablaufs der **Testsuiten**

WICHTIG ist die **Protokollierung** der IDs und Versionen

- Der Testelemente resp. des Testobjekts
- Der Testwerkzeuge und der Testmittel

Ausführen der Tests (manuell / autom.)

- Vergleich der Ist-Ergebnisse mit den **erwarteten Ergebnissen**



Testdurchführung

2 von 3

Aufzeichnung der Ergebnisse der Testdurchführung (**Testprotokoll**)
mit entsprechendem Status: z.B. bestanden / fehlgeschlagen / blockiert

Alle **Fehlerwirkungen / Abweichungen** werden protokolliert und **analysiert**,
um den Grund des Problems zu lokalisieren (Code / Daten / ...)

Berichten über Fehlerzustände auf Basis der beobachteten Fehlerwirkung

Wiederholung von Testaktivitäten

- Durchführung eines **korrigierten Tests**
- **Fehlernachtests** zur Prüfung der erfolgreichen Fehlerbehebung
- **Regressionstests** zur Sicherstellung der Altfunktionalität

Verifizierung und Aktualisierung der **bidirektionalen** Verfolgbarkeit

- Testbasis ↔ Testbedingungen ↔ Testfällen ↔ Testabläufen ↔ **Testergebnissen**



Begriffe zur Testdurchführung

3 von 3

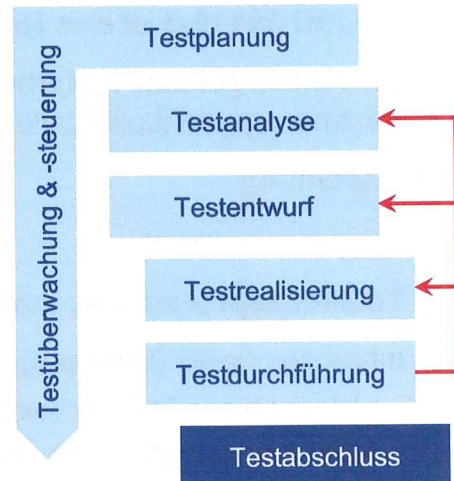
- **Testprotokoll** (test log)
Eine chronologische Aufzeichnung von Einzelheiten der Testausführung. [IEEE 829]
(wer hat wann, was, wie, mit welchen Ergebnissen getestet)
- **Überdeckung** (coverage)
Der Grad, ausgedrückt in Prozent, zu dem bestimmte Überdeckungselemente behandelt oder durch eine Testsuite ausgeführt wurden

Testabschluss (test completion)

1 von 2

Zeitpunkt für die **Testabschlussaktivitäten**

- Projektmeilensteine (Freigabe eines Systems)
 - Abschluss eines Testprojekts
 - Ende einer Projektiteration (agile)
 - Abschluss einer Teststufe
 - Abschluss eines Wartungsreleases
- Alle bekannten **Fehlerberichte** sollen geschlossen, auf ein späteres Release verschoben oder als Einschränkung akzeptiert worden sein
 - Erstellen eines **Testabschlussberichts** (durch den TM) in Rücksprache mit den Stakeholdern



Testabschluss

2 von 2

- **Finalisierung** und **Archivierung** von Testumgebung, Testdaten, Testinfrastruktur und anderer Testmittel (zur Wiederverwendbarkeit)
- Übergabe der **Testmittel** an das Wartungsteam, den Support / Kunden / Systemanwender, die einen Nutzen daraus ziehen
- Analyse der Erkenntnisse für notwendige Änderungen für zukünftige Iterationen, Produktreleases und Projekte
- Analyse und Dokumentation von „**lessons learned**“ zur Verbesserung der Prozessreife (KVP)*
 - Wie spät gefundene oder unerwartete Fehlerhäufungen
 - Unrealistische Testschätzungen
 - Fehlertendenzen aus Ursache-Wirkungs-Analyse
 - Zu späte Änderungsanträge

* KVP = kontinuierlicher Verbesserungsprozess

Testarbeitsergebnisse (test deliverable)

1 von 5

- Die **ISO 29119-3** ist eine Richtlinie für Testarbeitsergebnisse
- Die nachfolgenden Testarbeitsergebnisse können in Werkzeugen für Testmanagement und Defektmanagement erstellt und verwaltet werden

Testplanung

- Mastertestkonzept
- **Testkonzept** (eins oder mehrere z.B. je Qualitätsmerkmal oder Teststufe)
- Inhalt der dieser Arbeitsergebnisse sind
 - Informationen über die Testbasis
 - **Endekriterien** (definition-of-done)
 - Bezug aus **bidirektionale** Verfolgbarkeitsmatrix

Testarbeitsergebnisse

2 von 5

Testüberwachung und -steuerung

- **Testberichte**
- **Testfortschrittsberichte** (wiederholend, kontinuierlich / regelmäßig)
- **Testabschlussberichte** (bei Meilensteinen)

Inhalt dieser Arbeitsergebnisse sind

- Zielgruppenrelevante Details
(Testfortschritt, Berichtszeitpunkt, Testdurchführungsergebnisse)
- Antworten auf Projektmanagementfragen
(Aufgabenabschluss, Ressourcenzuordnung, Ressourcennutzung, Aufwand)

Testarbeitsergebnisse

3 von 5

Testanalyse

- **Priorisierte Testbedingungen** (bidirektionale Verfolgbarkeit, Überdeckung)
- Test-Charta (exploratives Testen)
- **Fehlerberichte** zu Fehlerzuständen der **Testbasis**

Testentwurf

- **Testfälle** bzw. Sets von Testfällen z.B. als **abstrakte Testfälle** zur Wiederverwendbarkeit sowie deren **bidirektionale** Verfolgbarkeit zu den Testbedingungen (Überdeckung)
- Entwurf und Identifizierung von **Testdaten**, der Testumgebung, der Infrastruktur, der **Testwerkzeuge**
- **Verfeinerung der Testbedingungen**, die in der Testanalyse definiert sind

Testarbeitsergebnisse

4 von 5

Testrealisierung

- Einen oder mehrere **Testabläufe**
- Testsuiten
- **Testausführungsplan** (für den kommenden Testzyklus)
- Service-Virtualisierung und automatisierte Testskripte
- Fertig- / Erstellung / Verifizierung von **Testdaten**
- Fertig- / Erstellung / Verifizierung der **Testumgebung**
- Überführung abstrakter Testfälle in **konkreten Testfällen** (konkret erwartete Ergebnisse, konkrete Testdaten, Nutzung eines Testorakels)
- Beim **explorativen** Testen können die Arbeitsergebnisse **erst während** der Testdurchführung erstellt werden
- **Verfeinerung der Testbedingungen**, die in der Testanalyse definiert sind

Testarbeitsergebnisse

5 von 5

Testdurchführung

- **Status** der Testfälle / Testabläufe (z.B. ausführbar, bestanden, fehlgeschlagen, blockiert, geplant ausgelassen)
- **Fehlerbericht** (s. Kap. 5.6)
- Dokumentation über konkreten Einsatz von Testelementen, Testwerkzeugen, Testmitteln
- Status jedes Elementes der Testbasis aus **bidirektionaler** Verfolgbarkeit sichtbar
- Verifizierung der Überdeckungskriterien
- Nachvollziehbar für die Stakeholder

Testabschluss

- **Testabschlussbericht(e)**, offene Punkte zur Verbesserung nachfolgender Projekte bzw. Iterationen (agile Retrospektive), Change Requests, Product-Backlog-Elemente, finalisierte Testmittel

Verfolgbarkeit zwischen Testbasis und Testarbeitsergebnissen

- Grundsätzlich können Testarbeitsergebnisse und deren Bezeichnungen stark variieren
- Für eine **effektive Testüberwachung** und -steuerung ist eine lückenlose **konsistente Verfolgbarkeit** über den gesamten Testprozess elementar wichtig
- Zusätzlich zur Bewertung der Testüberdeckung gibt es weitere Vorteile
 - Unterstützung bei der Auswirkungsanalyse von Änderungen
 - Nachvollziehbarkeit vom Testergebnissen
 - IT-Governance-Kriterien nachweisen
 - Verständlichkeit von Testfortschrittsberichten (Status der Elemente der Testbasis)
 - Transparenz technischer Aspekte für Stakeholder schaffen
 - Basis zur Beurteilung von Produktqualität, Prozessfähigkeit, Projektfortschritt

Testmanagementtools unterstützen Testarbeitsergebnisse und Verfolgbarkeit

Zusammenfassung – Prozess

Der **Testprozess** kann in Phasen aufgeteilt werden:

- **Testplanung:** Festlegung des Aufgabenumfangs, Definition der Testziele und Testaktivitäten
- **Testüberwachung und -steuerung:** Fortlaufende Kontrolle des Testfortschritts, ggf. Korrekturmaßnahmen
- **Testanalyse:** Review der Testbasis
- **Testentwurf:** Spezifikation der Testfälle und Testdaten
- **Testrealisierung:** Endgültige Festlegung der Testfälle und Testdaten
- **Testdurchführung:** Ausführung der Testfälle und Ergebnisvergleich, Abweichungen verfolgen
- **Testabschluss:** Konsolidierung der Daten, KVP

Kapitel I – Grundlagen des Softwaretestens

- I/1 Was ist Testen?
- I/2 Warum ist Testen notwendig?
- I/3 Sieben Grundsätze des Softwaretestens
- I/4 Testprozess
- I/5 Die Psychologie des Testens

Humanpsychologie und Testen

- Testen dient zum **Identifizieren von Fehlerzuständen**
Anforderungsreviews, User-Story-Verfeinerungssession (refinement), Fehlerwirkungen bei dynamischen Tests
- **Bestätigungsfehler** (Element der Humanpsychologie)
 - Informationen widersprechen der aktuellen Überzeugung (ist doch richtig)
 - Erschweren es Entwicklern, zu akzeptieren, dass der Code nicht korrekt ist
- Tester überbringen häufig schlechte Nachrichten, Überbringer schlechter Nachrichten werden oft dafür verantwortlich gemacht
- Testen wird als destruktive Aktivität gesehen

Gegenmaßnahme

- Informationen über Fehlerzustände und -wirkungen konstruktiv kommunizieren
- Abbau von Spannungen zwischen Testern und Analysten, Product Owner, Designern, Entwicklern

Eigenschaften eines guten Testers und Testmanagers

Gute Kommunikation und soziale Kompetenzen

- Zusammenarbeit statt Streit (**gemeinsames Ziel**: gute Qualität des Systems)
- Nutzen des Testens aufzeigen und Informationen über Fehlerzustände sammeln
 - Arbeitsergebnisse und Fähigkeiten zu verbessern
 - Zeit und Geld sparen, um das allgemeine Risiko für die Produktqualität zu reduzieren
- Sachliche, neutrale, faktenorientierte Darstellung der Testergebnisse
 - Objektive und tatsachenbasierte Fehlerberichte und Reviewbefunde
 - Keine Schuldzuweisungen, namentliche Beschuldigungen, persönliche Kritik
- Die Gefühle der anderen Person verstehen und respektieren
 - Perspektivwechsel vornehmen: „Wie würde ich mich an seiner Stelle fühlen?“
- Feedback einholen (wurde die Aussage verstanden?)

Klare Definition der Testziele (Team, Testmanagement, Stakeholder)

- Menschen passen ihre Pläne und ihr Verhalten gerne den Zielen an
- Tester sollten auch an diesen Zielen festhalten

Denkweisen von Entwicklern und Testern

1 von 2

Der Entwickler

- setzt die Anforderungen um
- entwickelt Strukturen
- programmiert die Software

Für ihn ist das Schaffen eines Produktes der Erfolg

Entwickler sind **konstruktiv!**

Der Tester

- plant seine Tests
- spezifiziert Testfälle
- testet und will Fehler finden

Für ihn ist das Fehlerfinden der Erfolg

Tester sind **destruktiv!**

Falsch!

Testen ist *konstruktiv*. Das Ziel ist, ein qualitativ hochwertiges Endprodukt mit möglichst wenigen Fehlerzuständen und -wirkungen auszuliefern!

Durch das Testen ist der Fehler nicht in das Programm gekommen!

Denkweisen von Entwicklern und Testern

2 von 2

• Denkweise eines Testers

- Neugier und gute Beobachtungsgabe, diese wächst und reift mit steigender Erfahrung
 - Hineindenken in die **Sichtweise** des Kunden
 - **Analysefähigkeit** für IT-Struktur der Testobjekte
 - Entdeckung **kleiner Details**, an denen sich Fehlerwirkungen zeigen
- Professioneller Pessimismus, kritische Einstellung, Detailgenauigkeit
- Motivation zu guter und positiver Kommunikation und Beziehungen

• Denkweise eines Entwicklers

- Lösungsorientiert: stärker interessiert an Entwurf und Erstellung von Lösungen
- Nachdenken, über Fehler in den Lösungen, wird vernachlässigt
- Bestätigungsfehler können es erschweren, Fehler in der eigenen Arbeit zu finden

Zusammenfassung

- Kein Mensch arbeitet fehlerfrei – in jeder Entwicklung passieren **Fehler**
- Die Natur des Menschen erschwert es, eigenen Fehlern offen gegenüberzustehen – **Fehlerblindheit**, u. Ä.
- Entwickler und Tester – es treffen „zwei Welten“ aufeinander:
 - **Entwickeln** ist „**konstruktiv**“ – es wird etwas erschaffen!
 - **Testen** erscheint auf den ersten Blick „**destruktiv**“ – Fehler sind zu finden. Dieses Bild täuscht jedoch!
 - **Entwickeln und Testen** sind zusammen betrachtet **konstruktiv** im Sinne einer besseren Qualität der Software

Schlüsselbegriffe

- Debugging
- Fehlerwirkung
- Fehlerzustand
- Fehlhandlung
- Grundursache
- Qualität
- Qualitätssicherung
- Verfolgbarkeit
- Testablauf
- Testabschluss
- Testanalyse
- Testausführungsplan
- Testbasis
- Testbedingung
- Testdurchführung
- Testdaten
- Testen
- Testentwurf
- Testfall
- Testmittel
- Testobjekt
- Testorakel
- Testplanung
- Testprozess
- Testrealisierung
- Teststeuerung
- Testsuite
- Testüberwachung
- Testziel
- Überdeckung
- Validierung
- Verifizierung

II. Testen im Softwareentwicklungslebenszyklus » Agenda



Kapitel II – Testen im Softwareentwicklungslebenszyklus

- II/1 Softwareentwicklungslebenszyklusmodelle
- II/2 Teststufen
- II/3 Testarten
- II/4 Wartungstest

Lernziele für Testen im Softwareentwicklungslebenszyklus 1 von 2

2.1 Softwareentwicklungslebenszyklusmodelle

- FL-2.1.1 (K2) Die Beziehungen zwischen Softwareentwicklungsaktivitäten und Testaktivitäten im Softwareentwicklungslebenszyklus erklären können
- FL-2.1.2 (K1) Gründe identifizieren können, warum Softwareentwicklungslebenszyklusmodelle an den Kontext des Projekts und an die Produktmerkmale angepasst werden müssen

2.2 Teststufen

- FL-2.2.1 (K2) Die unterschiedlichen Teststufen unter den Aspekten der Testziele, Testbasis, Testobjekte, typischen Fehlerzustände und Fehlerwirkungen sowie der Testvorgehensweise und Verantwortlichkeiten vergleichen können

Lernziele für Testen im Softwareentwicklungslebenszyklus 2 von 2

2.3 Testarten

- FL-2.3.1 (K2) Funktionale, nicht-funktionale und White-Box-Tests vergleichen können
- FL-2.3.2 (K1) Erkennen können, dass funktionale, nicht-funktionale und White-Box-Tests auf jeder Teststufe eingesetzt werden können
- FL-2.3.3 (K2) Den Zweck von Fehlernachtests und Regressionstests vergleichen können

2.4 Wartungstest

- FL-2.4.1 (K2) Auslöser für Wartungstests zusammenfassen können
- FL-2.4.2 (K2) Den Einsatz der Auswirkungsanalyse im Wartungstest beschreiben können

Kapitel II – Testen im Softwareentwicklungslebenszyklus

- II/1 Softwareentwicklungslebenszyklusmodelle
- II/2 Teststufen
- II/3 Testarten
- II/4 Wartungstest

II. Testen im Softwareentwicklungslebenszyklus » 1. Softwareentwicklungslebenszyklusmodelle



Allgemeines

Softwareentwicklungslebenszyklusmodelle

Beschreiben werden Aktivitäten, die in jeder Phase durchgeführt werden und wie diese Aktivitäten logisch und zeitlich zueinander stehen, um einen standardisierten Projektablauf zu ermöglichen

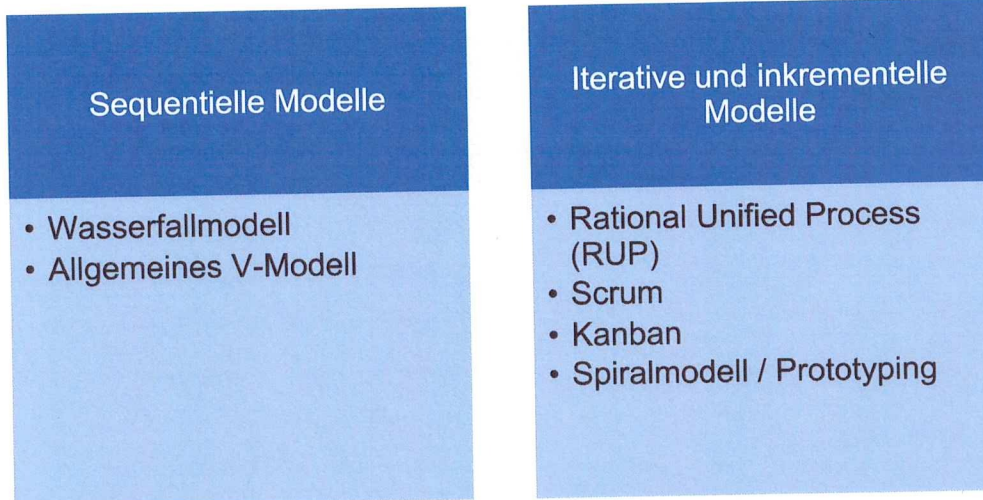
In jedem Softwareentwicklungslebenszyklusmodell gibt es mehrere Merkmale für gutes Testen:

- Jede Testaktivität bezieht sich auf eine Entwicklungsaktivität
- Jede Teststufe hat ihre stufenspezifischen Tests
- Testanalyse und Testentwurf beginnen während der Entwicklungsaktivität
- Tester beteiligen sich an Reviews von Arbeitsergebnissen schon bei ersten Entwürfen (Anforderungen, User Stories, Architekturdesign, etc.)

Verschiedene Modelle erfordern unterschiedliche Testansätze

Allgemeines

Kategorisierung von Softwareentwicklungslebenszyklusmodellen:

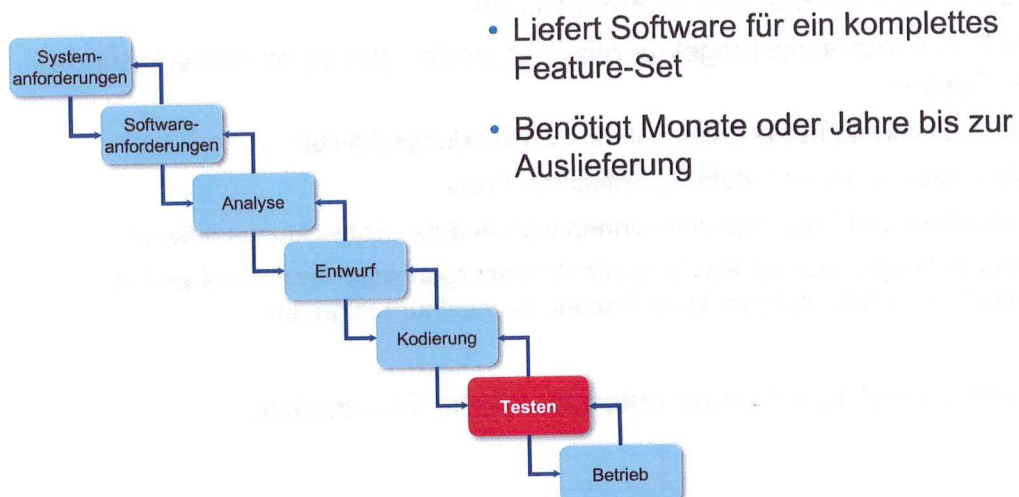


Softwareentwicklung und Softwaretesten

1 von 8

Wasserfallmodell (Royce 1970)

Linearer, sequentieller Ablauf von Aktivitäten. Alle Entwicklungsaktivitäten werden nacheinander abgeschlossen, bevor mit dem Testen begonnen wird



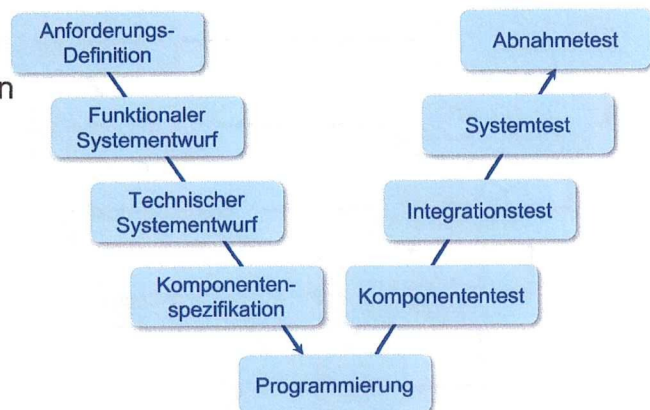


Softwareentwicklung und Softwaretesten

2 von 8

Allgemeines V-Modell (v-model)

- Ein **sequentielles** Entwicklungsmodell
 - Integration des Testprozesses in den Entwicklungsprozess (frühes Testen)
 - Zu jeder **Entwicklungsstufe** korrespondiert eine **Teststufe** gegenüber
- **Sequentielle** Durchführung der Tests in einer Teststufe
- **Überlappungen** sind zugelassen
- Liefert Software für ein komplettes Feature-Set
- Benötigt Monate oder **Jahre** bis zur Auslieferung



Softwareentwicklung und Softwaretesten

3 von 8

Vom **allgemeinen V-Modell** kann in der Praxis abgewichen werden, abhängig vom Projektvorgehen oder dem Produkt, mit mehr oder weniger Stufen

- Komponentenintegrationstests werden nach dem Komponententest durchgeführt
- Systemintegrationstests nach dem Systemtest

Entwicklungsdokumente, die während der Entwicklung entstehen, sind Basis für die Tests einer oder mehrerer Stufen

Standards für generische Arbeitspapiere stehen u.a. unter

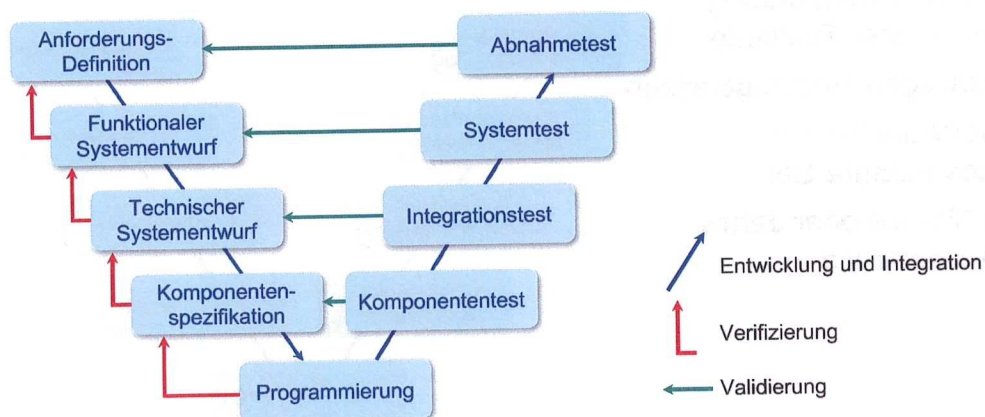
- CMMI (Capability Maturity Model Integration)
- ISO 12207 („Software life cycle processes“)

Verifizierung und Validierung werden während der Erstellung der Entwicklungsdokumente durchgeführt

Softwareentwicklung und Softwaretesten

4 von 8

- **Verifizierung** bedeutet, dass geprüft wird, ob die **Vorgaben**, die aus der **Vorstufe** übergeben wurden, korrekt umgesetzt worden sind
- **Validierung** bedeutet, dass die **Software** oder Komponente auf der jeweiligen Teststufe gegen **Benutzeranforderungen** und -bedarf geprüft wird



Softwareentwicklung und Softwaretesten

5 von 8

• Inkrementelle Entwicklung

- Anforderungsdefinition, Entwurf, Implementierung und Test werden in **kleinere Schritte** unterteilt, sodass sie inkrementell wachsen
- Die Größe eines Inkrements wird durch die Wahl der Methode bestimmt
- Ein Inkrement kann auch nur eine einzige Veränderung in einer GUI Maske oder Abfrageoption sein

• Iterative Entwicklung

- Eine Reihe von Zyklen für eine Gruppe von Features – **eine Iteration**
- Oft mit festgelegtem Zeitrahmen für Spezifizierung, Entwurf, Implementierung und Test
- Eine Iteration kann Änderungen von **Leistungsmerkmalen** früherer Iterationen und Erweiterungen oder Änderungen am **Projektumfang** umfassen
- Jede Iteration liefert eine **lauffähige Software** (wachsende Teilmenge des Gesamten)
- Am Ende wird die endgültige Software geliefert oder die Entwicklung ist beendet
- Beispiele: RUP, Scrum, Kanban, Spiralmodell / Prototyping

Softwareentwicklung und Softwaretesten

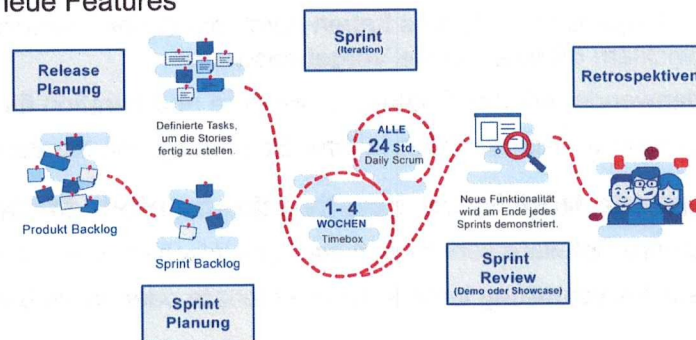
6 von 8

• Rational Unified Process (RUP)

- Jede Iteration ist relativ lang (zwei bis drei Monate)
- Die Inkremente der Features sind entsprechend groß (zwei bis drei Gruppen)

• Scrum

- Jede Iteration ist relativ kurz (Stunden, Tage, wenige Wochen)
- Die Inkremente der Features sind relativ klein. Sie umfassen Verbesserungen oder zwei bis drei neue Features



Certified Tester Foundation Level

© trendig technology services GmbH

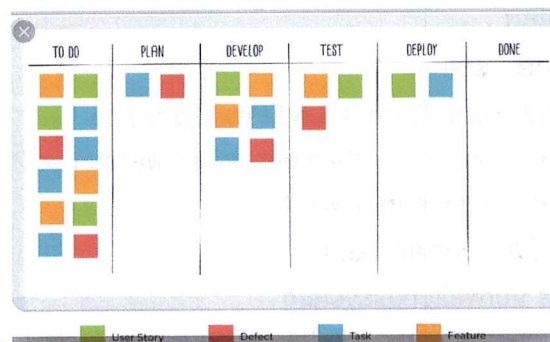
108

Softwareentwicklung und Softwaretesten

7 von 8

• Kanban

- Die Länge einer Iteration wird nicht zwangsläufig vorgegeben (mit oder ohne WiP*)
- Der Umfang variiert von einer Verbesserung, einem Feature bis hin zu einer Gruppe von Features



*WiP=Work in Progress

• Spiralmodell / Prototyping

- Schaffen von experimentellen Inkrementen
- Diese werden in folgenden Iterationen stark überarbeitet oder aufgegeben

Softwareentwicklung und Softwaretesten

8 von 8

- **Oftmals gemeinsamer Einsatz von iterativen und inkrementellen Entwicklungsmodellen**
 - Jedes Feature sollte bis zur Auslieferung auf mehreren Teststufen getestet werden
 - Überlappungen in der Entwicklung und iterative Teststufen sind üblich
 - Kontinuierliche Auslieferung (continuous delivery) oder kontinuierliche Bereitstellung (continuous deployment) kommt oft zum Einsatz
 - Einsatz automatisierter Tests auf allen Teststufen als Teil von Delivery-Pipelines
 - Selbstorganisierten Teams haben durch die Zusammenarbeit von Testern und Entwicklern oft eine eigene Vorgehensweise
 - Endanwender können Feature für Feature und Iteration für Iteration freigeben
 - Regressionstests wachsen in ihrer Bedeutung von Iteration zu Iteration
- **Sequentielle Modelle vs. iterativ-inkrementelle Modelle**
 - Nutzbare Software schon nach wenigen Wochen oder Monaten
 - Gesamtfertigstellung kann trotzdem Monate oder Jahre brauchen

Softwareentwicklungslebenszyklusmodelle im Kontext

1 von 3

Auswahl und Anpassung eines Softwareentwicklungslebenszyklusmodells

- Auswahl und Anpassung erfolgt im Kontext von Projekt- und Produktmerkmalen
- Basis dafür können sein
 - Projektziel, identifizierte Projekt- und Produktrisiken
 - Frage nach dem Produkt, welches entwickelt werden soll
 - Geschäftsprioritäten (time-to-market)
- Unterschiede dafür können sein
 - Kleines internes Verwaltungssystem
 - Sicherheitskritisches System wie Bremssteuerung im Fahrzeug
 - Organisatorische und kulturelle Probleme



Softwareentwicklungslebenszyklusmodelle im Kontext

2 von 3

Begriff:

- **Kommerzielle Standardsoftware** (commercial off-the-shelf) (COTS)

Ein Softwareprodukt, das für den allgemeinen Markt entwickelt wurde, d.h. für eine große Anzahl von Kunden, und dass dieses in identischer Form an viele Kunden ausgeliefert wird.

- Teststufen und Testaktivitäten können kombiniert oder neu definiert werden
- Bei der Integration von Standardsoftware kann der Kunde z.B. fordern
 - Interoperabilitätstest auf Systemintegrationsstufe durchzuführen
 - Funktionale und nicht-funktionale Tests beim Abnahmetest

Softwareentwicklungslebenszyklusmodelle im Kontext

3 von 3

- Kombinationen von Softwareentwicklungsmodellen sind möglich
Beispiele

- V-Modell für ein Backend-System (Entwicklung und Tests) beim Integrationstest
- Agile Modelle für Entwicklung und Test der Frontend-Benutzerschnittstelle (UI)
- Prototyping in der Anfangsphase, dann übergeleitet in ein inkrementelles Modell

- Internet der Dinge (Internet of Things, IoT) (versch. Geräte, Dienste, ..)
 - Jedes Objekt in einem eigenständigem Modell
 - Herausforderung für die Entwicklung von Versionen
(Aktualisierung im laufenden Betrieb, Außerbetriebnahme)

Softwareentwicklungslebenszyklusmodelle anpassen

Gründe für die Anpassung der Softwareentwicklungsmodelle an Projekt- oder Produktanforderungen können sein

- **Unterschiedliche Produktrisiken** der Systeme (komplexe, einfache, sicherheitskritische Projekte)
- Viele unterschiedliche Organisationseinheiten sind an der Entwicklung beteiligt und müssen zusammengeführt werden (**Mischung** aus sequenziellen und agilen **Methoden**)
- Marktanforderungen verlangen **schnelle Lieferungen** (Teststufen oder Testarten werden zusammengefasst)
- Unterschiedliche **Mitarbeitererfahrungen** oder -fähigkeiten (Scrum, Kanban, V-Modell, Wasserfall)

Zusammenfassung

- Softwareentwicklungslebenszyklusmodelle werden zur **Softwareentwicklung** genutzt und schließen Testaktivitäten mit ein. Bei Bedarf werden diese Modelle **an Projekt- und Produktanforderungen angepasst**
- Das bekannteste sequentielle Entwicklungsmodell ist das **V-Modell**, welches Entwicklungs- und Teststufen auf zwei Ästen darstellt, die einander komplementär gegenüber stehen
- Wichtige **iterativ-inkrementelle Entwicklungsmodelle** sind RUP und agile Modelle
- **Verifizierung**: Prüfung, ob die Ergebnisse einer Entwicklungsphase die Vorgaben der Phaseneingangsdokumente erfüllen. Mathematisch formaler Beweis der Korrektheit eines Programms
- **Validierung**: Prüfung, ob ein Entwicklungsergebnis die individuellen Anforderungen bezüglich einer speziellen beabsichtigten Nutzung erfüllt

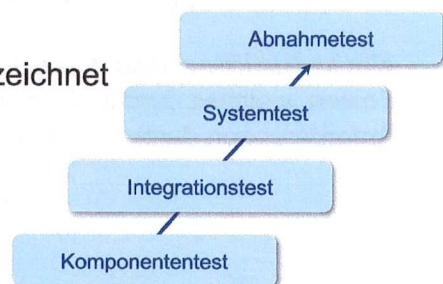
Kapitel II – Testen im Softwarelebenszyklus

- II/1 Softwareentwicklungslebenszyklusmodelle
- II/2 Teststufen
- II/3 Testarten
- II/4 Wartungstests

II. Testen im Softwareentwicklungslebenszyklus » 2. Teststufen: Komponententest

Teststufen

- Gruppen von Testaktivitäten, die gemeinsam organisiert und verwaltet werden
- Jede Teststufe ist eine eigene Instanz des Testprozesses mit entsprechenden Aktivitäten
- Umfassen Testaktivitäten entsprechend einer festgelegten Entwicklungsstufe
- Stehen mit anderen Aktivitäten innerhalb des SW-Entwicklungslebenszyklusses in Verbindung
- Benötigen eine passende Testumgebung
- Werden durch folgende Eigenschaften gekennzeichnet
 - Spezifische Ziele
 - Testbasis, daraus werden die Testfälle abgeleitet
 - Testobjekt (was getestet wird)
 - Typische Fehlerzustände und Fehlerwirkungen
 - Spezifische Ansätze und Verantwortlichkeiten





Begriffe zu Teststufen

1 von 2

- **Teststufe** (test level)
Eine spezifische Instanziierung eines Testprozesses.
- **Komponententest** (component testing)
Testen einer einzelnen Hardware- oder Softwarekomponente.
- **Integrationstest** (integration testing)
Testen mit dem Ziel, Fehlerzustände in den Schnittstellen und im Zusammenspiel zwischen integrierten Komponenten aufzudecken.
Siehe auch **Komponentenintegrationstest**, **Systemintegrationstest**
- **Systemtest** (system testing)
Testen eines integrierten Systems, um sicherzustellen, dass es spezifizierte Anforderungen erfüllt. [Hetzel]
- **Abnahmetest** (acceptance testing)
Formales Testen hinsichtlich der Benutzeranforderungen und -bedürfnisse bzw. der Geschäftsprozesse. Es wird durchgeführt, um einem Auftraggeber oder einer bevollmächtigten Instanz die Entscheidung auf der Basis der Abnahmekriterien zu ermöglichen, ob ein System anzunehmen ist oder nicht. [IEEE 610]



Begriffe zu Teststufen

2 von 2

- **Testumgebung** (test environment)
Eine Umgebung, die benötigt wird, um Tests auszuführen. Sie umfasst Hardware, Instrumentierung, Simulatoren, Softwarewerkzeuge und andere unterstützende Hilfsmittel. [IEEE 610]
- **Funktionales Testen** (functional testing)
Testen, welches durchgeführt wird, um die Erfüllung der funktionalen Anforderungen durch eine Komponente oder ein System zu bewerten.
- **Nicht-funktionales Testen** (non-functional testing)
Testen, welches durchgeführt wird, um die Erfüllung der nicht-funktionalen Anforderungen durch eine Komponente oder ein System zu bewerten.

Komponententest (Unit-, Modultest)

Test der Komponenten auf innere Verarbeitung

- **Ziele**

- Risikoreduktion
- Verifizierung gegen Entwicklungsspezifikation (funktionale und nicht-funktionale Verhaltensweisen)
- Vertrauen in die Qualität der **Komponente** schaffen
- Weitergabe von Fehlerzuständen in die höheren Teststufen verhindern
- Automatisierte Komponententests (agile Entwicklung), bauen Vertrauen auf, dass bei Änderungen keine bestehenden Komponenten beschädigt wurden

- **Testbasis**

- Feinentwurf
- Code
- Datenmodelle
- Komponentenspezifikationen

- **Testobjekte**

- Komponenten, Units oder Modelle
- Code, Datenstrukturen
- Klassen
- Datenbankmodule



Komponententest

- Ausschließlicher Test **einzelner Komponenten**

- Eine Komponente ist die kleinste Einheit, für die es eine **separate Spezifikation** gibt. Eine Komponente kann dabei aus mehreren kleinsten Bausteinen bestehen

- Jede Komponente wird isoliert vom Rest des Systems getestet

- Es soll nach **internen Fehlerzuständen** gesucht werden
- **Außer Betracht** bleiben **Wechselwirkungen** zu anderen Programmkomponenten

- Häufige **Fehlerzustände** und **Fehlerwirkungen**

- Nicht korrekte Funktionalität gegenüber der Entwurfsspezifikation
- Datenflussprobleme
- Nicht korrekter Code und/oder nicht korrekte Logik

- **Fehlerzustände**

- Behebung sofort, i.d.R. ohne formales Fehlermanagement durch Entwickler
- Grundursachenanalyse und Prozessverbesserung allerdings ohne Fehlerbericht schwer durchführbar

Komponententest – Funktionale und nicht-funktionale Tests

- **Test der Funktionalität**
 - Der Komponententest soll die **Funktionalität** der Komponente sicherstellen
 - Werden alle Funktionen **korrekt** ausgeführt?
 - Sind die Berechnungen korrekt?
 - Werden die Spezifikationen **vollständig** erfüllt?
- **Test der Effizienz**
 - Test auf die Verwendung von Ressourcen wie **Speicherengpässe**
- **Test der Robustheit**
 - Robustheit bezeichnet die **Unanfälligkeit** einer Software bei Fehleingaben etc.
 - Robustheit wird anhand von **Negativtests** geprüft, der bewussten Eingabe von falschen oder unzulässigen Werten, die vom System abgefangen werden müssen
- **Struktureller Test**
 - Strukturelle Tests wie Anweisungs- und/oder Entscheidungsüberdeckung

Komponententest – Was und wie wird getestet?

- Testobjekte sind programmierte **einzelne Komponenten**
 - Diese Testobjekte sind oftmals **nicht eigenständig ausführbare** Programmteile
- **Treiber, Platzhalter**, Mock-Objekte, Service-Virtualisierung, Rahmen und Simulatoren werden benötigt je nach SW-Entwicklungslebenszyklus-Modell
 - Treiber bildet bzw. nutzt die spezifizierte Schnittstelle zum Testobjekt (Komponente)
 - Testtreiber ermöglichen die Ausführung der Komponente in der Software-Umgebung
 - Testtreiber simulieren **Eingaben**, z. B. vom User
 - Platzhalter nehmen **Ausgabewerte** entgegen
 - Treiber stellt i. d. R. Software-Ablauf-Umgebung zur Verfügung

Komponententest – Aufrufhierarchie

- **Treiber** können selbst programmiert werden, wenn
 - **Programmierkenntnisse** vorhanden sind
 - Programmcode offen liegt
 - Programmierwerkzeuge zur Verfügung stehen
- **Platzhalter** oder **Stubs** werden benötigt, um
 - Noch nicht realisierte Komponenten für die Durchführung von Tests zu ersetzen bzw. zu simulieren

Entwicklertests

- Der Entwickler testet seinen Code
- Er findet und behebt Fehlerzustände
- Wechselt zwischen dem Schreiben der Tests und der Ausführung
- Im agilen Umfeld können die automatisierten Tests schon vor dem Schreiben des Codes erstellt werden (TDD)



Testgetriebene Entwicklung oder Test-First-Ansatz

(TDD – test driven development)

- **Testfälle dominieren Entwicklungszyklen**
 - **Vor Entwicklungsbeginn** werden die **Testfälle** vorbereitet
 - Testfälle werden für die Durchführung **automatisiert** (Tool-Einsatz notwendig!)
- **Entwicklung gegen die Testfälle**
 - Die Komponente wird in einer ersten Version entwickelt
 - Automatisierte Durchführung der Testfälle
 - Behebung der aufgetretenen Abweichungen durch Entwicklung einer Folgeversion
 - Erneute Durchführung der Testfälle, bis keine Abweichungen mehr auftreten
 - Hoch **iterativer** Ansatz
- **Refactoring**
 - Bezeichnet in der Software-Entwicklung die manuelle oder automatisierte Strukturverbesserung von Quelltexten unter Beibehaltung des beobachtbaren Programmverhaltens

Zusammenfassung Komponententest

- **Komponenten** sind die kleinsten spezifizierten Einheiten der Software
- **Modul-, Unit-, Klassen- und Entwicklertest** werden synonym für Komponententest verwendet
- Aufrufende Funktionen werden durch **Treiber** (driver), aufgerufene Funktionen durch **Platzhalter** (stubs) ersetzt, Umgebungen werden durch **Simulatoren** abgebildet
- Der Komponententest kann neben **funktionalen** auch **nicht-funktionale** Qualitätsmerkmale (Robustheit / Speicherengpässe) messen

Integrationstest

1 von 2

Test der Interaktion zwischen Komponenten (und Systemen)

- **Ziele**
 - Risikoreduktion
 - Verifizierung gegen Entwurf und Spezifikation (funktionale und nicht-funktionale Verhaltensweisen) der Schnittstellen
 - Vertrauen in die Qualität der **Schnittstellen** schaffen
 - Fehlerzustände finden
 - Weitergabe von Fehlerzuständen in die höheren Teststufen verhindern
 - Automatisierte Integrationstests (insbesondere bei agiler Entwicklung), liefern Vertrauen, dass bei Änderungen keine bestehenden Komponenten oder Systeme beschädigt wurden



Integrationstest

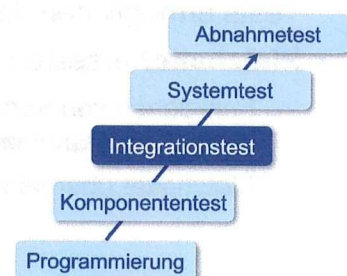
2 von 2

- **Testbasis**

- Software- und Systementwurf
- Sequenzdiagramme
- Spezifikationen von Schnittstellen, Protokollen
- Anwendungsfälle, Workflows
- Architektur auf Komponenten- oder Systemebene
- Externe Schnittstellendefinition

- **Testobjekte**

- Subsysteme
- Datenbanken
- Infrastruktur
- Schnittstellen (interfaces)
- APIs
- Microservices



Begriffe zu Integrationstest

- **Komponenten-Integrationstest** (component integration testing)

Testen wird durchgeführt mit dem Ziel, Fehlerzustände in den Schnittstellen und dem Zusammenwirken der integrierten Komponenten aufzudecken.

- Interaktionen und Schnittstellen zwischen Komponenten
- Durchführung nach erfolgreichem Komponententest, meist automatisiert
- Oft Teil des kontinuierlichen Integrationsprozesses (continuous integration)

- **System-Integrationstest** (system integration testing)

Testen der Verbindung und Interaktion von Systemen

- Interaktionen und Schnittstellen zwischen Systemen, Paketen, Microservices
- Überdeckung von Interaktionen und Schnittstellen zu externen Organisationen
Herausforderung, weil die Schnittstellen in anderen Zuständigkeitsbereichen liegen
- Durchführung nach oder parallel zu dem Systemtest

Integrationstest – Was und wie wird getestet?

- Getestet wird ein **(Teil-) System**, das sich aus Einzelkomponenten zusammensetzt
 - Jede der Komponenten hat eine **Schnittstelle**, die **nach außen** und / oder **zu einer anderen Komponente** des (Teil-) Systems führt
- Es werden **Treiber** (siehe Komponententest) benötigt, die
 - Eingaben und Ausgaben für das (Teil-) System ermöglichen bzw. produzieren
 - Daten protokollieren
- **(Test-)Monitore**, die mit der **Aufzeichnung von Daten** eine Kontrolle der Tests ermöglichen, können sinnvoll eingesetzt werden
- **Platzhalter** ersetzen **fehlende Komponenten**
 - Funktionen von Komponenten, die noch nicht integriert wurden, werden über speziell programmierte Platzhalter eingefügt
 - Platzhalter übernehmen die elementaren Aufgaben fehlender Komponenten

Integrationstest – Typische Fehlerzustände und Fehlerwirkungen

- Falsche oder fehlende Daten, falsche Datenverschlüsselung
- Falsche Reihenfolge, fehlerhafter zeitlicher Ablauf von Schnittstellenaufrufen
- Schnittstellenfehlanpassung
- Fehlerwirkungen in der Kommunikation zwischen Komponenten / Systemen
- Nicht behandelte oder nicht korrekt behobene Fehlerwirkungen in der Kommunikation zwischen Komponenten / Systemen
- Annahmen über Bedeutung, Einheiten oder Grenzen der Daten differieren zwischen den Komponenten / Systemen
- Inkonsistente Nachrichtenstrukturen zwischen den Systemen
- Fehlende Konformität mit erforderlichen Richtlinien zur Informationssicherheit

Integrationstest – Verantwortlichkeiten

- Komponenten-Integrationstest liegen meist beim Entwickler
- System-Integrationstests liegen meist beim Tester
- Tester im System-Integrationstest sollten die Systemarchitektur verstehen und an der Planung der **Integrationsstrategie** beteiligt sein

Integrationsstrategien

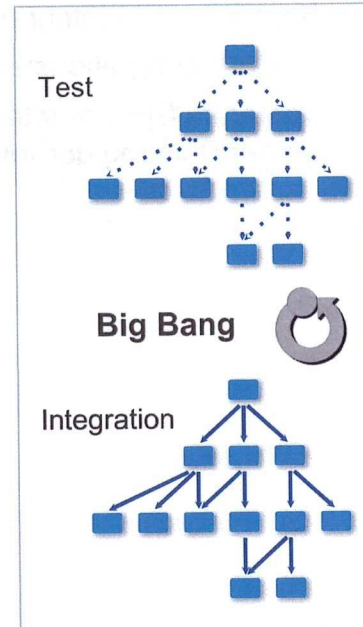
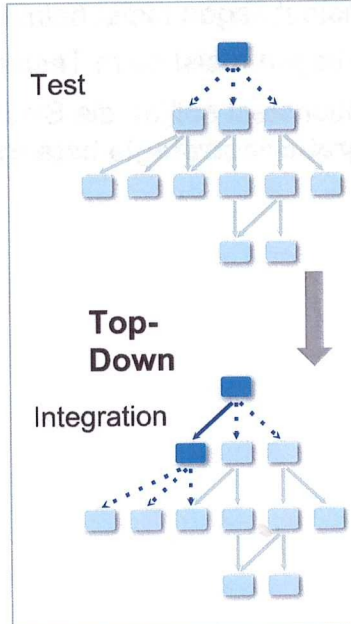
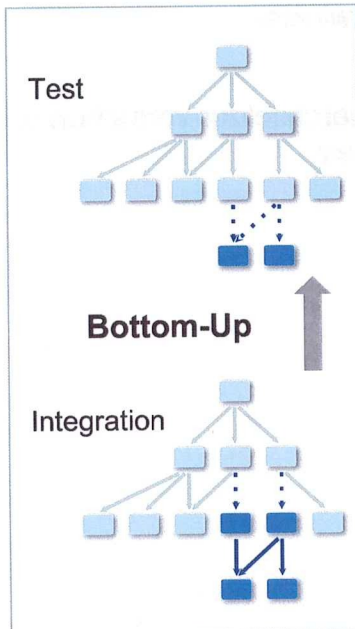
1 von 3

Integrationsstrategien müssen auf die Entwicklung abgestimmt sein

- Planung schon vor der Entwicklung
- Reihenfolge der Entwicklung von Komponenten und Systemen auf effektives Testen abstellen!
- Inkrementeller Ansatz vereinfacht die Isolation von Fehlerzuständen
- Systematische Strategien basieren auf
 - Der Systemarchitektur (**Top-Down, Bottom-Up**)
 - Funktionalen Aufgaben, der Reihenfolge von Transaktionen oder ähnlichen Aspekten (**geschäftsprozessorientiert**)
- **Big Bang** Strategie: Integration in einem Schritt erschwert die Isolation von Fehlerzuständen
- **Kontinuierliche Integration** (continuous integration) auf Komponentenbasis
- Risikoanalyse der komplexesten Schnittstellen unterstützt einen zielgerichteten Einsatz von Integrationstests

Integrationsstrategien

2 von 3



Integrationsstrategien

3 von 3

Kontinuierliche Integration (continuous integration)

- Je kleiner der Umfang der Integration desto einfacher die Isolation von Fehlern
- D.h. Risikoreduktion sowie niedrigerer Zeitaufwand für Fehlerbehebung
- Kontinuierliche Integration sollte auf Komponentenbasis implementiert sein (funktionale Integration)
- Unterstützung durch automatisierte Regressionstests auf mehreren Teststufen
- Einbettung in automatisierte Build-Pipelines

Zusammenfassung Integrationstest

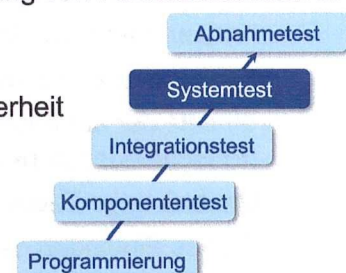
- Das Zusammenfügen von Komponentengruppen wird **Integration** genannt
- Der **Integrationstest** prüft deren Zusammenwirken gegen die Spezifikation der Schnittstellen
- Verbreitete Integrations-Strategien sind **Bottom-Up**, **Top-Down**, **Big Bang**, **geschäftsprozessorientiert** oder **Kontinuierliche Integration**
- Auch der Test der **Integration** bereits integrierter **Teilsysteme** fällt unter den Integrationstest

Systemtest

1 von 3

Prüfung des integrierten **Gesamtsystems** auf Einhaltung der spezifischen **Anforderungen** und auf das Verhalten des Systems, meist durch **unabhängige** Tester nach Vorgaben der Spezifikation – mit folgenden **Zielen**:

- Risikoreduktion
- Vertrauen in die Qualität des **Systems**
- Verifizierung, ob die funktionalen und nicht-funktionalen Verhaltensweisen des Systems mit der Spezifikation übereinstimmen
- Validierung, ob das System vollständig ist und funktioniert wie erwartet
- Finden von Fehlerzuständen und verhindern von Mitführung von Fehlerzuständen in die nächste Stufe
- Verifizierung der Datenqualität
- Einsatz automatisierter Systemregressionstests zur Sicherheit
- Lieferung von Informationen an Stakeholder z.B. für Freigabeentscheidungen
- Nachweis der Erfüllung von Standards, regulatorischen oder rechtlichen Anforderungen



Systemtest

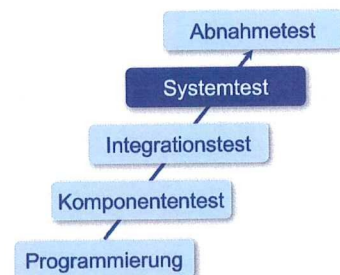
2 von 3

• Testbasis

- System- und Software-Anforderungsspezifikation funktional und nicht-funktional
- Risikoanalyseberichte
- Anwendungsfälle
- Epics und User Stories
- Modelle des Systemverhaltens
- Zustandsdiagramme
- System- und Benutzeranleitungen

• Testobjekte

- Anwendungen
- Hardware / Softwaresysteme
- Betriebssysteme
- Systeme unter Test (SUT)
- Systemkonfiguration
- Konfigurationsdaten



Systemtest

3 von 3

• Typische Fehlerzustände und Fehlerwirkungen

- Falsche Berechnungen
- Falsches / unerwartetes Systemverhalten (funktional und nicht-funktional)
- Innerhalb des Systems – falsche Kontroll- und / oder Datenflüsse
- Versagen bei der korrekten / vollständigen Ausführung der End-to-End-Aufgaben
- In der Produktivumgebung – ordnungsgemäße Nutzung des Systems versagt
- System- und Benutzeranleitungen – System funktioniert nicht wie beschrieben

• Spezifische Ansätze und Verantwortlichkeiten

- Konzentration auf das allgemeine End-to-End-Verhalten des ganzen Systems
- Funktionale und nicht-funktionale Aspekte stehen im Vordergrund
- Einsatz der am besten geeigneten Testverfahren zum zu testenden Aspekt (Entscheidungstabelle für funktionales Verhalten der Geschäftsregeln)
- Unabhängige Tester zur Vermeidung „falsch-positiver“ / „falsch-negativer“ Ergebnisse
- Frühes Einbinden der Tester in statische Testaktivitäten zur Vorbeugung

Systemtest – Softwarequalität

- Der Systemtest unterscheidet die Anforderungen entsprechend der **Softwarequalität** (ISO 25010) in
- Funktionale Anforderungen
 - Funktionalität
- Nicht-funktionale Anforderungen
 - Zuverlässigkeit
 - Gebrauchstauglichkeit
 - Effizienz
 - Wartbarkeit
 - Übertragbarkeit
 - (IT-)Sicherheit
 - Kompatibilität

Systemtest – Funktionale Anforderungen

Es werden drei Ansätze zum Test funktionaler Anforderungen unterschieden

- **Geschäftsprozessbasierter Test**
 - Einzelne Geschäftsprozesse dienen als Grundlage für die Ableitung von Testfällen
 - Eine Rangfolge der Prozesse kann in die Priorisierung der Testfälle übernommen werden
- **Anwendungsfallbasierter Test**
 - Die Testfälle werden aus Sequenzen erwarteter bzw. sinnvoller Nutzung abgeleitet
 - Werden bestimmte Anwendungsfälle häufiger ausgeführt, erhalten sie im Test höhere Priorität
- **Anforderungsbasierter Test (Bausteine)**
 - Die Testfälle werden aus der Anforderungsdefinition abgeleitet
 - Je nach Anforderung variiert die Anzahl der Testfälle

Systemtest – Nicht-funktionale Anforderungen

Die **Einhaltung** dieser Anforderungen (Zuverlässigkeit, Benutzbarkeit, Effizienz, Wartbarkeit, Übertragbarkeit, Sicherheit, Kompatibilität) ist genauso wichtig, aber **schwerer zu testen**, da diese

- a) oftmals gar nicht oder **unvollständig spezifiziert** sind
- b) **schlecht zu quantifizieren** sind

Der Anforderungsdefinition ist oft nicht klar zu entnehmen „wie gut“ etwas funktionieren soll

- Häufig **vage Definitionen**: z. B. „problemlos zu bedienen“, „übersichtliche Oberfläche“, etc., werden vorausgesetzt daher nicht definiert

Eine **Quantifizierung** bei nicht-funktionalen Anforderungen mit Hilfe von Metriken ist teilweise **problematisch**, z. B. wegen der **Subjektivität** bei der Bewertung von Kriterien durch den Menschen

Zusammenfassung Systemtest

- Gesamtsystem wird mit **funktionalen** und **nicht-funktionalen** Tests geprüft
- Der funktionale Systemtest hat zum Ziel, die Konformität des entwickelten Systems mit dem **funktionalen Systementwurf** nachzuweisen (Validierung)
- Der nicht-funktionale Systemtest prüft **Zuverlässigkeit, Gebrauchstauglichkeit, Effizienz, Wartbarkeit, Übertragbarkeit, (IT-)Sicherheit und Kompatibilität**
- Nicht-funktionale Qualitätsmerkmale sind oft **implizit** in den Anforderungen enthalten. Das macht eine Validierung schwierig
- Hierfür werden oft unabhängige Testteams eingesetzt

Abnahmetest

1 von 3

Test des Verhaltens und der Fähigkeiten des Systems bzw. Produkts

- **Ziele**

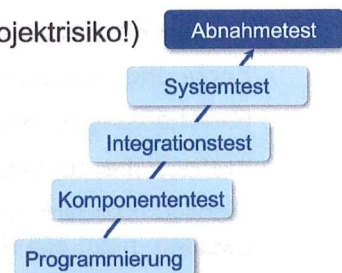
- Vertrauen des Kunden in die Qualität des Systems als Ganzes
- Validieren, ob das System vollständig ist und wie erwartet funktioniert
- Funktionale und nicht-funktionale Verhaltensweisen gemäß Spezifikation verifizieren

- **Weitere Ziele des Abnahmetests**

- Informationen sammeln, ob das System geeignet und bereit für den Einsatz durch den Anwender (User) ist
- Fehlerzustände finden (nicht Ziel des Abnahmetestes! Projektrisiko!)
- Nachweis der Erfüllung von Standards, rechtlichen oder regulatorischen Anforderungen

- **Ausprägungen des Abnahmetests**

- Benutzer-Abnahmetest, Betrieblicher Abnahmetest
- Vertraglicher Abnahmetest, Alpha- und Beta-Test



Abnahmetest

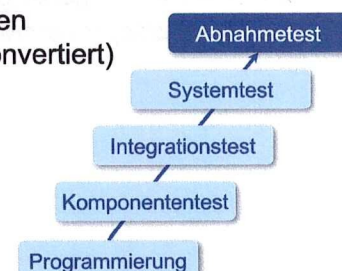
2 von 3

- **Testbasis**

- Benutzer- / Fachanforderungen
- Systemanforderungen
- Anwendungsfälle / Use Cases
- User Stories
- Geschäftsprozesse
- Risikoanalyseberichte
- System- / Benutzerdokumentation
- Installationsverfahren
- Vorschriften, Standards, Verträge

- **Testobjekte**

- System unter Test (SUT)
- Geschäftsprozesse im vollintegrierten System
- Betriebs- und Wartungsprozesse
- Systemkonfigurationen
- Konfigurationsdaten
- Formulare und Auswertungen / Berichte
- Produktionsdaten (bestehend / konvertiert)



Abnahmetest

3 von 3

- **(zusätzliche) Testbasis für betrieblichen Abnahmetest**
 - Sicherungs- und Wiederherstellungserfahren
 - Disaster-Recovery-Verfahren
 - Nicht-funktionale Anforderungen, Performance
 - Betriebsdokumentation
 - Anweisungen zur Bereitstellung und Installation
 - Datenbankpakete
 - Informationssicherheit, Vorschriften, Standards, Daten- und Zugriffssicherheit
- **Typische Fehlerzustände**
 - Fach- und Benutzeranforderungen an Systemworkflow nicht erfüllt
 - Geschäftsregeln werden nicht korrekt eingehalten
 - Vertragliche, regulatorische Anforderungen werden nicht erfüllt
 - Nicht-funktionale Fehlerwirkungen (Sicherheitsschwachstellen, Performance, Lasttest, nicht alle Plattformen unterstützt) vorhanden



Begriffe zum Abnahmetest

1 von 2

- **Alpha-Test** (alpha testing)
Testen in einer Simulations- oder Nutzungsumgebung **beim Hersteller**, welches durch Rollen außerhalb der Herstellerorganisation durchgeführt wird.
- **Beta-Test** (beta testing)
Testen in einer (Hersteller-) **externen** Simulations- oder Nutzungsumgebung, welches durch Rollen **außerhalb der Herstellerorganisation** durchgeführt wird.
- **Benutzer-Abnahmetest** (user acceptance testing)
Abnahmetest, der durch vorgesehene Benutzer in einer echten oder simulierten betrieblichen Umgebung durchgeführt wird, mit dem Fokus ihren Bedarf, Anforderungen und Geschäftsprozessen zu prüfen.



Begriffe zum Abnahmetest

2 von 2

- **Betrieblicher Abnahmetest** (operational acceptance testing)
Ein Betriebstest innerhalb des Abnahmetests, üblicherweise in einer (simulierten) Produktionsumgebung durch den Betreiber und / oder Administrator durchgeführt, mit Schwerpunkt bei den operationalen Aspekten, z.B. Wiederherstellbarkeit, Ressourcenverwendung, Installierbarkeit und technische Kompatibilität
- **Regulatorischer Abnahmetest** (regulatory testing)
Konformitätstest (compliance testing)
Abnahmetest mit dem Ziel, zu verifizieren, ob ein System zu relevanten Gesetzen, Richtlinien und Vorschriften konform ist.

Spezifische Ansätze und Verantwortlichkeiten

Der Abnahmetest erfolgt **nicht unbedingt als letzte Teststufe**

- Bei Standardsoftware kann der Abnahmetest zum Zeitpunkt der Installation oder Integration stattfinden
- Der Abnahmetest einer neuen funktionalen Erweiterung kann **vor** dem **Systemtest** erfolgen

Iterative Entwicklung kennen **verschiedene Formen** der Abnahmetests

- **Während oder am Ende** einer Iteration oder einer Serie von Iterationen
- Verifizierung eines neuen Features in Bezug auf seine Abnahmekriterien
- Validierung, ob ein neues Feature die Bedürfnisse der Benutzer erfüllt
- Alpha- und Beta-Tests am Ende / Abschluss jeder Iteration oder einer Serie davon
- Durchführung von Benutzerabnahmetests, betriebliche Abnahmetests, regulatorische und vertragliche Abnahmetests
 - zum / nach Abschluss einer Iteration oder
 - nach einer Reihe von Iterationen

Die Verantwortung liegt häufig beim Kunden, Fachanwender, Product Owner

Regulatorischer oder vertraglicher Abnahmetest

Hauptziel: Vertrauen aufbauen! Auflagen / Verträge werden erfüllt!

- **Vertraglicher Abnahmetest**

- Kundenindividuelle Software muss vom Auftraggeber abgenommen werden
- Nachprüfbare **Abnahmekriterien** sollten bereits bei **Vertragsschluss** definiert werden – zur Absicherung beider Vertragsparteien
- Durchführung von unabhängigen Testern oder den Benutzern (Usern)

- **Regulatorischer Abnahmetest**

- Einhaltung von **gesetzlichen**, staatlichen oder anderen verbindlichen **Vorschriften zur funktionalen Sicherheit** (z.B. bei Banken: Auflagen der Aufsicht zur Funktionentrennung)
- Durchführung von unabhängigen Testern oder den Benutzern
- Oft auch Abnahme durch Aufsichtsbehörden, Auditoren

Benutzer-Abnahmetest (user acceptance testing (UAT))

Wird die Software von den späteren Anwendern / Nutzern akzeptiert?

- Wird auch **möglichst während der Entwicklung** schon geprüft (z.B. durch Prototypen)

Getestet wird, ob das System

- die Erwartungen **verschiedener Benutzer** bzw. **Benutzergruppen** erfüllt
- einen positiven Eindruck beim Anwender hinterlässt
- Mängel aufweist, die die Benutzerakzeptanz reduzieren
- für den Gebrauch durch den Anwender / Kunden tauglich ist

Betrieblicher Abnahmetest (operational acceptance testing (OAT))

Abnahme durch **Systemadministratoren**, Mitarbeitern des späteren Produktionsbetriebs

Durchführung in einer **simulierten** Produktivumgebung

- Einbindung in vorhandene **IT-Infrastrukturen** (Backup- / Restore-Systeme, Lauffähigkeit auf Servern, Restart-, Installations-, De-Installations-Fähigkeit)
- **Benutzerverwaltung**: Einbindung in vorhandene Verzeichnis-Strukturen oder Verwaltung einer eigenen Benutzer-Liste
- Umsetzung Benutzerrechte-Konzept
- Identifizieren von **Sicherheitsschwachstellen**
- Prüfung von Datenlade- und Migrationsaufgaben
- Möglichkeit der Durchführung von **Wartungsaufgaben** (z. B. Datenbank-Reorganisationen)
- Notfallwiederherstellung (disaster recovery), wo setzt das System wieder auf
- Performancetest, Lasttest, ggf. Stresstest

Alpha- und Beta-Test

- Meist genutzt von Entwicklern von Standardsoftware und COTS* Software
- Hersteller erhält Rückmeldung von Fehlerwirkungen, Bewertung der Benutzbarkeit etc. **vor der Markteinführung**
- Testdurchführung erfolgt durch den potentiellen Kunden, Betreiber oder von unabhängigen Testteams
 - **Alpha-Test**: Test in der Umgebung des Herstellers / Entwicklers
 - **Beta-Test**: Test in und an deren eigenen Standorten
- Vorteile des Alpha- bzw. Beta-Tests
 - Vertrauen schaffen, das System ist unter normalen Umständen gut nutzbar und bringt dem Anwender einen Nutzen
 - Finden von Fehlerzuständen, die sich auf bestimmte Bedingungen, Umgebungen, Zielsysteme beziehen, die im Test nur schwer nachzubauen sind

* COTS = commercial off the shelf

Zusammenfassung Abnahmetest

- Der Abnahmetest ist der Systemtest durch den **Kunden**
- Der **regulatorische** und **vertragliche** Abnahmetest stellt die **vertragliche Erfüllung** sicher, die Software wird auf Erfüllung der Kundenanforderungen geprüft
- Der **Benutzer-Abnahmetest** prüft die **Tauglichkeit eines Systems** durch Benutzer bzw. Kunden. Es werden **unterschiedliche Benutzergruppen** berücksichtigt
- Der **betriebliche Abnahmetest** wird vom Systemadministrator durchgeführt (Benutzerverwaltung)
- **Alpha-Tests** finden beim Hersteller der SW durch (potenzielle) Kunden / Nutzer in separaten Umgebungen statt
- **Beta-Tests** sind **Feldtests**, in denen Kunden Vorabversionen der Software in ihrer eigenen Umgebung testen

Kapitel II – Testen im Softwarelebenszyklus

- II/1 Softwareentwicklungslebenszyklusmodelle
- II/2 Teststufen
- **II/3 Testarten**
- II/4 Wartungstests



Begriffe zu Testarten

- **Testart** (test type)

Eine Gruppe von Testaktivitäten basierend auf **bestimmten Testzielen** mit der Absicht, eine Komponente oder ein System auf einige zusammenhängende Qualitätsmerkmale zu prüfen. [nach TMap]

- **Testziel** (test objective)

Der Grund oder Zweck des Testens.

- Bewertung **funktionaler Qualitätsmerkmale**
(Vollständigkeit, Korrektheit, Angemessenheit)
- Bewertung **nicht-funktionaler Qualitätsmerkmale**
(Zuverlässigkeit, Performance, Informationssicherheit, Kompatibilität, Usability)
- Bewertung der **Struktur oder Architektur** auf Korrektheit und Vollständigkeit
(Komponente, System)
- Bewertung der **Auswirkung von Änderungen**
(Behebung von Fehlerzuständen, Suche nach unerwünschten Nebenwirkungen (Regressionstests))



Begriffe zu Testarten

- **Fehlernachtest** (confirmation testing)

Dynamisches Testen nach einer Fehlerkorrektur zum Zweck der Bestätigung, dass Fehlerwirkungen nicht mehr auftreten, nachdem die dafür ursächlichen Fehlerzustände korrigiert wurden.

- **Regressionstest** (regression testing)

Testen einer bereits getesteten Komponente oder eines Systems nach einer Modifikation, um sicherzustellen, dass in nicht geänderten Bereichen durch die vorgenommenen Änderungen keine Fehlerzustände eingebaut oder bisher maskierte Fehlerzustände freigelegt wurden.

- **White-Box Test** (white-box testing)

Ein Test, der auf der Analyse der internen Struktur einer Komponente oder eines Systems basiert.

Funktionale Tests (was soll das System tun?)

Die **Funktionalität** bewertet die Funktionen, die das System ausführen soll

- Getestet wird gegen die **funktionalen Anforderungen** (aus Fachanforderungen, Spezifikationen, Epics, User-Stories, Anwendungsfällen, funktionalen Spezifikationen oder ähnlichen Dokumenten)
- Das Verhalten der Software kann mit **Black-Box-Verfahren** getestet werden: aus der Funktionalität der Komponente / des Systems werden Testbedingungen und Testfälle abgeleitet

Einsatz auf **allen Teststufen** jedoch mit unterschiedlichem Fokus / Intensität

- Z.B. Komponententests gegen Komponentenspezifikation

Funktionale Überdeckung (Prozentsatz der Testarten, die abgedeckt sind)

- Einsatz der Verfolgbarkeit zwischen Test und funktionalen Anforderungen
- Überdeckungslücken können identifiziert werden

Spezielle Fähigkeiten und **Fachwissen** bei Testentwurf und Tests erforderlich

- Z.B. bei Computerspielen, Tests von Fahrerassistenzsystemen, geologischer Modellierungssoftware für die Gasindustrie

Nicht-funktionale Tests

1 von 4

Merkmale von Systemen und Software werden getestet

- Wie gut erfüllt das Testobjekt seinen Zweck?
- Produktmerkmale nach ISO 25010: Zuverlässigkeit, Benutzbarkeit, Effizienz, Änderbarkeit, Übertragbarkeit, IT-Sicherheit, Kompatibilität

Einsatz allen Teststufen, unterschiedlicher Intensität, so **früh wie möglich**

- Spätes Entdecken von Fehlerzuständen kann den Projekterfolg extrem gefährden
- Black-Box-Verfahren können herangezogen werden, um Testbedingungen und Testfälle abzuleiten (z.B. Grenzwertanalyse für Stresstest in der Performance)

Nicht-funktionale Überdeckung (Prozentsatz der abgedeckten Testarten)

- Nutzung der Verfolgbarkeit zwischen Tests und z.B. unterstützten Endgeräten bei mobilen Applikationen, Kompatibilitätstests
- Aufdecken von potentiellen Überdeckungslücken

Fachwissen erforderlich (z.B. Sicherheitsschwachstellen, Benutzbarkeit)

- Erkennen von Schwachstellen im Entwurf (Technologie, Software, Benutzergruppe)



Nicht-funktionale Tests – Beispiele

2 von 4

- **Lasttest** (load testing)
Eine Art des Performanztests, die das Verhalten eines Systems oder einer Komponente unter wechselnder Last bewertet, üblicherweise zwischen zu erwartender niedriger, typischer sowie Spitzenlast.
- **Stresstest** (stress testing)
Spezifische Form des Performanztests, die durchgeführt wird, um ein System oder eine Komponente an oder über den Grenzen, die in den Anforderungen spezifiziert wurden, zu bewerten.
- **Performanztest** (performance testing)
Testen zur Bestimmung der Performanz eines Softwareprodukts.



Nicht-funktionale Tests – Beispiele

3 von 4

- **Gebrauchstauglichkeitstest** (usability testing)
Testen mit dem Ziel herauszufinden inwieweit das System durch spezifizierte Benutzer in einem bestimmten Kontext mit Effektivität, Effizienz und Zufriedenheit genutzt werden kann.
- **Zuverlässigkeitstest** (reliability testing)
Testen, um die Zuverlässigkeit eines Softwareprodukts zu bestimmen.
- **Übertragbarkeitstest** (portability testing)
Testen zur Bestimmung der Übertragbarkeit eines Softwareprodukts.
- **Wartbarkeitstest** (maintainability testing)
Testen, um die Änderbarkeit eines Softwareprodukts zu bestimmen.
- **Robustheitstest** (robustness testing)
 - (1) Test zum Ermitteln der Robustheit eines Softwareprodukts
 - (2) siehe Negativtest



Nicht-funktionale Tests – Beispiele

4 von 4

- **IT-Sicherheitstest** (security testing)

Die Durchführung von Tests, um die Sicherheit (im Sinne von Zugriffsschutz) eines Softwareprodukts zu bestimmen.

- **Interoperabilitätstest / Test der Kompatibilität** (interoperability testing)

Testen zur Bestimmung der Interoperabilität eines Softwareprodukts. Wie zwei oder mehr Komponenten oder Systeme Informationen austauschen und diese nutzen können.

White-Box Test

- **Strukturelle Überdeckung**

- Analysiert wird **die interne Struktur** des Testobjekts (Code, Architektur, Workflows, Datenflüsse)
- Gemessen wird der Prozentgrad der Überdeckung, wie vollständig die strukturellen Elemente durch Tests ausgeführt wurden

- **Teststufe: Komponententest**

- Codeüberdeckung, Überdeckungsmessung, wie viele Anweisungen / Entscheidungen ausgeführt wurden in Relation zur Gesamtanzahl (mittels Tools)

- **Teststufe: Komponentenintegrationstest**

- Überdeckungsmessung auf Basis der Architektur des Systems (interne Schnittstellen)
- aber auch **im System- und Abnahmetest** möglich (Menüstrukturen, Geschäftsprozesse)

- **Spezielle Fähigkeiten und Fachwissen** über White-Box-Testing / -Tools

- Z.B. wie ist der Code aufgebaut; wie werden die Daten gespeichert für Datenbankabfragen; korrekte Interpretation der Ergebnismessungen der Tools

Änderungsbezogenes Testen

1 von 3

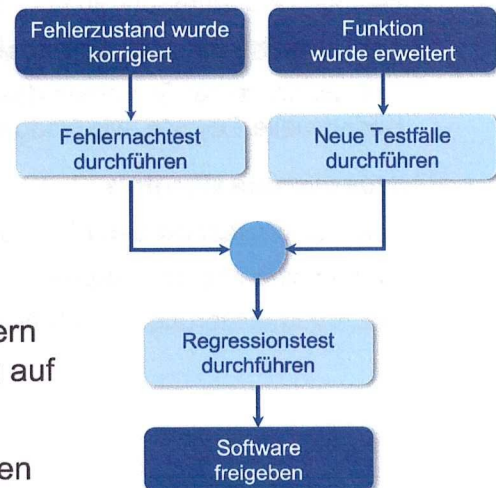
Veränderungen am Testobjekt können neue Fehlerwirkungen bringen

Erneute Tests stellen sicher, dass

- **Fehlerzustände** erfolgreich korrigiert wurden (alte Testfälle, Fehlernachtest)
- **neue** Funktionen korrekt implementiert sind (neue Testfälle)
- keine ungewünschte Beeinflussung der **Altfunktionalität** stattgefunden hat (alte Testfälle, Regressionstest)

Modularität und gute Dokumentation erleichtern Bewertung, welchen **Einfluss** eine Änderung auf das Altsystem hat

Einsatzgebiet in **allen Teststufen** und mit allen Testarten



Änderungsbezogenes Testen

2 von 3

- **Fehlernachtest** (Korrektur eines Fehlerzustandes)
 - Durchführung aller Testfälle, die aufgrund des Fehlerzustandes fehlschlagen
 - Durchführung aller Schritte (Reproduktion), die zur Fehlerwirkung geführt haben
- **Test der neuen Features** (bzw. Änderungen)
 - Neue Testfälle bei ergänzten Funktionalitäten
- **Regressionstest** (Auffindung unbeabsichtigter Nebeneffekte)
 - Änderungen im Code können versehentlich das Verhalten anderer Teile des Codes beeinflussen
 - Jede Art von Änderung kann unerwünschte Fehlerwirkungen erzeugen (innerhalb der Komponente, in anderen Komponenten, anderen Systemen)
 - Änderungen der Umgebung wie Versionswechsel des Betriebssystems oder des Datenbankmanagementsystems können Fehlerwirkungen erzeugen
 - Die unerwünschten Nebeneffekte nennt man **Regression**
 - **Regressionstests sollen diese Nebeneffekte aufzeigen**

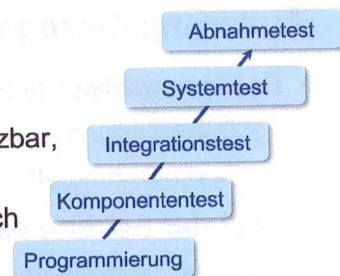
Änderungsbezogenes Testen

3 von 3

- **Iterative und inkrementelle** Entwicklungslebenszyklen (agile)
 - Neue Features, Änderungen in Features, Code-Restrukturierung (refactoring)
 - Häufige Codeänderungen erfordern änderungsbezogene Tests
 - **Fehlernachtests und Regressionstests** sind elementar **wichtig**
 - Insbesondere bei Systemen des Internets der Dinge (Individuelle Objekte, wie Endgeräte, werden aktualisiert oder ersetzt)
- **Regressionstestsuiten**
 - Regressionstest bleiben oft stabil, unterliegen wenigen Änderungen
 - Automatisierung der Regressionstests ist daher sinnvoll
 - Die Automatisierung sollte früh im Testprozess beginnen (s. Kap. 6)

Testarten und Teststufen

- **Teststufen**
 - Jede der genannten Testarten ist in jeder Teststufe einsetzbar, aber mit unterschiedlichem Fokus und Intensität
 - In jeder Teststufe sind die Testbedingungen unterschiedlich
- **Testarten**
 - **Funktionale Tests** (Nachweis der Funktionalität)
 - **Nicht-funktionale Tests** (Nachweis der Produktmerkmale)
 - **Strukturbezogener Test** (Überdeckung erreichen)
 - **Änderungsbezogenes Testen** (Nachweis korrekter Fehlerbehebung)



Im folgenden sind Beispiele für die Anwendung der Testarten auf die Teststufen auf Basis einer Banking-Anwendung skizziert



Testarten und Teststufen – Funktionale Tests

- Komponententest
 - Grundlage der Tests: Berechnung von Zinseszinsen
- Komponenten-Integrationstest
 - Grundlage der Tests: Übertragung der Erfassung von Kontoinformationen auf der Benutzeroberfläche in die fachliche Logik
- Systemtest
 - Grundlage der Tests: Kontoinhaber beantragt eine Kreditlinie für sein Konto
- System-Integrationstest
 - Grundlage der Tests: Nutzung eines Microservices zur Bonitätsprüfung des Kontoinhabers
- Abnahmetest
 - Grundlage der Tests: Bankmitarbeiter entscheidet über Annahme oder Ablehnung der Kreditanfrage



Testarten und Teststufen – Nicht-funktionale Tests

- Komponententest
 - Bewertung der Anzahl von CPU-Zyklen zur Berechnung der Gesamtverzinsung
- Komponentenintegrationstest
 - Informationssicherheitstests für Speicherüberlauf-Schwachstellen für Daten, die von der GUI an die fachliche Logik übertragen werden
- Systemtest
 - Übertragbarkeitstests zur Prüfung, ob die Präsentationsschicht auf allen unterstützten Browsern und Endgeräten funktioniert
- Systemintegrationstest
 - Zuverlässigkeitstests: Robustheit des Systems bei Nichtantwort des Microservices
- Abnahmetest
 - Gebrauchstauglichkeit: Barrierefreiheit der Kreditbearbeitungsoberfläche für Menschen mit Behinderungen



Testarten und Teststufen – Strukturbezogene Tests

- Komponententest
 - 100% Überdeckung für Anweisungen und Entscheidungen der Berechnungskomponente
- Komponenten-Integrationstest
 - Übertragung der Daten von einer GUI an die nächste und an die Fachlogik
- Systemtest
 - Überdeckung aller Websites, die während der Kreditanfrage angezeigt werden
- System-Integrationstest
 - Ausführung aller möglichen Anfragearten an den Bonitäts-Microservice
- Abnahmetest
 - Überdeckung aller unterstützten Dateistrukturen der Finanzdaten und Wertebereiche für Zahlungsverkehr zwischen Banken



Testarten und Teststufen – Änderungsbezogenes Testen

- Komponententest
 - Automatisierte Regressionstests für jede Komponente des Continuous-Integration Frameworks
- Komponenten-Integrationstest
 - Prüfung nach dem Einchecken im Code-Repository, ob die schnittstellenbasierten Fehlerzustände behoben wurden
- Systemtest
 - Nach einer GUI-Änderung, Wiederholung aller Tests zu diesem Workflow
- System-Integrationstest
 - Wiederholung der Interaktion mit dem Bonitäts-Microservice nach als Teil der kontinuierlichen Verteilung (continuous deployment)
- Abnahmetest
 - Wiederholung aller fehlgeschlagenen Tests des Abnahmetests nach Fehlerbehebung

Zusammenfassung

- In den unterschiedlichen **Teststufen** können unterschiedliche **Testarten** eingesetzt werden
- Bei Testarten wird unterschieden: **funktionale, nicht-funktionale, White-Box-Tests und änderungsbezogene Tests**
- Funktionale Tests testen das **Ein- / Ausgabeverhalten** des Produktes
- Nicht-funktionale Tests prüfen die **Softwaremerkmale**
- Wichtige nicht-funktionale Tests sind **Lasttest, Performancetest, Stresstest, Robustheitstest**
- **White-Box-Tests** leiten sich aus der **internen Struktur oder der Umsetzung** des Systems ab und messen Überdeckungen
- Die wichtigsten änderungsbezogenen Tests sind der **Fehlernachtest** und der **Regressionstest**

Kapitel II – Testen im Softwarelebenszyklus

- II/1 Softwareentwicklungslebenszyklusmodelle
- II/2 Teststufen
- II/3 Testarten
- II/4 Wartungstests



Testen nach der Produktabnahme – Wartungstest

- **Wartungstest** (maintenance testing)

Testen der Änderungen an einem laufenden System oder der Auswirkungen einer geänderten Umgebung auf ein laufendes System.

Der Kunde hat das **Produkt abgenommen** und es in Produktion gebracht

Die **Software** selbst steht aber erst am **Anfang** ihres Lebens

- Sie wird oft über viele Jahre eingesetzt und auch weiterentwickelt
- Sie ist sicher nicht fehlerfrei und wird deshalb weiter bearbeitet
- Sie muss an neue Bedingungen angepasst oder in neue Umgebungen integriert werden

Wartungs-Release

- Wartungstests erstrecken sich über **mehrere Teststufen** und mehrere **Testarten**
- **Risikohöhe** der Änderung (Grad der Abhängigkeit zu anderen Bereichen)
- Größe des Systems und **Umfang** der Änderungen

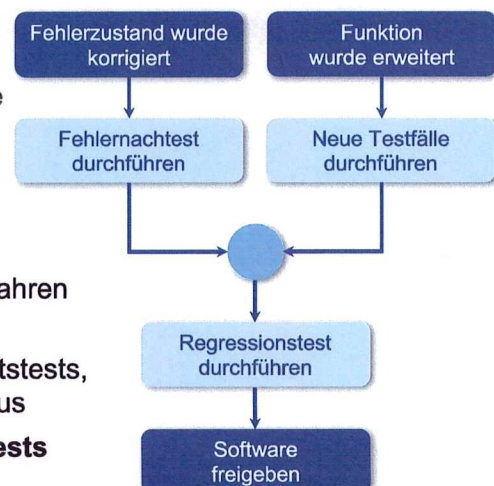
Auslöser für Wartung

Softwarewartung unterscheidet drei Arten von Tätigkeiten

- **Modifikation:** Beseitigung von Fehlerzuständen, geplante Verbesserungen, Aktualisierung von kommerzielle Standardsoftware
- **Migration:** Test in neuer Umgebung, Datenkonvertierungen, Datenübernahme
- **Außerbetriebnahme:** Ablösung der Software

Testumfang der Wartungstests

- Fehlerkorrektur: **Fehlernachtests**
- Weiterentwicklung: **neue Testfälle**
- **Datenarchivierung** / Wiederherstellungsverfahren
- Bei „Internet der Dinge“-Systemen stehen Integrationstests (Netzwerke, etc.), Sicherheitstests, Hardwaregeräte und Softwaredienste im Fokus
- **Zusätzlich** immer ausgiebige **Regressionstests**





Auswirkungsanalyse für Wartung

- **Auswirkungsanalyse** (impact analysis)

Die **Untersuchung** und Darstellung der Auswirkungen einer **Änderung** von spezifizierten Anforderungen auf die Entwicklungsdokumente, auf die Testdokumentation und auf die **Komponenten**

- Bewertung möglicher **Nebeneffekte** auf andere Bereiche des Systems
- Auswirkung der Änderung auf **bestehende** Tests
- Umfang der **Regressionstests** der Systeme, die von der Änderung betroffen sind
- Bewertung potentieller Folgen für **andere Bereiche** des Systems
- **Risiken**:
 - Spezifikationen **veraltet** oder nicht vorhanden
 - Testfälle nicht dokumentiert oder **veraltet**
 - **Bidirektionale Verfolgbarkeit** zwischen Tests und Testbasis **nicht** gegeben
 - Werkzeugunterstützung kaum oder nicht existent
 - **Fach- und Systemkenntnisse** nicht mehr vorhanden bei den Stakeholdern
 - Wartbarkeit und Dokumentation bei **Systemerstellung** vernachlässigt

Zusammenfassung

- Fertiggestellte **Software ändert sich** bzw. muss angepasst werden
- Auswirkungen dieser Änderungen werden durch eine **Auswirkungsanalyse** erfasst (Umfang der Regressionstests)
- Eine vorausschauende Release-Planung ist entscheidend
- Durch erneute Tests ist sicherzustellen, dass
 - **neue Funktionen** korrekt implementiert sind (durch neue Testfälle)
 - **Fehlerzustände** erfolgreich **korrigiert** wurden (durch Testfälle die vorher die Fehlerwirkungen erzeugt haben)
 - keine ungewünschte Beeinflussung der **Altfunktionalität** stattgefunden hat (durch alte Testfälle, Regressionstest)
- **Migrationstests** sind Wartungstests
- Wird die **Software eingezogen** (aus dem Betrieb genommen), sind ggf. Tests der Datenmigration, Systemintegration und Archivierung notwendig

Schlüsselbegriffe

- Abnahmetest
- Änderungsbezogenes Testen
- Alpha-Test
- Auswirkungsanalyse
- Benutzerabnahmetest
- Beta-Test
- Betrieblicher Abnahmetest
- Fehlernachtest
- Funktionaler Test
- Integrationstest
- Kommerzielle Standardsoftware
- Komponentenintegrationstest
- Komponententest
- Nicht-funktionaler Test
- Regressionstest
- Regulatorischer Abnahmetest
- Sequenzielles Entwicklungsmodell
- Systemintegrationstest
- Systemtest
- Testart
- Testbasis
- Testfall
- Testobjekt
- Teststufe
- Testumgebung
- Testziel
- Vertraglicher Abnahmetest
- Wartungstest
- White-Box-Test

Kapitel III – Statischer Test

- III/1 Grundlagen des statischen Tests
- III/2 Reviewprozess

Lernziele für den statischen Test

1 von 2

3.1 Grundlagen des statischen Tests

- FL-3.1.1 (K1) Arten von Softwarearbeitsergebnissen erkennen können, die durch die verschiedenen statischen Testverfahren geprüft werden können
- FL-3.1.2 (K2) Beispiele nennen können, um den Wert des statischen Tests zu beschreiben
- FL-3.1.3 (K2) Den Unterschied zwischen statischen und dynamischen Verfahren unter Berücksichtigung der Ziele, der zu identifizierenden Fehlerzustände und der Rolle dieser Verfahren innerhalb des Softwarelebenszyklus erklären können

Lernziele für den statischen Test

2 von 2

3.2 Reviewprozess

- FL-3.2.1 (K2) Die Aktivitäten des Reviewprozesses für Arbeitsergebnisse zusammenfassen können
- FL-3.2.2 (K1) Die unterschiedlichen Rollen und Verantwortlichkeiten in einem formalen Review erkennen können
- FL-3.2.3 (K2) Die Unterschiede zwischen den unterschiedlichen Reviewarten erklären können: informelles Review, Walkthrough, technisches Review und Inspektion
- FL-3.2.4 (K3) Ein Reviewverfahren auf ein Arbeitsergebnis anwenden können, um Fehlerzustände zu finden
- FL-3.2.5 (K2) Die Faktoren erklären können, die zu einem erfolgreichen Review beitragen

Kapitel III – Statischer Test

- III/1 Grundlagen des statischen Tests
- III/2 Reviewprozess

Statische Prüftechniken resp. Testverfahren

- Statische Tests umfassen
 - **Reviews** (manuelle Prüfung) von Arbeitsergebnissen
 - **Statische Analysen** (werkzeuggestützt) des Codes oder anderer Arbeitsergebnisse
- Jedes Arbeitsergebnis der Softwareentwicklung kann einem Review unterzogen werden, auch Code
- Statische Analyse ist eine werkzeuggestützte Bewertung des Prüfobjekts
 - Sicherheitskritische Systeme (Luftfahrt, Medizintechnik, Nuklearkraftwerk)
 - IT-Sicherheitskritische Systeme
 - Eingebettet in automatisierte Build & Delivery-Systeme im agilen Umfeld Teil der continuous delivery / deployment Auslieferung
- Vor einem **Codereview** kann eine statische Analyse (Kontrollflussanalyse, Datenflussanalyse) hilfreich sein
- Statische Tests adressieren **Fehlerzustände**, nicht Fehlerwirkungen
- Der Code oder das Arbeitsergebnis werden **nicht** ausgeführt



Begriffe

- **Statischer Test** (static testing)
Testen von Software-Entwicklungsartefakten, z.B. Anforderungen oder Quelltext, ohne diese auszuführen. Z.B. durch Reviews oder statische Analyse
- **Review** (review)
Eine Art des statischen Tests, während der ein Arbeitsergebnis oder ein Prozess von einer oder mehreren Personen beurteilt werden, um Befunde zu erheben und um Verbesserungspotentiale zu identifizieren. [IEEE 1028]
- **Statische Analyse** (static analysis)
Der Prozess der Bewertung eines Testobjekts (Komponente oder System) basierend auf seiner Form, seiner Struktur, seines Inhalts oder seiner Dokumentation, ohne es auszuführen. [ISO 24765]
- **Dynamischer Test** (dynamic testing)
Prüfung des Testobjekts durch Ausführung auf einem Rechner

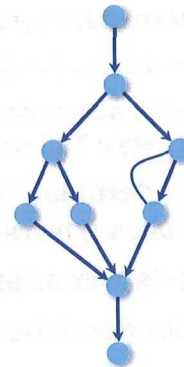
Review vs. Statische Analyse

Bei einem Review analysiert der Gutachter bzw. Reviewer (Tester) das Prüfobjekt manuell

- Z. B. Spezifikationen oder Code
- Gutachter muss analytische Arbeit liegen und das Prüfobjekt inhaltlich (fachlich, technisch) beurteilen können

Die statische Analyse prüft formale Strukturen von Arbeitsergebnissen

- Z. B. Code oder Modelle mittels Kontrollfluss- oder Datenflussanalyse
- Geeignetes statisches Analysewerkzeug notwendig
- Auch für natürliche Sprache
(Rechtschreibung, Grammatik, Lesbarkeit)



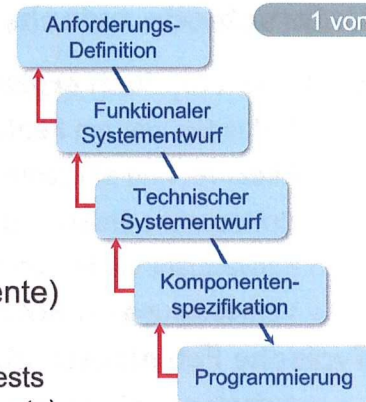
Arbeitsergebnisse, die durch statische Tests geprüft werden können

Fast jedes Arbeitsergebnis der Softwareentwicklung kann einem Review oder einer statischen Analyse unterzogen werden

- Spezifikationen (Fachkonzepte, Sicherheitsanforderungen, etc.)
- Epics, User-Stories, Abnahmekriterien
- Spezifikationen von Architektur und Entwürfen
- Code, Websites, Benutzeranleitungen
- Testmittel (Testkonzepte, Testfälle, Testabläufe, automatisierte Testskripte)
- Verträge, Pläne für Projekt, Zeit, Budget, Personen
- Test-Infrastruktur/Testumgebung mit deren unterschiedlichen Konfigurationen
- Modelle wie Aktivitätsdiagramme (modellbasiertes Testen, CTFL Model Based Tester)

Vorteile des statischen Tests

- **Einsatz früh** im SW-Entwicklungslebenszyklus
 - Die **Verifizierungen** im linken Zweig des V-Modells – also der **korrekten Umsetzung** der Anforderungen von einer Stufe zur nächsten – erfolgen per Review
- Eine hohe **Qualität der Dokumente** (aller Dokumente) fördert zwangsläufig die Produktqualität
 - Erkennen von Fehlerzuständen **bevor** dynamische Tests durchgeführt werden (Anforderungsreviews, Refinements)
- Kosten können eingespart werden
 - Fehlerzustände **frühzeitig** beheben, ist günstiger, als sie erst nach Fertigstellung und Einsatz im Produkktivsystem zu finden und zu beheben
 - Fehlerzustände frühzeitig zu finden und zu beheben, ist meist günstiger, als erst beim Einsatz dynamischer Tests zur Auffindung von Fehlerzuständen, die dann mit Fehlernachtests und Regressionstests verbunden sind



1 von 2

Vorteile des statischen Tests

Weitere Vorteile statischer Tests

- Fehlerzustände erkennen und beheben vor der Durchführung dynamischer Tests
- Fehlerzustände identifizieren, die mit dynamischen Tests schwerer zu finden sind
- Fehlerzustände aufdecken in Anforderungen vor der Umsetzung
 - Inkonsistenzen, Widersprüche, Mehrdeutigkeiten, Redundanzen, Auslassungen, etc.
- Wartungsfähiger Code, verbesserte Entwürfe (Entwicklungsproduktivität)
- Reduktion von Entwicklungskosten und -zeit
- Reduktion von Testkosten und -zeit
- Reduktion der Gesamtkosten der Qualität über die gesamte Lebenszeit der SW
 - Weniger Fehlerwirkungen im späteren Verlauf des Lebenszyklus oder nach Auslieferung
- Verbesserte Kommunikation zwischen den Teammitgliedern
 - Teilnahme an Reviews fördert gegenseitiges Verständnis und Austausch des Know-Hows
- Auffinden von Wartbarkeitsfehlern
 - Schlechte Modularisierung, schlechte Wiederverwendbarkeit von Komponenten, schlecht lesbarer Code, etc.

2 von 2

Unterschiede zwischen statischen und dynamischen Tests

Statische Verfahren **ergänzen** dynamische Testmethoden

- Gefunden werden **Fehlerzustände**, nicht **Fehlerwirkungen**
- Effektiver und effizienter als durch dynamische Tests zu finden
- Untersucht werden auch **Konzepte**, nicht nur ausführbare Software
- Fehlerzustände lassen sich später durch dynamische Tests oft schwerer finden
- Verbesserung der Konsistenz und der internen Qualität der Arbeitsergebnisse

Typische Fehlerzustände, die einfacher und günstiger zu beheben sind

- Anforderungsfehler (Inkonsistenzen, Widersprüche, Redundanzen, etc.)
- Entwurfsfehler (hohe Kopplung, geringe Kohäsion, ineffiziente Algorithmen, etc.)
- Programmierfehler (Variablen, die undefiniert oder nicht verwendet werden, etc.)
- Abweichungen von Standards (coding conventions, Programmierrichtlinien)
- Falsche Schnittstellenspezifikationen (verschiedene Metriken, Datenlängen)
- Schwachstellen in der Zugriffssicherheit (Pufferüberlauf, etc.)
- Verfolgbarkeit (Lücken, mangelnde Überdeckung, Ungenauigkeiten)

Zusammenfassung

- Bei **statischen Tests** wird das zu testende Objekt nicht ausgeführt
- Wichtige statische Prüftechniken sind **Reviews** und **statische Analysen**
- Reviews können **früh im Entwicklungsprozess** eingesetzt werden und ergänzen dynamische Testverfahren

Kapitel III – Statischer Test

- III/1 Statische Prüftechniken und der Testprozess
- III/2 Reviewprozess

Reviewprozess

Der Reviewprozess umfasst grundsätzlich die folgenden Hauptaktivitäten (die je nach Review-Art variieren können):



Reviewprozess für Arbeitsergebnisse

1 von 3

1. Planung

- Organisation des Reviews, Definition des Umfangs inkl. des Zwecks des Reviews, Auswahl der zu prüfenden Teile der Dokumente und der Qualitätsmerkmale, die bewertet werden sollen
- Schätzung von Aufwand und Zeitbedarf
- Auswahl der Teilnehmer und Zuordnung der Rollen
- Festlegung der **Eingangs- und Endekriterien**
- Prüfung, ob Eingangskriterien erfüllt sind

2. Reviewbeginn

- Ggf. Treffen aller Beteiligten zu einer Vorbesprechung
- Ausgabe der Prüfobjekte und anderer Materialien, wie Befundlistenvorlage, Checklisten und zugehörige Arbeitsergebnisse
- Erläuterung der Ziele, des Prozesses und der Dokumente

Reviewprozess für Arbeitsergebnisse

2 von 3

3. Individuelles Review (individuelle Vorbereitung)

- Review der Dokumenten / Prüfobjekte oder Teilen davon
- Potentielle Fehlerzustände / Empfehlungen / Fragen werden festgehalten

4. Befundkommunikation und -analyse (Reviewsitzung)

- Kommunikation identifizierter potenzieller Fehlerzustände (z.B. in einer Reviewsitzung)
- Analyse potenzieller Fehlerzustände, Zuweisung von Zuständigkeit und Status
- Bewertung und Dokumentation von Qualitätsmerkmalen
- Bewertung der Reviewbefunde gegenüber den Endekriterien, um eine Reviewentscheidung zu treffen (ablehnen, umfangreiche Änderungen notwendig, annehmen, vielleicht mit geringfügigen Änderungen)
- **Sachliche und fachliche Vorgehensweise** bei der Erörterung der zu prüfenden Aspekte
- Diskussion oder Protokollierung mit dokumentierten Ergebnissen

Reviewprozess für Arbeitsergebnisse

3 von 3

5. Fehlerbehebung und Bericht

- a) Der Autor behebt die gefundenen Fehlerzustände
Protokollierung des aktualisierten Fehlerstatus
- b) Das Management und der Reviewleiter erhalten das Protokoll der Sitzung mit folgenden Inhalten
 - Prüfobjekt und Vergleichsdokumente
 - Prüfen, ob die Fehlerzustände zugewiesen wurden
 - Sammeln von **Metriken**
 - Entscheidung, ob die **Reviewziele** erreicht wurden

Bei formaleren Review-Arten

- Messen der erzielten Ergebnisse anhand von Metriken
- Prüfung von Endekriterien

Rollen und Verantwortlichkeiten in einem formalen Review

1 von 2

- Ein typisches **formales** Review definiert folgende Rollen:
 - Management
 - Ist verantwortlich für die Reviewplanung, **initiiert das Review** und entscheidet über die Durchführung, bestimmt Teilnehmer, Budget und Fristen, überwacht die Kosteneffizienz
 - Trifft Steuerentscheidungen bei unangemessenen Ergebnissen
 - Reviewleiter
 - Übernimmt die Verantwortung für die Reviewdurchführung
 - Entscheidet über Teilnehmer, Ort und Zeit der Durchführung
 - (Review)Moderator
 - Steht **neutral** zum Prüfobjekt und leitet die Reviewsitzung
 - Lenkt die Diskussion / das Gespräch (zielorientiert)
 - **Vermittelt** zwischen den verschiedenen Standpunkten

Von der **Qualität des Moderators** hängt wesentlich der **Erfolg** des Reviews ab!

Rollen und Verantwortlichkeiten in einem formalen Review

2 von 2

- Protokollant
 - Protokolliert die Fehlerzustände, die bei individuellen Reviewaktivitäten gefunden wurden
 - Erfasst neue Fehlerzustände, offene Punkte und Entscheidungen der Sitzung
- Autor
 - **Ersteller** oder Vertreter der Ersteller des Prüfobjektes
 - Nimmt anschließend **selbst** die geforderten **Änderungen** vor
- Gutachter
 - Sind die **Fachexperten**, die an dem Review teilnehmen
 - Decken potentielle **Fehlerzustände, Problemstellen**, etc. auf
 - Haben idealerweise Fokus auf spezielle Aspekte (Tester, Programmierer, Benutzer, Betreiber, Businessanalysten, Gebrauchstauglichkeitsexperten)
 - **Schlagen vor**, welche Teile des Prüfobjektes korrigiert werden sollen und welche unverändert beibehalten werden können

Reviewarten (ISO 20246)

1 von 7

Reviews können von informell bis hin zu formal durchgeführt werden

- **Informelle** folgen keinem definierten Prozess, haben kein dokumentiertes Ergebnis
- **Formale** definieren die Teilnahme von Teams, dokumentierte Ergebnisse und Vorgehensweisen für systematische Durchführung

Die Formalität bestimmt sich aus

- Dem Softwareentwicklungslebenszyklus-Modell
- Der Reife des Entwicklungsprozesses
- Der Komplexität der zu prüfenden Arbeitsergebnisse
- Sicherheitsrelevanz des Produkts
- Rechtlichen oder regulatorischen Anforderungen/Prüfungsnachweis

Die Ziele eines Reviews bestimmen den Fokus eines Reviews

- Finden von **Fehlerzuständen**, Verständnis gewinnen, Training von Teilnehmern
- Diskussionen und Entscheidungen treffen auf Basis eines Konsens

Reviewarten

2 von 7

Die hier vorgestellten gängigsten Reviewarten sind:

- **Informelles Review, Walkthrough, Technisches Review, Inspektion**

Hauptziel der Reviews ist – Fehlerzustände aufdecken

- Verwendung der Reviewarten für unterschiedliche Zwecke
- Basieren auf den Projektbedürfnissen, den Ressourcen, der Produktart, Produktrisiken, Geschäftszweig, Unternehmenskultur

Reviewgegenstand – Arbeitsergebnisse

- Ein Arbeitsergebnisse kann Gegenstand mehrerer Reviewarten sein
- Die Reihenfolge ist nicht festgelegt (z.B. informelles Review vor einem technischen)

Durchführung in Form von Peer-Reviews

- Gleichgestellte Kollegen, gleiche Ebene in der Unternehmenshierarchie

Gefundene Fehlerarten variieren abhängig vom geprüften Arbeitsergebnis



Reviewarten

3 von 7

• **Informelles Review** (informal review)

Eine Art von Reviews, die keinem formalen (dokumentierten) Ablauf folgt. [ISO 20246]
z.B. Buddy-Check, Pairing (paarweises Zusammenarbeiten), paarweises Review)

- **Hauptzweck** – Erkennen von potentiellen Fehlerzuständen
- **Zusätzliche Zwecke** – Schnelle Lösung kleiner Probleme, Einbringen neuer Ideen oder Lösungen
- Keine Reviewsitzung erforderlich
- Ergebnisse können in Listenform weitergegeben werden
- Häufig in Form eines **Gegenlesens** der Prüfobjekte durch Kollegen
- Der Nutzen hängt vom Gutachter ab
- Checklisteneinsatz optional
- Einsatz verbreitet in der agilen Entwicklung



Reviewarten

4 von 7

- **Walkthrough**

Eine Reviewart, bei der ein Autor die Reviewteilnehmer durch ein Arbeitsergebnis leitet und die Teilnehmer Fragen stellen und potentielle Befunde kommentieren. [ISO 20246]

Siehe auch Peer Review

- **Hauptzweck** – Fehlerzustände finden, Softwareprodukt verbessern, Alternativen prüfen; Konformität mit Standards und Spezifikationen bewerten
- Zusätzliche Zwecke – Ideenaustausch (Verfahren, Stilvarianten), Konsenserzielung, Einarbeitung der Teilnehmer
- Der **Autor** selbst leitet die Sitzung
- Protokoll ist obligatorisch
- Protokolle potentieller Fehlerzustände und Reviewberichte
- **Durchführungsformen**: Szenarios, Dry Run (Probelauf), Simulationen
- Durchführung in der Praxis variiert von informell bis zu formal
- **Optional**: Nutzung von Checklisten, individuelle Vorbereitung



Reviewarten

5 von 7

- **Technisches Review** (technical review)

Eine formale Reviewart, bei der ein Team von technisch qualifizierten Personen die Eignung eines Arbeitsergebnisses für seine beabsichtigte Verwendung prüft und Abweichungen von Spezifikationen oder Standards identifiziert. [Gilb and Graham, ISO 20246]

- **Hauptzweck** – Finden von Fehlerzuständen, Konsensgewinnung
- Weitere Zwecke – Bewertung von Qualität, Vertrauen in das Arbeitsergebnis erzielen, neue Ideen einbringen, Verbesserung zukünftiger Arbeitsergebnisse
- **Fachexperten** werden benötigt als Gutachter
- Vorbereitung der Gutachter **vor der Sitzung**
- Eine Reviewsitzung ist optional, wird aber dann geleitet durch einen geschulten **Reviewmoderator** (nicht den Autor)
- Protokollführung ist optional (nicht Autor), Optional: Einsatz von Checklisten
- **Reviewbericht** mit Liste der Befunde üblich (nicht vom Autor erstellt)
- Z. B. als Peer Review (ohne Management) durchführbar



Reviewarten

6 von 7

- **Inspektion** (inspection)

Eine formale Reviewart deren Ziel die Identifizierung von Befunden in einem Arbeitsprodukt ist, und welche Messungen zur Verbesserung des Reviewprozesses und des Softwareentwicklungsprozesses liefert. [ISO 20246]

- **Hauptzweck**

- Erkennen von Fehlerzuständen
 - Bewertung der Qualität
 - Vertrauen in das Arbeitsergebnis schaffen, durch das Lernen des Autors
 - Grundursachenanalyse zur Vermeidung zukünftiger ähnlicher Fehlerzustände

- **Weitere Zwecke**

- Motivation des Autors
 - Befähigen des Autors in Zukunft bessere Arbeitsergebnisse zu erstellen
 - Verbesserung des Softwareentwicklungsprozesses
 - Konsens erzielen

Reviewarten

7 von 7

- **Inspektion – Ablauf**

- **Planen – Kick-off** als Reviewbeginn – **individuelle Vorbereitung** als individuelles Review – **Sitzung** als Befundkommunikation und -analyse – **Fehlerbehebung und Bericht**

- Definierter Prozess, formal dokumentierte Ergebnisse nach Regeln und Checklisten
 - Rollen und Verantwortlichkeiten sind klar definiert und verpflichtend (s. oben)
 - **Individuelle Vorbereitung** vor der Sitzung
 - Eventuell **Vorleser**, der das Prüfobjekt während der Sitzung laut vorliest
 - Gutachter sind **gleichrangig** mit dem Autor oder Experten in anderen Fachrichtungen
 - Eingangs- und Endekriterien werden festgelegt und eingehalten
 - Ein formaler Inspektionsbericht/Befundliste muss erstellt werden
 - Sitzungsleitung erfolgt durch einen geschulten Moderator (nicht der Autor)
 - Erstellung von Reviewberichten und Protokollierung potentieller Fehlerzustände
 - Sammlung von Metriken zur Prozessverbesserung (für SE- und auch Prüfprozess)

Die Anwendung von Reviewverfahren

1 von 4

- Reviewverfahren dienen zur **Anleitung** der individuellen **Vorbereitung** auf Reviews, also der originären Prüfarbeit durch den Gutachter
- Verwendung bei allen vorgestellten Reviewarten möglich, ihre Effektivität variiert dabei
- Folgende Reviewverfahren werden vorgestellt
 - Ad hoc
 - Checklistenbasiert
 - Szenarien und Dry Runs (Probelaufe)
 - Rollenbasiert
 - Perspektivisch

Die Anwendung von Reviewverfahren

2 von 4

- **Ad hoc**
 - Wenig oder keine Anleitung zur Durchführung der Aufgabe als Gutachter
 - Sequentielles Lesen und Identifizieren / Dokumentieren der Befunde
 - **Wenig Vorbereitung erforderlich**
 - Ergebnis hängt von den Fähigkeiten des Gutachters ab
 - Kann zu vielen doppelten Befunden führen
- **Checklistenbasiert** – systematisches Verfahren
 - Systematische Überdeckung typischer Fehlerarten und Befundarten
 - Befunde werden auf Basis von **vorab verteilten Checklisten** erkannt
 - Ein Set von Fragen, erstellt auf Basis potentieller Fehlerzustände und Erfahrungen
 - **Zugeschnitten** auf das Arbeitsergebnis, den **Reviewgegenstand**
 - Regelmäßige **Aktualisierung** und Ergänzung aus vorhergegangenen Erkenntnissen
 - Fehlerzustände außerhalb der Checkliste nicht vernachlässigen

Die Anwendung von Reviewverfahren

3 von 4

- **Szenarien** und Dry Runs (Probelaufe) – strukturierte Richtlinien
 - Strukturierte Richtlinien zum Durchlesen eines Arbeitsergebnisses
 - Probelaufe auf Basis der **erwarteten Nutzung** des Arbeitsergebnisses
 - Anwendungsfälle können als Basis genommen werden
 - **Unterstützung** bei der Suche nach spezifischen **Fehlerarten**
 - Weitere Fehlerarten, wie fehlende Leistungsmerkmale, sollten beachtet werden
- **Rollenbasiert** – sich in die Situation des Anwenders versetzen
 - Individuelle Stakeholder-Rollen werden eingenommen
 - Als Personen: erfahren, unerfahren, Senioren, Kinder etc.
 - Als Organisationen: Benutzeradministrator, Performancetester, Systemadministrator
 - viele der Prinzipien des perspektivischen Reviews passen auch hier

Die Anwendung von Reviewverfahren

4 von 4

- **Perspektivisch** – perspektivisches Lesen
 - Betrachtung unterschiedlicher Standpunkte unterschiedlicher Stakeholder
 - Typische Sichtweisen: Endanwender, Marketing, Designer, Tester, Betrieb
 - Erfolg: weniger Dopplungen von Befunden unter den Gutachtern, mehr Tiefe im individuellen Review
 - Prüfung, ob sich aus dem Arbeitsergebnis z.B. ein Entwurf für Testfälle ableiten lässt
 - Einsatz von **Checklisten** (möglich)
 - Studien haben gezeigt, dass **perspektivisches Lesen** das **effektivste** allgemeine Verfahren für das Review von Anforderungen und technischen Arbeitsergebnissen ist
 - Voraussetzung:
 - Korrekter Einbezug der Sichtweisen der Stakeholder
 - Risikobasierte, angemessene Gewichtung

Erfolgsfaktoren für Reviews – organisatorisch

1 von 2

- Definition klarer Ziele in der Reviewplanung (messbares Endekriterium)
- Angemessene Anwendung der Reviewarten und Techniken in Abhängigkeit zu den zu reviewenden Unterlagen und Teilnehmern
- Alle **Checklisten** sind aktuell und enthalten Hauptrisiken
- Alle Reviewverfahren nutzen zur effektiven Identifizierung von Fehlerzuständen
- Autoren schon früh Rückmeldung geben zu Fehlerzuständen, Beginn mit den Reviews schon bei den ersten Teildokumenten
- Ausreichend Zeit und Budget für die Vorbereitung einplanen
- Reviews planen mit angemessener Vorankündigung
- **Managementunterstützung** schon in der Projektplanung, z.B. Bereitstellung von angemessener Zeit für die Reviewaktivitäten bereits im Projektplan

Erfolgsfaktoren für Reviews – personenbezogen

2 von 2

- Auswahl der richtigen Personen, passend zu den Reviewzielen
 - Unterschiedliche Sichtweisen und Fähigkeiten (kein Streit unter den Gelehrten)
- Tester früh einbeziehen, sie sind wichtige Gutachter
 - Frühes Kennenlernen des Arbeitsergebnisses zur Vorbereitung von Tests, früher und effektiver
- Den Details angemessene Aufmerksamkeit und Zeit widmen
- Aufteilung der Reviews in kleine Schritte zum Erhalt der Konzentration der Gutachter
- Reviews **zielorientiert** durchzuführen (Fehlerzustände in den Reviewobjekten wertfrei und objektiv identifizieren), keine Bewertung der Teilnehmer
- Gute Reviewleitung, Review wird von den Teilnehmern als wertvoller Beitrag gesehen
- Vertrauensvolle Atmosphäre, keine personenbezogenen Bewertungen
- Keine Körpersprache, die Langeweile, Frust oder Feindseligkeit ausdrückt
- Angemessene Schulungen für die formalen Reviewarten wie die Inspektion
- Förderung einer Kultur des Lernens und der Prozessverbesserung

Übersicht über die Reviewarten – Vor- und Nachteile

Reviewart	Vorteile	Nachteile
Inspektion	strukturierte Sitzung Protokoll	langer Vorlauf viele Rollen, wie Moderator
Walkthrough	kurzer Vorlauf längere Sitzungsdauer Protokollant obligatorisch formal bis informell	Autoren gesteuert
Technisches Review	Gesamtbewertung Standards Protokoll Reviewsitzung optional	langer Vorlauf evtl. viele Rollen
Informelles Review	kurzer Vorlauf kostengünstig keine Sitzung	kein Protokoll

Übersicht über die Reviewarten – Eigenschaften

	Informelles Review	Technisches Review	Inspektion	Walkthrough
Ziel	- Aufdeckung von Unklarheiten und Fehlern - Bewertung der Qualität	- Beurteilung der Eignung des Prüfobjekts für den beabsichtigten Einsatz - Aufdeckung von Unklarheiten und Fehlern - Bestimmung der Qualität des Prüfobjekts	- Aufdeckung von Unklarheiten und Fehlern - Bestimmung der Qualität des Prüfobjekts - Verbesserung des Entwicklungsprozesses	- Fehlerzustände finden - Wissenstransfer - Verständnis erzielen
Rolle des Autors	Meist der Initiator	Nicht beteiligt oder beantwortet Fragen während Sitzung	Beantwortet Fragen während Sitzung	Stellt Prüfobjekt vor und beantwortet Fragen während der Sitzung
Vorbereitung	Mittlerer Aufwand: Verwendung von Prüfkriterien nicht vorgeschrieben	Hoher Aufwand: - Gutachter prüfen das Prüfobjekt gemäß Reviewzielen (Checklisten) - Befunde werden schriftlich festgehalten - Verwendung ausschließlich offizieller Referenzdokumente - Verwendung von formalen Prüfkriterien	Hoher Aufwand: - Gutachter prüfen das Prüfobjekt gemäß Reviewzielen (Checklisten) - Befunde werden schriftlich festgehalten - Verwendung von formalen Prüfkriterien	Geringer Aufwand (z.T. kann sogar darauf verzichtet werden): Verwendung von Prüfkriterien nicht vorgeschrieben
Review-sitzung	Findet meist nicht statt	Erfassung und Diskussion der Befunde mit dem Ziel, ein einstimmiges Gesamturteil zu fällen	Formal und straff geführte Sitzung: - Erfassung - Diskussion - Entscheid	Inhalte des Prüfobjektes werden (ablaurorientiert) vorgetragen



Übung III.1: Review durchführen

(15 Min. Gruppenarbeit)

Führe ein Walkthrough für die gegebene Spezifikation durch
(Online-Shop – Seite 4 Übungen)

Zusammenfassung

- **Phasen eines Reviews:** Planung – Reviewbeginn – individuelles Review – Befundkommunikation und -analyse – Fehlerbehebung und Bericht
- **Aufgaben und Rollen im Review:** Management – Reviewleiter – Moderator – Protokollant – Autor – Gutachter
- **Reviewarten:** Inspektion – Walkthrough – Technisches Review – Informelles Review
- **Reviewverfahren:** Ad hoc – Checklistenbasiert – Szenarien und Dry Runs – Rollenbasiert – Perspektivisch
- **Schulungen** in Reviewtechniken
- Jedes Review hat klar vordefinierte **Ziele**
- **Eingangs- und Endekriterien** festlegen

Schlüsselbegriffe

- Ad-hoc-Review
- Checklistenbasiertes Review
- Dynamischer Test
- Formales Review
- Informelles Review
- Inspektion
- Perspektivisches Lesen
- Review
- Rollenbasiertes Review
- Statische Analyse
- Statischer Test
- Szenariobasiertes Review
- Technisches Review
- Walkthrough

Kapitel IV – Testverfahren

- IV/1 Kategorien von Testverfahren
- IV/2 Black-Box-Testverfahren
- IV/3 White-Box-Testverfahren
- IV/4 Erfahrungsbasierte Testverfahren

Lernziele Testverfahren

1 von 2

4.1 Kategorien von Testverfahren

FL-4.1.1 (K2) Die Eigenschaften, Gemeinsamkeiten und Unterschiede zwischen Black-Box-Testverfahren, White-Box-Testverfahren und erfahrungsbasierten Testverfahren erklären können

4.2 Black-Box-Testverfahren

FL-4.2.1 (K3) Die Äquivalenzklassenbildung anwenden können, um Testfälle aus vorgegebenen Anforderungen abzuleiten

FL-4.2.2 (K3) Die Grenzwertanalyse anwenden können, um Testfälle aus vorgegebenen Anforderungen abzuleiten

FL-4.2.3 (K3) Entscheidungstabellentests anwenden können, um Testfälle aus vorgegebenen Anforderungen abzuleiten

FL-4.2.4 (K3) Zustandsbasierte Tests anwenden können, um Testfälle aus vorgegebenen Anforderungen abzuleiten

FL-4.2.5 (K2) Erklären können, wie man Testfälle aus einem Anwendungsfall ableitet

Lernziele Testverfahren

2 von 2

4.3 White-Box-Testverfahren

- FL-4.3.1 (K2) Anweisungsüberdeckung erklären können
- FL-4.3.2 (K2) Entscheidungsüberdeckung erklären können
- FL-4.3.3 (K2) Die Bedeutung von Anweisungs- und Entscheidungsüberdeckung erklären können

4.4 Erfahrungsbasierte Testverfahren

- FL-4.4.1 (K2) Die intuitive Testfallermittlung erklären können
- FL-4.4.2 (K2) Exploratives Testen erklären können
- FL-4.4.3 (K2) Checklistenbasiertes Testen erklären können

Kapitel IV – Testverfahren

- IV/1 Kategorien von Testverfahren
- IV/2 Black-Box-Testverfahren
- IV/3 White-Box-Testverfahren
- IV/4 Erfahrungsbasierte Testverfahren

Auswahl von Testverfahren

1 von 3

- Informationsstand zum Testobjekt
 - Von welcher Art und Komplexität ist die Komponente / das Systems?
 - Wie hoch ist Nutzungshäufigkeit der Software?
 - Wird mit einem Softwareentwicklungslebenszyklus-Modell gearbeitet?
 - Welche Werkzeuge sind verfügbar?
 - Ist Dokumentation verfügbar, aktuell und von welcher Qualität?
- Risikoaspekte und Kundenanforderungen
 - Wie groß sind potenzielle Schäden beim Nichtentdecken von Fehlerzuständen und -wirkungen? (Risikoart und Risikostufe)
 - Gibt es Kundenanforderungen? Vertragliche Anforderungen?
 - Gibt es regulatorische Anforderungen an die Testdurchführung oder die Testüberdeckung? Standards?

Auswahl von Testverfahren

2 von 3

- Projekt-Rahmenbedingungen
 - Wie viel Zeit und Budget steht zum Testen zur Verfügung?
 - Welche Personen stehen als Tester zur Verfügung? (Kenntnisse, Fähigkeiten)
 - Welche Erfahrung ist vorhanden in der Anwendung von Testverfahren?
 - Mit welchen Softwareentwicklungsmodellen wird gearbeitet?
 - Welche Testziele sind zu beachten?
- Eigenschaften des Testobjekts
 - Welche Testfälle und Testdaten sind für das Testobjekt möglich?
 - Welche Arten von Fehlerzuständen sind zu erwarten (Komponente / System)?

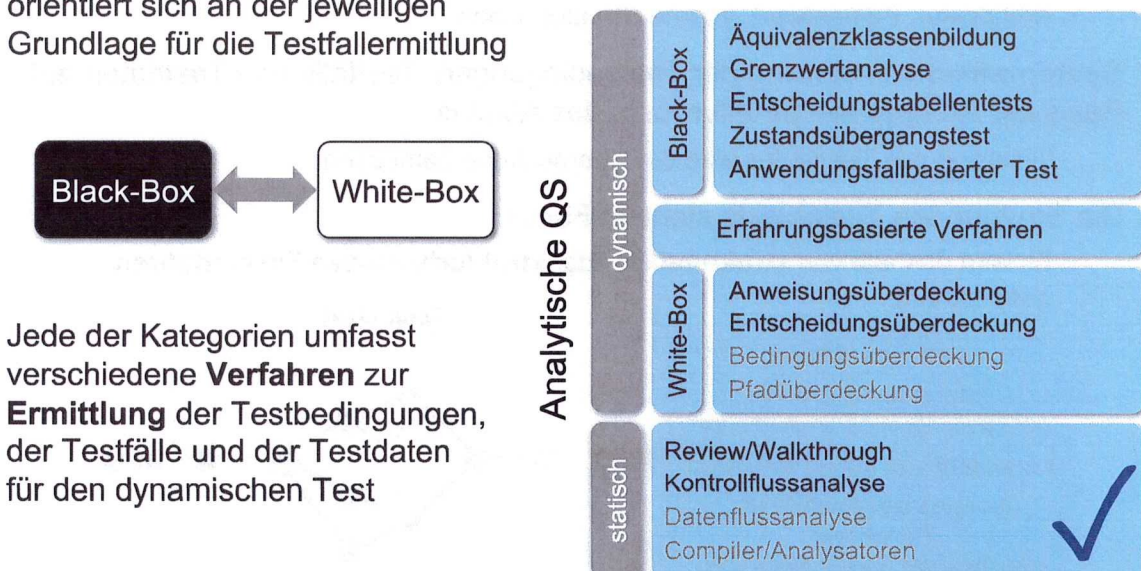
Auswahl von Testverfahren

3 von 3

- Teststufen
 - Auf welchen Teststufen sind die welche Tests durchführbar?
 - Einige Verfahren sind auf allen Teststufen einsetzbar, andere weniger geeignet
- **Kombination** von Testverfahren
 - Wie wird eine angemessene Überdeckung gewährleistet?
 - Welche Verfahren sollten geeignet kombiniert werden?
 - Welche Testverfahren eignen sich unter dem gegebenen Testaufwand und erzielen die besten Ergebnisse?
 - Gibt es Vorgaben für die Formalität (sehr informell – sehr formal)?
 - Welche Einschränkungen liegen vor, bezüglich
 - Reife des Test- und Entwicklungsprozesses, zeitliche Beschränkungen
 - Informationssicherheits- oder regulatorische Anforderungen
 - Kenntnisse und Fähigkeiten der involvierten Personen (nicht nur der Tester)
 - Gewähltes Softwareentwicklungslebenszyklus-Modell?

Kategorien von Testverfahren und ihre Eigenschaften

Dynamische Testverfahren werden in **drei Kategorien** unterteilt, diese orientiert sich an der jeweiligen Grundlage für die Testfallermittlung



Black-Box-Testverfahren

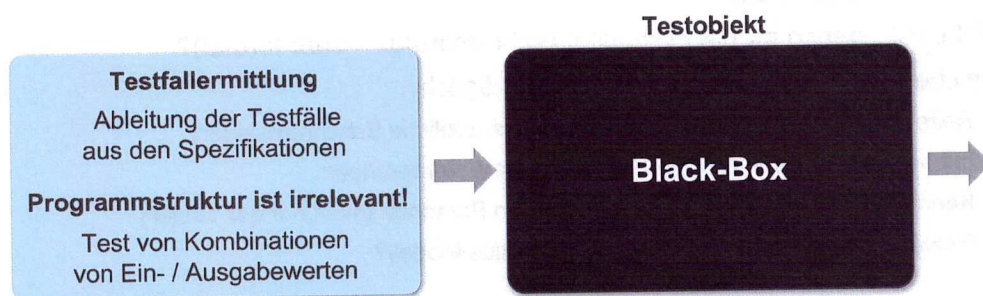
Konzentration auf die **Eingaben und Ausgaben** des Testobjekts.

Ableitung der Testbedingungen, Testfälle, Testdaten **systematisch** aus der **Testbasis** wie zum Beispiel

- Softwareanforderungen, Spezifikationen, Anwendungsfälle, User-Stories, Geschäftsprozesse

Funktionale und nicht-funktionale Tests stehen im Fokus

- Es wird deshalb von **spezifikationsbasierten Testverfahren** gesprochen



White-Box-Testverfahren

Analyse der (inneren) Struktur des Testobjekts

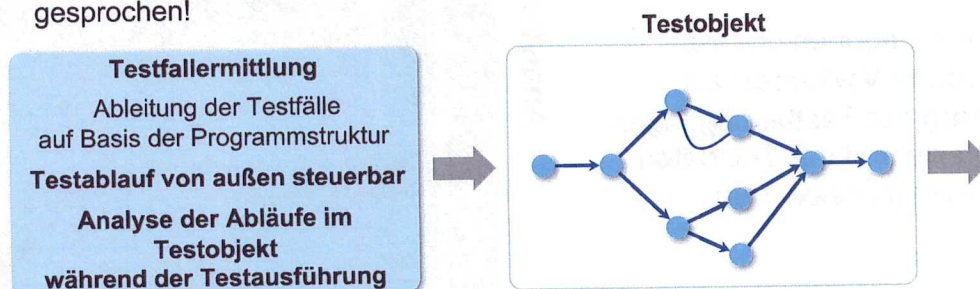
- Architektur, Feinentwurf, interne Struktur, Code

Systematische Erstellung der Testbedingungen, Testfälle und Testdaten auf Basis der Analyse der Struktur resp. des Ablaufs

- Während des Testlaufes wird die Überdeckung gemessen

Die Struktur des Testobjekts steht im Fokus

- Es wird deshalb von **strukturellen oder strukturbasierten Testverfahren** gesprochen!



Gängige Merkmale der Verfahren

- **Black-Box-Verfahren**
 - **Lücken** und Abweichungen zwischen der Testbasis (Anforderungen, etc.) und der Realisierung erkennen
 - **Überdeckung** – Anzahl der getesteten Elemente in Verhältnis zu den mit dem Verfahren ermittelten Elementen
- **White-Box-Verfahren**
 - **Überdeckung** – Anzahl der getesteten Elemente einer Struktur im Verhältnis zu der Gesamtanzahl von Elementen
- **Erfahrungsbasierte Verfahren**
 - **Testbasis** bilden die Erfahrung und Kenntnisse der Tester, Entwickler, User, etc.
 - **Fokus**: die erwartete Nutzung des Systems, die Umgebung, mögliche Fehlerzustände, Verteilung der Fehlerzustände
 - Oft eine **wichtige Ergänzung** zu Black-Box- und White-Box-Verfahren

(ISO 29119-4 Beschreibungen von Testverfahren und ihren Überdeckungsmaßen)

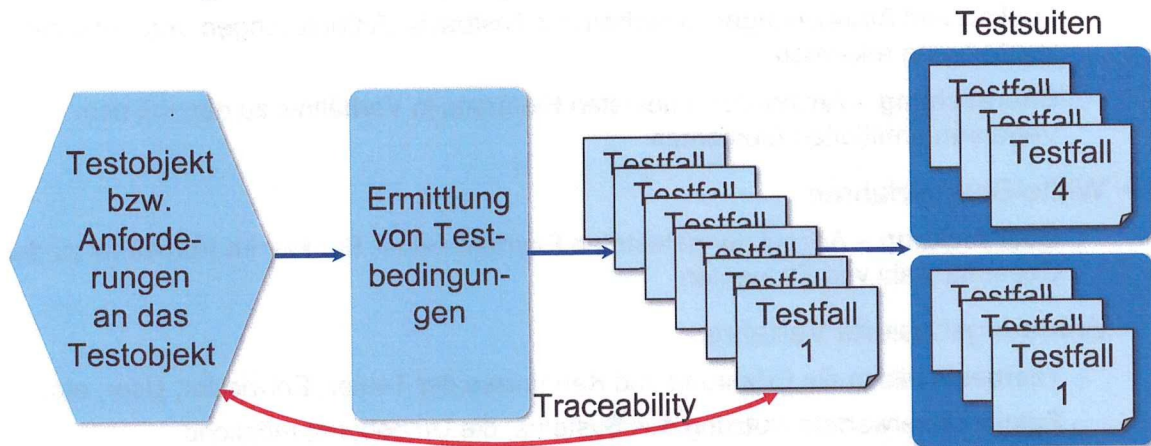


Begriffe

- **Black-Box Testverfahren** (black-box test design technique)
Ein Verfahren zur Herleitung und/oder Auswahl von Testfällen, das auf einer Analyse der funktionalen oder nicht-funktionalen Spezifikation einer Komponente oder eines Systems basiert, ohne Berücksichtigung ihrer internen Struktur.
- **White-Box-Testverfahren** (white-box test design technique)
Ein Verfahren zur Herleitung und Auswahl von Testfällen, basierend auf der internen Struktur einer Komponente oder eines Systems. Überdeckungsmessung auf Basis der Testobjekte und der angewandten Technik.
- **Erfahrungsbasierte Testverfahren** (experienced-based test design technique)
Eine Vorgehensweise, mit der Testfälle auf Basis der Erfahrungen, dem Wissen und der Intuition der Tester abgeleitet und/oder ausgewählt werden.

Herleitung der Testfälle und Verfolgbarkeit / Traceability

Die Herleitung von Testfällen erfolgt mittels eines gesteuerten Prozesses



Je nach konkreter **Projektsituation** und Reifegrad der Prozesse kann der Entwurf von Testfällen und Testsequenzen sehr **formal** ablaufen **oder auch informell** durchgeführt werden

Zusammenfassung

- Testfälle können nach **unterschiedlichen Verfahren** hergeleitet werden
- Steht die **Funktionalität** eines Testobjektes im Vordergrund, spricht man vom spezifikationsorientierten Verfahren (Black-Box)
- Steht der **innere Aufbau** eines Testobjektes im Vordergrund, spricht man vom strukturorientierten Verfahren (White-Box)
- **Erfahrungsbasierte** Verfahren nutzen hauptsächlich die **Kenntnisse** und das Wissen von Testern, Entwicklern und Anwendern

Kapitel IV – Testverfahren

- IV/1 Kategorien von Testverfahren
- IV/2 Black-Box-Testverfahren
- IV/3 White-Box-Testverfahren
- IV/4 Erfahrungsbasierte Testverfahren

Allgemeines

- Folgende Black-Box Testverfahren werden vertieft behandelt
 - Äquivalenzklassenbildung
 - Grenzwertanalyse
 - Entscheidungstabellentests
 - Zustandsübergangstest
 - Anwendungsfallbasierter Test (use cases)
- Es handelt sich dabei um die wichtigsten und gebräuchlichsten Verfahren
- Ziele der **systematischen Testfallermittlung** sind eine **Reduktion** der Testfälle und sicherzustellen, dass die verbleibenden Testfälle eine **hohe Wahrscheinlichkeit** aufweisen, **Fehlerzustände zu finden**

Funktionales Testen

Ziel des funktionalen Testens ist die Prüfung auf **Vollständigkeit** und **Korrektheit** der Funktionen

- Die Software wird gegen ihre Spezifikationen geprüft
- Sind alle geforderten Funktionen korrekt enthalten?

Die Durchführung des Tests sollte redundanzarm und dennoch umfassend sein

- Nicht mehr als nötig testen!



Quelle: Wikipedia



Äquivalenzklassenbildung

• Äquivalenzklassenbildung (equivalence partitioning)

Ein Black-Box-Testverfahren, bei dem die Testfälle im Hinblick auf die Ausführung von Äquivalenzklassen entworfen werden, wobei von jeder Äquivalenzklasse ein Repräsentant genutzt wird. [ISO 29119]

Bei der Äquivalenzklassenbildung werden die **Definitionsbereiche der Werte** betrachtet

- **Eingabewerte** des Programms (übliche Nutzung der ÄK-Methode)
- **Ausgabewerte** des Programms (seltene Nutzung der ÄK-Methode)

Prinzip der Äquivalenzklassenbildung

- Alle Werte, für die ein **identisches Verhalten** des Testobjekts **erwartet** wird, werden zu einer Äquivalenzklasse (ÄK) zusammengefasst
- Äquivalenzklassen dürfen sich **nicht überlappen** und **keine Lücken** aufweisen
- Zerlegung in **gültige** (gÄK) und **ungültige** (uÄK) Äquivalenzklassen
- Werte mit Sonderbehandlung (z. B. nicht numerisch)

Äquivalenzklassenbildung – Beispiel ÄK

1 von 3

Äquivalenzklassen werden für **gültige** und für **ungültige** Eingabewerte erstellt

- Ist der Wert x definiert als $0 \leq x \leq 100$, ergeben sich daraus zunächst drei Äquivalenzklassen:
 - (1) alle Werte für x zwischen 0 und 100 (gültige ÄK)
 - (2) alle Werte für $x > 100$ (ungültige ÄK)
 - (3) alle Werte für $x < 0$ (ungültige ÄK)
- Ferner können für x nicht-numerische Werte eingesetzt werden – dies stellt eine weitere mögliche Äquivalenzklasse dar

Getestet wird jeweils nur mit **einem Repräsentanten** der ÄK

- Man erwartet für jeden anderen Vertreter der Äquivalenzklasse ein **identisches bzw. analoges Verhalten**

< 0	0 – 100	> 100
-----	---------	-------

Äquivalenzklassenbildung – Beispiel ÄK

2 von 3

Ein Programm verlangt die Eingabe eines **Prozentwertes** unter den folgenden Rahmenbedingungen

- Nur ganzzahlige Werte (Integer) sind erlaubt
- 0 ist die gültige Untergrenze und 100 die gültige Obergrenze

Gültig sind hier alle Zahlen ≥ 0 und ≤ 100 , ungültig sind alle negativen Werte, alle Werte über 100, alle Dezimalwerte sowie nicht numerische Eingaben (z. B. „Null“)

- Gültige Äquivalenzklasse: $0 \leq x \leq 100$
- 1. ungültige Äquivalenzklasse: $x < 0$
- 2. ungültige Äquivalenzklasse: $x > 100$
- 3. ungültige Äquivalenzklasse: $x = \text{kein Integer}$
- 4. ungültige Äquivalenzklasse: $x = \text{nicht-numerisch (n.n.)}$

< 0	0 – 100	> 100
-----	---------	-------

Äquivalenzklassenbildung – Beispiel ÄK

3 von 3

Der Prozentwert wird in einem Balkendiagramm dargestellt.
Zusätzlich gilt nun folgendes (jeweils beide Werte einschließlich):

- Wert zwischen 0 und 15: grauer Balken
- Wert zwischen 16 und 50: grüner Balken
- Wert zwischen 51 und 85: gelber Balken
- Wert zwischen 86 und 100: roter Balken

Jetzt ergeben sich statt einer gültigen ÄK vier gültige ÄK:

- 1. gültige Äquivalenzklasse: $0 \leq x \leq 15$
- 2. gültige Äquivalenzklasse: $16 \leq x \leq 50$
- 3. gültige Äquivalenzklasse: $51 \leq x \leq 85$
- 4. gültige Äquivalenzklasse: $86 \leq x \leq 100$



Äquivalenzklassenbildung – Repräsentantenauswahl

Abschließend wird für jede ÄK ein Repräsentant gewählt,
dies kann **jeder beliebige** Wert aus der Äquivalenzklasse sein

Folgende Repräsentanten werden gewählt:

Variable	Äquivalenzklasse	Repräsentant
Prozentsatz	ÄK ₁ : $0 \leq x \leq 15$	
	ÄK ₂ : $16 \leq x \leq 50$	
	ÄK ₃ : $51 \leq x \leq 85$	
	ÄK ₄ : $86 \leq x \leq 100$	
Prozentsatz	ÄK ₅ : $x < 0$	
	ÄK ₆ : $x > 100$	
	ÄK ₇ : x kein Integer	
	ÄK ₈ : x kein numerischer Wert	



Übung IV.1: Äquivalenzklassen

(10 Min. Einzelarbeit / 15 Min. Ergebnisdiskussion)

Aus der gegebenen Spezifikation (Online-Shop – Seite 5 Übungen) sind

- Die Eingabewerte zu bestimmen
- Äquivalenzklassen für die Eingabe-Wertebereiche zu definieren
 - Gültige Äquivalenzklassen
 - Ungültige Äquivalenzklassen

Äquivalenzklassenbildung – Vorgehensweise

Beispiel ÄK 2:

- Ein Programmteil berechnet den Preis eines Produkts aus dem Warenwert, einem Rabatt und Versandkosten

Variable	Äquivalenzklasse	Status	Repräsentant
Warenwert	ÄK ₁₁ : $x \geq 0$	Gültig	1000,00
	ÄK ₁₂ : $x < 0$	Ungültig	-1000,00
	ÄK ₁₃ : x kein numerischer Wert	Ungültig	P
Rabatt	ÄK ₂₁ : $0\% \leq x \leq 100\%$	Gültig	10%
	ÄK ₂₂ : $x < 0\%$	Ungültig	-10%
	ÄK ₂₃ : $x > 100\%$	Ungültig	200%
	ÄK ₂₄ : x kein numerischer Wert	Ungültig	Q
Versandkosten	ÄK ₃₁ : $x = 6$	Gültig	6
	ÄK ₃₂ : $x = 9$	Gültig	9
	ÄK ₃₃ : $x = 12$	Gültig	12
	ÄK ₃₄ : $x \neq \{6, 9, 12\}$	Ungültig	4
	ÄK ₃₅ : x kein numerischer Wert	Ungültig	R

Folgende Annahmen:

- **Warenwert** definiert als positive Zahl mit zwei Nachkommastellen
- **Rabatt** definiert als Prozentzahl ohne Nachkommastellen von 0% bis 100% jeweils einschließlich
- Für die **Versandkosten** werden feste Werte geliefert, die 6, 9, oder 12 betragen können

Äquivalenzklassenbildung – positive Testfälle

Aus den **gültigen** Äquivalenzklassen lassen sich folgende Kombinationen bzw. Testfälle (T01, T02 und T03) bilden

Variable	Äquivalenzklasse	Status	Repräsentant	T01	T02	T03
Warenwert	ÄK ₁₁ : $x \geq 0$	Gültig	1000,00	*	*	*
	ÄK ₁₂ : $x < 0$	Ungültig	-1000,00			
	ÄK ₁₃ : x kein numerischer Wert	Ungültig	P			
Rabatt	ÄK ₂₁ : $0\% \leq x \leq 100\%$	Gültig	10%	*	*	*
	ÄK ₂₂ : $x < 0\%$	Ungültig	-10%			
	ÄK ₂₃ : $x > 100\%$	Ungültig	200%			
	ÄK ₂₄ : x kein numerischer Wert	Ungültig	Q			
Versandkosten	ÄK ₃₁ : $x = 6$	Gültig	6	*		
	ÄK ₃₂ : $x = 9$	Gültig	9		*	
	ÄK ₃₃ : $x = 12$	Gültig	12			*
	ÄK ₃₄ : $x \neq \{6, 9, 12\}$	Ungültig	4			
	ÄK ₃₅ : x kein numerischer Wert	Ungültig	R			

Äquivalenzklassenbildung – negative Testfälle

Ungültige ÄK dürfen nur mit gültigen ÄK kombiniert werden (zwecks Vermeidung von Fehlermaskierungen); die Werte der Repräsentanten bleiben unverändert

Variable	Äquivalenzklasse	Status	Repräsentant	T04	T05	T06	T07	T08	T09	T10
Warenwert	ÄK ₁₁ : $x \geq 0$	Gültig	1000,00			*	*	*	*	*
	ÄK ₁₂ : $x < 0$	Ungültig	-1000,00	*						
	ÄK ₁₃ : x kein numerischer Wert	Ungültig	P		*					
Rabatt	ÄK ₂₁ : $0\% \leq x \leq 100\%$	Gültig	10%	*	*				*	*
	ÄK ₂₂ : $x < 0\%$	Ungültig	-10%			*				
	ÄK ₂₃ : $x > 100\%$	Ungültig	200%				*			
	ÄK ₂₄ : x kein numerischer Wert	Ungültig	Q					*		
Versandkosten	ÄK ₃₁ : $x = 6$	Gültig	6	*	*	*	*	*		
	ÄK ₃₂ : $x = 9$	Gültig	9							
	ÄK ₃₃ : $x = 12$	Gültig	12							
	ÄK ₃₄ : $x \neq \{6, 9, 12\}$	Ungültig	4						*	
	ÄK ₃₅ : x kein numerischer Wert	Ungültig	R							*

Äquivalenzklassenbildung – Testfälle insgesamt

Es ergeben sich daraus insgesamt 10 Testfälle

- 3 positive Testfälle und 7 Negativtestfälle

Variable	Status	Repräsentant	T01	T02	T03	T04	T05	T06	T07	T08	T09	T10
Warenwert	Gültig	1000,00	*	*	*			*	*	*	*	*
	Ungültig	-1000,00				*						
	Ungültig	P					*					
Rabatt	Gültig	10%	*	*	*	*	*				*	*
	Ungültig	-10%						*				
	Ungültig	200%							*			
	Ungültig	Q								*		
Versandkosten	Gültig	6	*			*	*	*	*	*		
	Gültig	9		*								
	Gültig	12			*							
	Ungültig	4									*	
	Ungültig	R										*

Ausgabewerte als Basis für Äquivalenzklassenbildung

Bildung von Äquivalenzklassen auf Basis von **Ausgabewerten**

- Analog kann das Verfahren auch für eine Unterteilung aller definierten Ausgabewerte eingesetzt werden
- Die Variable (das Element) ist dann eine Ausgabe (z. B. ein Ausgabefeld einer GUI)
- Aus den möglichen definierten Ausgabewerten werden entsprechend wieder Äquivalenzklassen gebildet
 - ÄK umfasst alle Werte, für die ein identisches Ausgabeverhalten unterstellt wird
- Für jede Äquivalenzklasse wird ein Repräsentant gewählt
- Für die Repräsentanten wird abschließend bestimmt, über welche Eingabewerte sie zu erreichen sind

Höherer Aufwand, da Eingabewerte für verschiedene fest definierte Ausgaben gesucht werden müssen

Äquivalenzklassenbildung – Überdeckung

Übergang von Definitionen / Spezifikationen zu Äquivalenzklassen

- Oft eine schwere Aufgabe, da **unpräzise** oder lückenhafte Dokumente vorliegen
- **Ungenaue Grenzen** der Definitionsbereiche oder fehlende Angaben zur genauen Definition erschweren eine Umsetzung
- Häufig sind hier **Rücksprachen** mit dem **Kunden** notwendig

Endkriterium kann ein bestimmter **Überdeckungsgrad** sein

- Wie viele ÄK sind im Verhältnis zu allen definierten ÄK getestet worden?

$$\text{ÄK-Überdeckung in \%} = \frac{\text{Anzahl getesteter ÄK}}{\text{Gesamtzahl ÄK}} \times 100$$

Äquivalenzklassenbildung – Repräsentantenauswahl

- **Jeder mögliche Wert** einer ÄK kann als **Repräsentant** für den Test herangezogen werden – das können sein
 - Typische Werte (oft genutzte)
 - Problemwerte (vermutete Fehler)
 - Zufällig ermittelte Werte
- Die Repräsentanten **gültiger ÄK** dürfen **kombiniert** werden
- Die Repräsentanten **ungültiger ÄK** dürfen **nicht** miteinander **kombiniert** werden
- Die Repräsentanten **ungültiger ÄK** dürfen nur mit **gültigen** Repräsentanten anderer ÄK **kombiniert** werden (um mögliche **Fehlermaskierungen** zu vermeiden)
- Die **Repräsentanten ungültiger ÄK** sollen immer mit den selben gültigen Werten der restlichen ÄK (Standardkombinationen) zu Testfällen vereint werden

Äquivalenzklassenbildung – Fazit

- Systematische Testfallerstellung heißt, mit einer **minimalen Anzahl** an Testfällen eine hohe Fehlerentdeckungswahrscheinlichkeit erreichen
- Zerlegung in Äquivalenzklassen auf Basis der Definitionen / Spezifikationen erfasst sehr gut die **funktionalen Anforderungen**
- Eine Priorisierung der Äquivalenzklassen kann dazu dienen, **Testfälle zu priorisieren** (Eingaben, die selten gemacht werden, sind zuletzt zu testen)
- Der Test **ungültiger** Wertebereiche ist durch die **Negativtestfälle** sichergestellt
- Äquivalenzklassenbildung ist **auf jeder Teststufe** einsetzbar



Grenzwertanalyse

1 von 3

- **Grenzwertanalyse** (boundary value analysis)
Ein Black-Box-Testverfahren, bei dem die Testfälle unter Nutzung von Grenzwerten entworfen werden.

Die Grenzwertanalyse **erweitert die Äquivalenzklassenbildung** durch eine **Vorschrift zur Repräsentantenauswahl**

- **Randbereiche** der Äquivalenzklassen werden intensiver getestet
- Die Grenzwerte sind der größte und der kleinste Wert einer ÄK
- Zum Test verwendet man üblicherweise den **exakten Grenzwert** und die **beiden** jeweils **benachbarten** Werte

Anwendung der Grenzwertanalyse setzt voraus, dass

- die Äquivalenzklassen **numerische** Werte oder **Wertebereiche** umfassen und
- sich für die Wertebereiche **Grenzen** aus der Spezifikation identifizieren lassen



Grenzwertanalyse

2 von 3

Die Grenzwertanalyse kann auf **allen** Teststufen angewendet werden. Es ist eine **einfache** und **wirksame** Methode, Fehlerzustände zu finden

Grenzwertüberdeckung misst die Anzahl der getesteten Grenzwerte dividiert durch die Gesamtanzahl der identifizierten Grenzwerte

$$\text{GW-Überdeckung in \%} = \frac{\text{Anzahl getesteter GW}}{\text{Gesamtzahl GW}} \times 100$$

Grenzwertanalyse

3 von 3

Getestet wird üblicherweise mit **drei** Repräsentanten pro Grenze

Vorgehensweise
bei 3 Repräsentanten:

Wertebereich: Untergrenze δ x δ Obergrenze

Untergrenze – δ	Untergrenze	Untergrenze δ
Obergrenze – δ	Obergrenze	Obergrenze δ

δ ist dabei der kleinste per Definition mögliche Schritt
(z. B. 1 wenn ganzzahlige Werte definiert wurden)

Es gibt Fälle, da reichen zwei und andere, da werden sogar vier Repr. benötigt.

• Beispiel:

- Wertebereich für den Rabatt in %: $0,00 \leq x \leq 100,00$
- **Gültige ÄK:** $0,00 \leq x \leq 100,00$
- **Grenzwertanalyse** erweitert die Repräsentanten um:
-0,01; 0,00; 0,01; 99,99; 100,00; 100,01



Entscheidungstabellentests

1 von 9

- **Entscheidungstabellentest** (decision table testing)

Ein Black-Box-Testverfahren, bei dem Testfälle im Hinblick auf die Ausführung von Kombinationen der Bedingungen einer Entscheidungstabelle entworfen werden.
[Egler63]

Kombinatorische Verfahren testen Systemanforderungen, bei denen unterschiedliche Kombinationen von **Bedingungen** zu unterschiedlichen **Ergebnissen** führen

Abhängigkeiten der Eingabewerte werden bei den **bisher** besprochenen Verfahren **nicht berücksichtigt**

Die Verwendung aller möglichen Kombinationen der Eingabewerte als Testfälle ist aufgrund der großen Anzahl so entstehender Testfälle meist nicht praktikabel (siehe Begriffsdefinition „Vollständiger Test“ und Testfallexplosion in Kap. II 03)

Mittels **Ursache-Wirkungs-Graphen** und daraus abgeleiteten Entscheidungstabellen wird die große Anzahl möglicher Kombinationen auf eine **Untermenge** von Kombinationen systematisch reduziert

Entscheidungstabellentests

2 von 9

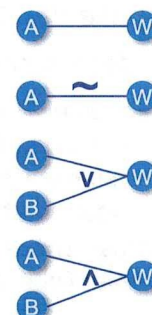
Ein Ursache-Wirkungs-Graph ist eine formale Sprache

Zur Erstellung eines U-W-G ist die (i. d. R. informelle) Spezifikation des Testobjektes in die formale Sprache zu übersetzen

Die Wirkungen, die am Testobjekt beobachtet werden können, sind auf Ursachen zurückzuführen, die auf das Testobjekt einwirken

Elemente eines Ursache-Wirkungs-Graphen:

- **Identität** (Wenn Ursache, dann Wirkung)
- **Negierung** (Wenn nicht Ursache, dann Wirkung)
- **Oder** (Wenn Ursache A oder Ursache B, dann Wirkung)
- **Und** (Wenn Ursache A und Ursache B, dann Wirkung)

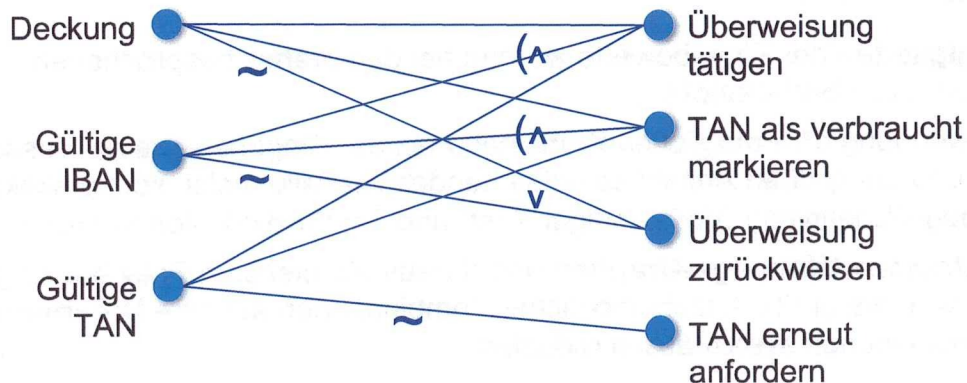


Entscheidungstabellentests

3 von 9

Beispiel 5: Online-Banking

Der Benutzer hat sich bereits am System mit seiner Kontonummer und PIN identifiziert. Ausreichende Deckung auf seinem Konto vorausgesetzt, kann er eine Überweisung tätigen. Dazu muss er eine gültige IBAN eingeben, ferner eine gültige TAN erfassen, die durch die Transaktion verbraucht wird



Entscheidungstabellentests – Beispiel

4 von 9

Bedingung		T-1	T-2	T-3	T-4	T-5
Bedingungen (Ursachen)	Deckung vorhanden	ja	ja	nein	ja	ja
	Gültige IBAN	ja	ja	--	nein	ja
	Gültige TAN	ja	ja	--	--	nein
Aktionen (Wirkungen)	Überweisung ausführen	ja	--	--	--	--
	TAN als verbraucht markieren	--	ja	--	--	--
	Überweisung zurückweisen	--	--	ja	ja	--
	TAN erneut anfordern	--	--	--	--	ja

Jede Spalte der Tabelle ergibt einen Testfall

Aufbau der Entscheidungstabelle:

- Auswahl einer Wirkung
- Durchlaufen des Graphen „nach vorn“, Ermittlung relevanter Ursachen
- Je gefundener Kombination von Ursachen wird eine Spalte (Testfall) in der Entscheidungstabelle eingerichtet
- Ggf. **zusammenführen** von identischen Ursachenkombinationen, die auf unterschiedliche Wirkungen führen (s. T1 und T2)

Entscheidungstabellentests – Beispiel

5 von 9

Bedingung		T-1	T-2	T-3	T-4	T-5
Bedingungen (Ursachen)	Deckung vorhanden	ja	ja	nein	ja	ja
	Gültige IBAN	ja	ja	--	nein	ja
	Gültige TAN	ja	ja	--	--	nein
Aktionen (Wirkungen)	Überweisung ausführen	ja	--	--	--	--
	TAN als verbraucht markieren	--	ja	--	--	--
	Überweisung zurückweisen	--	--	ja	ja	--
	TAN erneut anfordern	--	--	--	--	ja

Jede Spalte der Tabelle ergibt einen Testfall

Aufbau der Entscheidungstabelle:

- Auswahl einer Wirkung
- Durchlaufen des Graphen „nach vorn“, Ermittlung relevanter Ursachen
- Je gefundener Kombination von Ursachen wird eine Spalte (Testfall) in der Entscheidungstabelle eingerichtet
- Ggf. **zusammenführen** von identischen Ursachenkombinationen, die auf unterschiedliche Wirkungen führen (s. T1 und T2)

Übung IV.2: Entscheidungstabellentests

6 von 9

(20 Min. gemeinsame Erarbeitung)

Der Besuch im Zoo

Als Tierpfleger im Krefelder Zoo sind Sie für die Betreuung der Nashornbabys *Davu* und *Thabo* zuständig

Es gibt einige **Regeln** für die Besuche:

- Besucher mit **ansteckenden Krankheiten** dürfen gar nicht zu den Tieren
- Besucher **außerhalb der Besuchszeiten** dürfen nur für max. 15 Min. nach vorheriger Anmeldung zu den Tieren
- Besucher in **großen Gruppen** dürfen nur mit einem Pfleger zu den Tieren

Die **Regeln** sind der **Wichtigkeit** nach angeordnet. In Abhängigkeit von den **Ursachen** werden die (ggf. kombinierten) **Wirkungen** ermittelt

Entscheidungstabellentests

7 von 9

Einsatz in der Praxis

- Testfälle, die aus dem **Entscheidungstabellentest** resultieren, können mit Testfällen **kombiniert** werden, die aus einer separat durchgeführten **Grenzwertanalyse** stammen
- Je mehr Ursachen / Wirkungen berücksichtigt werden, desto **größer** wird die zugehörige Entscheidungstabelle und somit die Zahl der potenziellen Testfälle (2^n mögliche Testfälle bei n (atomaren) Bedingungen, die jeweils wahr oder falsch sein können). Ohne **Toolunterstützung** ist diese Methode für größere Systeme schwer handhabbar. Deswegen muss die Spezifikation in **überschaubare Stücke** aufgeteilt sein
- Durch Löschung von Kombinationen, die das Ergebnis nicht beeinflussen (z.B. **unmögliche Kombinationen**) kann die **Anzahl der Testfälle reduziert** werden
- Alle wichtigen Kombinationen von Bedingungen können identifiziert werden. Das hilft dabei, mögliche Lücken in den Anforderungen zu finden
- Entscheidungstabellen sind auf **jeder Teststufe** auf alle Situationen angewendet werden, Kombinationen von Bedingungen stehen im Fokus

Entscheidungstabellentests

8 von 9

• Vorteile

- **Systematische** Ermittlung von Eingabekonstellationen/Ursachenkombinationen, die mit anderen Testverfahren nicht gefunden werden
- **Einfache Herleitung** von Testfällen direkt aus der resultierenden Entscheidungstabelle
- **Überdeckung** wird gemessen, Anzahl der durchgeführten Tests in Relation zu der Gesamtanzahl der Überdeckung aller Spalten der resultierenden Entscheidungstabelle durch mindestens einen Testfall
- **Reduzierung der Testfallanzahl** durch systematisches **Zusammenschieben** von Spalten der Entscheidungstabelle möglich

• Nachteile

- Bei Berücksichtigung **vieler Ursachen** komplexe und **umfangreiche Resultate**, daher **hohes Fehlerpotenzial**
- Dies macht einen **Werkzeugeinsatz** erforderlich

Entscheidungstabellentests – Testüberdeckung

9 von 9

- Die minimale Testüberdeckung ist **ein Test pro Spalte**, unter der Voraussetzung, dass in der Spalte alle möglichen Kombinationen von Bedingungen berücksichtigt werden
- Durch die Berücksichtigung von Grenzbedingungen kann sich die Anzahl der Testfälle deutlich erhöhen
- Grenzwertanalyse und ÄK sind hierbei **ergänzende** Testverfahren
- Gemessen werden die überdeckten Kombinationen von Bedingungen in Relation zu der Gesamtanzahl der Kombinationen von Bedingungen

$$E\text{-Tabellenüberdeckung in \%} = \frac{\text{Überdeckte Kombinationen von Bedingungen}}{\text{Gesamtanzahl Kombination von Bedingungen}} \times 100$$



Zustandsübergangstest

1 von 6

- **Zustandsübergangstest** (state transition testing)
Ein Black-Box-Testverfahren, bei dem Testfälle aus Zustandsdiagrammen oder Zustandstabellen abgeleitet werden, um zu bewerten ob das Testelement gültige Zustandsübergänge erfolgreich ausführt und ungültige Übergänge verhindert.

Die drei **bisher** bekannten Verfahren berücksichtigen **nur** das **Ein- / Ausgabeverhalten** laut Spezifikationen

Verschiedene **Zustände**, die ein Testobjekt während seiner Ausführung annehmen kann, bleiben außer Betracht

- Haben bereits erfolgte Vorgänge Auswirkungen auf den weiteren Verlauf?

Die verschiedenen Zustände, die ein Testobjekt während seiner Ausführung einnehmen kann, werden in **Zustandsmodellen** abgebildet

Die Ermittlung von zustandsbezogenen Testfällen wird auch als **Zustandsübergangsanalyse** bezeichnet

Übung IV.3: Zustandsübergangstest

2 von 6

(10 Min. Gruppenarbeit)

- Ein Zustandsübergangsdiagramm für die möglichen unterschiedlichen Familienstände aus Sicht eines deutschen Standesamtes wird erarbeitet
- Die Umformung des Diagrammes in einen Zustandsbaum wird gemeinsam durchgeführt

Zustandsübergangstest

3 von 6

Erstellung der Testfälle erfolgt mittels **Übergangsbaumes** nach folgender Regel

- Der **Anfangszustand** ist die **Wurzel** des Baumes
- Für jeden möglichen Folgezustand wird jeweils ein Knoten gebildet und mit dem Vorzustand über eine Kante verbunden
- **Beendet** wird die weitere Verzweigung, wenn
 - der Zustand des Knotens ein **Endzustand** (d. h. Blatt) ist, oder
 - der Zustand des Knotens **bereits** im Baum **enthalten** ist

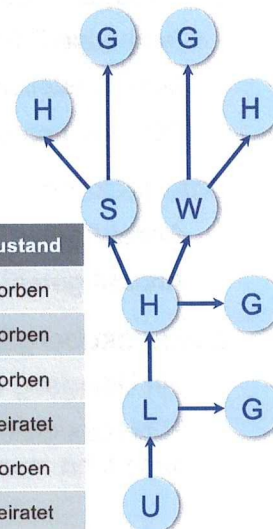


Zustandsübergangstest

4 von 6

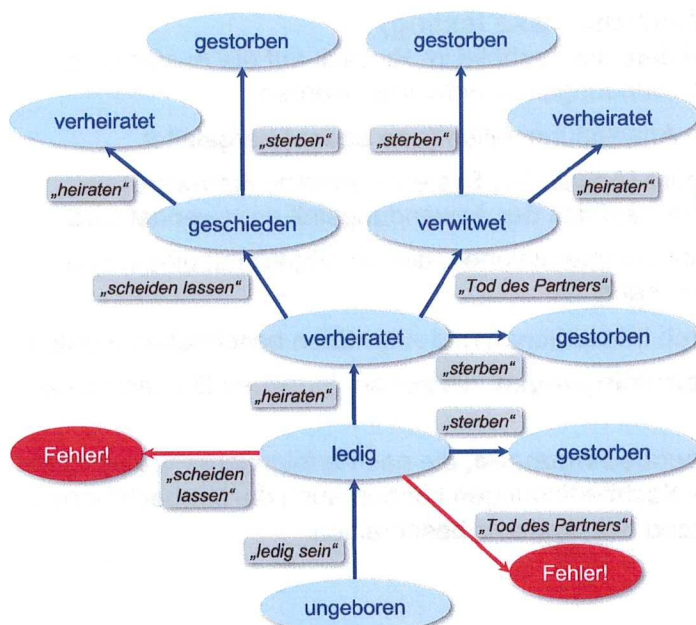
- Im **Zustandsbaum** stellt jeder mögliche Weg von der **Wurzel zu einem Blatt** einen Testfall des zustandsbezogenen Tests dar
- Aus dem Zustandsbaum für das Beispiel ergeben sich die **Testfälle** der abgebildeten Tabelle

TF	Zustand 1	Zustand 2	Zustand 3	Zustand 4	Zustand 5	Endzustand
t1	ungeboren	ledig	gestorben			gestorben
t2	ungeboren	ledig	verheiratet	gestorben		gestorben
t3	ungeboren	ledig	verheiratet	verwitwet	gestorben	gestorben
t4	ungeboren	ledig	verheiratet	verwitwet	verheiratet	verheiratet
t5	ungeboren	ledig	verheiratet	geschieden	gestorben	gestorben
t6	ungeboren	ledig	verheiratet	geschieden	verheiratet	verheiratet



Zustandsübergangstest – Zustandsübergangsbaum:

5 von 6



- Der Zustandsbaum kann um die ungültigen Funktionen erweitert werden (Test auf Robustheit des Testobjektes)
- Für den erweiterten Zustandsbaum lassen sich analog die Testfälle erstellen

Im Bild: zwei Beispiele für spezifikationswidrige Zustandsübergänge – es gibt noch mehr...

Nicht definierte (ungültige) Übergänge zwischen Zuständen sind nicht testbar

Zustandsübergangstest

6 von 6

- Zustandsübergangstest wird eingesetzt für
 - “embedded” Software und in der Automatisierungstechnik
 - Modellerstellung eines Geschäftsszenarios mit spezifischen Zuständen
 - Bildschirmnavigation
- Überdeckung
 - **Anzahl getesteter Zustände** in Relation zu der Gesamtanzahl der Zustände
 - **Anzahl getesteter Zustandsübergänge** in Relation zu allen Zustandsübergängen
- Zustandsübergänge
 - Ein Übergang wird durch ein Ereignis hervorgerufen (Benutzereingabe)
 - Führt das gleiche Ereignis zu zwei oder mehreren Übergängen aus einem Zustand, dann führt man dafür eine Schutzbedingung (guard condition) ein



Anwendungsfallbasierter Test

1 von 8

- **Anwendungsfallbasierter Test** (use case testing)
Ein Black-Box-Testverfahren, bei dem die Testfälle im Hinblick auf die Ausführung verschiedener Verhalten eines Anwendungsfalls entworfen werden
 - Die Testfälle werden aus den Anwendungsfällen (**use cases**) hergeleitet
 - Anwendungsfälle haben Akteure (Anwender, Systeme, externe Hardware) und Objekte (Komponente, System), auf das der Anwendungsfall angewendet wird
 - Ein Anwendungsfall beschreibt die Interaktionen, die ein **Objekt** mit einem oder mehreren **Akteuren** ausführen kann
 - Anwendungsfälle können durch Interaktionen und Aktivitäten beschrieben werden
 - Jeder Anwendungsfall hat **Vorbedingungen**, die zur erfolgreichen Durchführung erfüllt sein müssen
 - Jeder Anwendungsfall hat **Nachbedingungen**, die nach erfolgreicher vollständiger Durchführung herrschen. Die Nachbedingungen können auch die beobachtbaren Ergebnisse und den Endzustand des Systems beschreiben

Anwendungsfallbasierter Test

2 von 8

• Anwendungsfälle

- Es werden meistens ein Hauptszenario und alternative Szenarien oder Varianten dargestellt
- Anwendungsfälle können auf abstrakter Ebene, wie fachlicher Anwendungsvorfall, technologiefrei, Geschäftsprozessebene, aber auch auf Systemebene beschrieben werden, die die Systemfunktionalität als Systemanwendungsfall beinhaltet
- Ein Anwendungsfall besteht aus mehreren möglichen Varianten seines grundlegenden Verhaltens
- Sonder- und Fehlerbehandlung (Wiederherstellungsverfahren, alles, was zu Fehlermeldungen führt) werden beschrieben
- Die Tests sollen das definierte Verhalten nachweisen

• Überdeckung

- Anteil der getesteten Anwendungsvarianten in Relation zu der Gesamtanzahl der ermittelten Anwendungsvarianten in Prozent

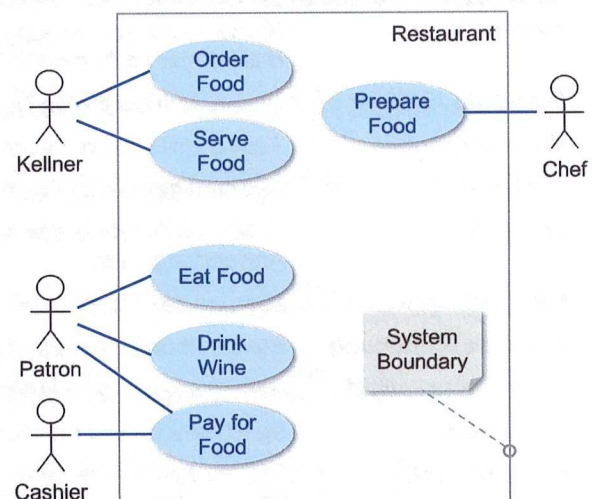
Anwendungsfallbasierter Test – Beispiel

3 von 8

UML – Use-Case-Diagramm (Anwendungsfall-Diagramm)

UML steht für *Unified Modeling Language*, einem weltweiten Industriestandard. Verantwortlich für die Sprachdefinition zeichnet sich das Softwarekonsortium *Object Management Group (OMG)*.

Sie entstand 1995 als Synthese einzelner Notationen der drei Amigos, Grady Booch (Firma Rational), James Rumbaugh und Ivar Jacobson. Seit 1997 liegt die Verantwortung für die UML bei der OMG.



Anwendungsfallbasierter Test

4 von 8

- Jeder Anwendungsfall beschreibt eine bestimmte Aufgabe (Nutzer-System-Interaktion), die mit dem System bearbeitet wird
- Beschreibungselemente eines Anwendungsfalles sind u. a.
 - Vorbedingungen
 - Vorausgesagte Ergebnisse / Verhaltensweisen des Systems
 - Nachbedingungen
- Dieselben Beschreibungselemente werden **auch für Testfälle** benötigt!
- Aus jedem **Anwendungsfall** lässt sich mindestens ein **Testfall** konstruieren
- Die typische Diagrammnotation ist für eine direkte Testfallermittlung **zu wenig konkret**. Eingabewerte und vorausgesagte Ergebnisse für die Testfälle müssen im Detail festgelegt werden
- Für jede im Diagramm enthaltene Alternative ist ein eigener Testfall vorzusehen

Anwendungsfallbasierter Test – Beispiel: Use-Case-Spez.

5 von 8

Strukturelement	Erläuterung
Bezeichnung	Name, Kurzbeschreibung, ID, ...; Bezeichner sollte ein VERB enthalten und eindeutig sein
Beteiligte Akteure	Die an dem konkreten Use Case beteiligten Akteure
Vorbedingungen	Ausgangssituation, bevor der Anwendungsfall eintritt
Auslöser	Wie wird der Ablauf des Anwendungsfalls gestartet?
Ergebnis	Prüfbares Ergebnis des Anwendungsfalls (Zielstellung des Anwendungsfalls)
Nachbedingung	Zustand des Systems, nach dem der Anwendungsfall ausgeführt wurde
Ablaufbeschreibung	Beschreibung des Ablaufs, ggf. nummeriert
Alternativer Ablauf	Beschreibung von Ausnahmen zum 'normalen' Ablauf
Ausnahmen	Beschreibung von möglichen Ausnahme- und Fehlersituationen
Dialogbeispiele	GUI-Mocks, Maskenentwürfe können die Abläufe für den Leser nachvollziehbarer machen

Anwendungsfallbasierter Test – Beispiel

6 von 8

Attribut	Beschreibung
Name des Anwendungsfalls	Geld mit/ohne Beleg beziehen
Kurzbeschreibung	Der Kunde kann einen frei wählbaren Betrag (im Rahmen seines Limits) in EUR mit oder ohne Ausdruck eines Belegs beziehen.
Hauptszenario	1. Der Kunde wählt die Dienstleistung 'Barbezug EUR'. 2. Je nach Tages(rest)limit und Noten-Verfügbarkeit werden dem Kunden verschiedene, vordefinierte Beträge und die Option, einen anderen Betrag einzugeben, angeboten. 3. Der eingegebene Betrag wird vom System autorisiert. 4. Verbuchung der Transaktion im System. 5. Der Kunde kann die Notenstückelung wählen, falls vorhanden. 6. Der Kunde wird gefragt, ob er einen Beleg wünscht. 7. Wünscht der Kunde einen Beleg, wird er gebeten, den Beleg und anschließend die Karte und den entsprechenden Betrag zu entnehmen. Wünscht der Kunde keinen Beleg wird er gebeten, die Karte und anschließend den entsprechenden Betrag zu entnehmen. 8. Es erscheint das 'Danke für Ihren Besuch' Fenster und anschließend das Grundbild. Die Interaktion ist beendet.
Varianten	3a. Der gewünschte Betrag kann nicht ausgezahlt werden, da das Tages(rest)limit überschritten ist. Der Kunde kann einen anderen Betrag wählen. 4a. Bei Überziehung des Kontos (d.h. Bezugsbetrag ist grösser als 'Aktueller Saldo') erfolgt eine Warnmeldung. Bei Abbruch aufgrund der Warnmeldung gelangt der Kunde zurück zum Fenster Betragswahl.
Vorbedingung	Der Kunde wurde erfolgreich authentifiziert (Anwendungsfall "Karte und PIN eingeben").
Nachbedingung	Der Kunde hat den gewählten Betrag in EUR und auf Wunsch einen Beleg erhalten. Die Karte wurde dem Kunden zurückgegeben.

Anwendungsfallbasierter Test – Beispiel

7 von 8

Testfall	Beschreibung	Anwendungsfall	Ziel	Vorbedingungen / Eingabedaten	Erwartete Ergebnisse
001	Hauptszenario – ohne Beleg	Geld mit/ohne Beleg beziehen	Gewählter Betrag ausbezahlt	Tages(rest)limit: 1000 Aktueller Saldo: 2000 Betrag: 500 Notenstückelung: 5 x 100 Beleg: nein	erfolgreich
002	Hauptszenario – mit Beleg	Geld mit/ohne Beleg beziehen	Gewählter Betrag ausbezahlt – Beleg gedruckt	Tages(rest)limit: 1000 Aktueller Saldo: 2000 Betrag: 500 Notenstückelung: 5 x 100 Beleg: ja	erfolgreich
003	Alternative 3a: Tages(rest)limit überzogen	Geld mit/ohne Beleg beziehen	Warnmeldung "Tageslimit überzogen"	Tages(rest)limit: 400 Aktueller Saldo: 2000 Betrag: 500	Kein Bezug, zurück zu Betragswahl
004	Alternative 4a: Überziehung des Kontos	Geld mit/ohne Beleg beziehen	Warnmeldung "Kontoüberzug"	Tages(rest)limit: 1000 Aktueller Saldo: 400 Betrag: 500	Kein Bezug, zurück zu Betragswahl

Anwendungsfallbasierter Test

8. von 8

- **Fazit**
 - Anwendungsfälle (**Szenarien**) beschreiben die „Prozessabläufe“ durch das System auf Grundlage seiner tatsächlichen Verwendung. Daraus abgeleitete Testfälle decken Fehlerzustände in den Prozessabläufen auf. Das Zusammenwirken und die gegenseitige Beeinflussung unterschiedlicher Komponenten kann im Integrationstest getestet werden
- **Vorteile**
 - Gut geeignet für **Abnahmetests und Systemtests**, da die Anwendungsfälle die Einsatzszenarien der Anwender gut wiedergeben
 - Gut geeignet, wenn die **Spezifikation** des Systems bereits mit **UML** oder Ablaufbeschreibungen vorgenommen wurde
- **Nachteile**
 - Keine Herleitung zusätzlicher Testfälle über die dargestellten Anwendungsfälle hinaus
 - Daher **Ergänzung** mit anderen Testverfahren notwendig

Black-Box-Testverfahren – Fazit

- Prüfung der **Funktionalität** als Ziel der **Black-Box-Testverfahren**, die Analyse erfolgt auf Basis der Anforderungen / Spezifikationen
 - Dabei ist das Testergebnis von der Qualität der Spezifikationen abhängig (z. B. fehlerhafte / unvollständige Spezifikationen)
 - Liegen in der Spezifikation Fehlerzustände vor, wird auch fehlerhaft getestet – getestet wird die beschriebene Funktionalität
 - Enthält das Testobjekt **Funktionen**, die nicht spezifiziert wurden, bleiben diese **unentdeckt**
 - Problematisch werden solche nicht geforderten Eigenschaften dann, wenn sie zu Stabilitäts- oder Sicherheitsproblemen führen können, z. B. Steuerung eines **Geldautomaten**
- **Korrekte Funktionalität** hat einen **hohen Stellenwert** unter den Qualitätsmerkmalen, daher haben sie eine hohe Priorität
- Die **interne** Struktur wird nicht betrachtet

Zusammenfassung

- Black-Box-Testverfahren
 - Äquivalenzklassenbildung
 - Grenzwertanalyse
 - Ursache-Wirkungs-Graphen und Entscheidungstabellentests
 - Zustandsübergangstest
 - Anwendungsfallbasierter Test (Systemtest, Abnahmetest, Integrationstest)
- Getestet wird bei Black-Box-Testverfahren (nur) gegen die **Spezifikation**, d.h. was nicht spezifiziert ist, wird nicht getestet
- **Überflüssiger Code** im System wird mit Black-Box-Testverfahren nicht gefunden

Kapitel IV – Testverfahren

- IV/1 Kategorien von Testverfahren
- IV/2 Black-Box-Testverfahren
- IV/3 White-Box-Testverfahren
- IV/4 Erfahrungsbasierte Testverfahren

Allgemeines

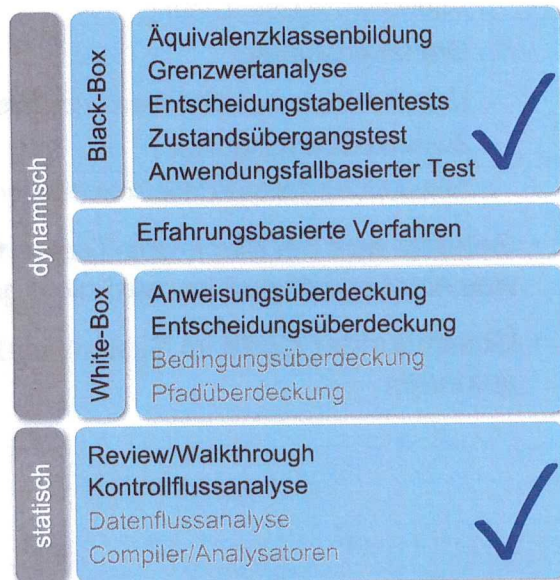
Dynamische Testverfahren werden in **zwei Kategorien** unterteilt

- Black-Box-Testverfahren testen nur, was spezifiziert ist, da Testfälle aus der Spezifikation hergeleitet werden



- White-Box-Testverfahren testen, was da ist, da die Überdeckung des vorhandenen Codes gemessen wird (Codeüberdeckung)

Qualitätssteuerung



Unterschiedliche White-Box-Testverfahren

Strukturorientierter Test betrachtet die **interne Struktur** des Testobjekts

- Komponentenebene: codebasiert, die Struktur der Softwarekomponente
- Integrationsebene: Schnittstellen, die Struktur des Aufrufgraphen
- Systemebene: die Struktur des Menüs, der Geschäftsprozesse oder auch der Websites

Folgende White-Box-Testverfahren werden nun vertieft

- **Anweisungstest und -überdeckung**
- **Entscheidungstest und -überdeckung**
- (Pfadüberdeckung)
- (Bedingungsüberdeckung)

White-Box-Testverfahren – Werkzeuge

1 von 2

Bei den kontrollflussorientierten Verfahren geht es – wie bei den Black-Box Testverfahren – darum, **Programmteile auszuführen**

- Theoretisch sollten **alle Teile des Programms** während der Tests **mindestens einmal ausgeführt** worden sein

Die Ausführung wird anhand von **Werkzeugen** (z. B. mit Hilfe von **Überdeckungsanalysatoren**) nachvollzogen

- Es wird eine **Instrumentierung** durchgeführt – d. h. es werden Zähler an entsprechenden Stellen des Testobjektes installiert
- Diese **Zähler** stehen **zu Beginn** der Programmausführung **auf Null** und werden mit jeder Ausführung des Programmteils stufenweise **erhöht**
- Bleiben Zähler am Testende **auf Null** bedeutet dies, dass der Programmteil **nicht zur Ausführung** gekommen ist

White-Box-Testverfahren – Werkzeuge

2 von 2

White-Box-Testverfahren benötigen in vielen Bereichen eine Unterstützung durch Werkzeuge – z. B. bei der

- Testfallerstellung
 - Z. B. Ableitung von Kontrollflussgraphen aus dem Programmcode
- Testdurchführung
 - Tools zur Überwachung und Steuerung der Abläufe innerhalb des Testobjektes

Der **Werkzeugeinsatz** stellt an dieser Stelle die Testqualität sicher und steigert die Effizienz

- Aufgrund der Komplexität der Vorgänge ist eine manuelle Ausführung
 - Zeit- und ressourcenverschlingend
 - Fehleranfällig und nur schwer durchführbar



Überdeckungsarten

- **Anweisungsüberdeckung** (statement coverage)

Der Anteil der Anweisungen, die durch eine Testsuite ausgeführt wurden, bezogen auf alle Anweisungen

- **Entscheidungsüberdeckung** (decision coverage)

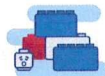
Der Anteil an Entscheidungsausgängen, die durch eine Testsuite geprüft wurden. 100% Entscheidungsüberdeckung schließt sowohl 100% Zweigüberdeckung als auch 100% Anweisungsüberdeckung ein

- **Pfadüberdeckung** (path coverage)

Der Anteil der vollständigen Pfade, die durch eine Testsuite ausgeführt wurden

- **Bedingungsüberdeckung** (condition coverage)

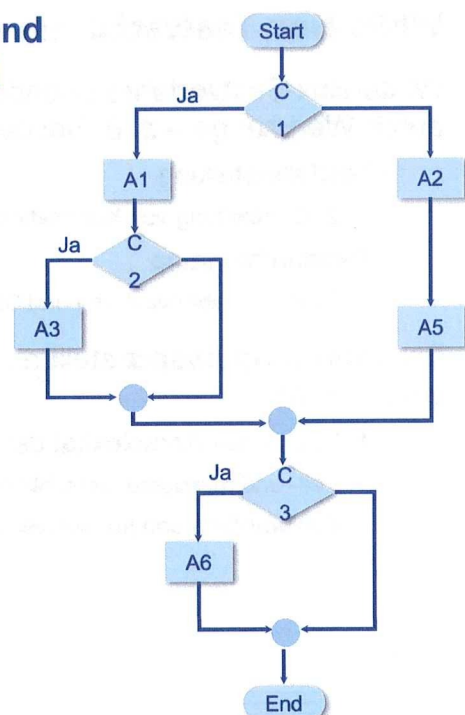
Der Anteil der Teilbedingungsergebnisse, die durch eine Testsuite ausgeführt worden sind. 100% Bedingungsüberdeckung bedeutet, dass jede atomare Teilbedingung in jeder Entscheidung mindestens einmal mit True und einmal mit False ausgeführt wurde



Übung IV.4: Anweisungsüberdeckung und Entscheidungsüberdeckung / Beispiel 1

(10 Min. gemeinsame Erarbeitung am Flipchart)

```
Start
If C1 > 0 then
  do A1
  if C2 < 5 then
    do A3
  endif
else
  do A2
  do A5
endif
If C3 = C1 then
  do A6
endif
End
```



Anweisungstest und -überdeckung (statement coverage)

- Basis der Untersuchungen ist der **Kontrollflussgraph**
 - Alle Anweisungen werden als **Knoten** und der Kontrollfluss zwischen den Anweisungen wird in Form der **Kanten** dargestellt
 - Mehrere aufeinander folgende Anweisungen sind in einem Knoten zusammengefasst, da sie direkt nacheinander ablaufen
- Die einzelnen Anweisungen eines Programms bzw. deren Ausführung stehen im Mittelpunkt der Betrachtung
- Ziel des Tests ist die Ausführung eines vorher bestimmten Anteiles aller Anweisungen (Anweisungs-Überdeckungsmaß oder C_0 Maß)

$$C_0 \text{ (in \%)} = \frac{\text{Anzahl ausgeführter Anweisungen}}{\text{Gesamtzahl Anweisungen}} \times 100$$

Entscheidungstest und -überdeckung (decision coverage)*

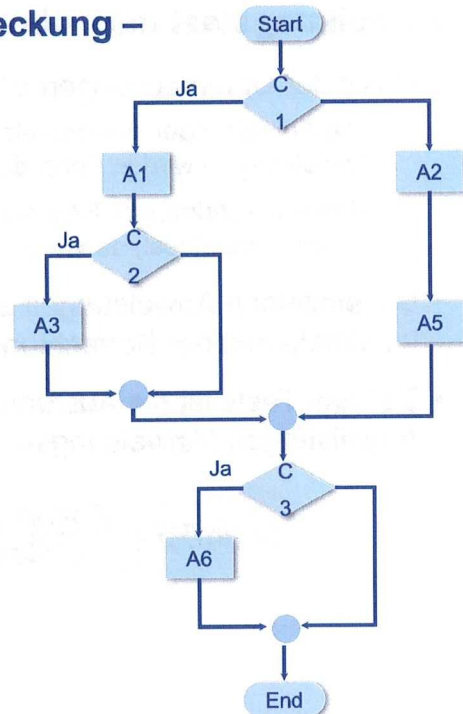
- Statt der Anweisungen steht der Kontrollfluss im Mittelpunkt der Auswertung (**nicht nur Knoten, sondern auch Kanten**)
 - Welche Testfälle sind nötig, um alle Kanten zu durchlaufen?
- Ziel des Tests ist die Ausführung eines vorher bestimmten Anteiles aller Entscheidungen (Entscheidungsüberdeckungsmaß oder C_1 Maß)

$$C_1 \text{ (in \%)} = \frac{\text{Anzahl ausgeführter Entscheidungen}}{\text{Gesamtzahl Entscheidungen}} \times 100$$

* Ähnlich: Zweigüberdeckung (engl. branch coverage)

Anweisungs- und Entscheidungsüberdeckung – Beispiel

- Die Wege durch alle Anweisungen des Kontrollflusses wären:
 - Start, C1, A1, C2, A3, C3, A6, End
 - Start, C1, A2, A5, C3, A6, End
- Die Wege durch alle Entscheidungen des Kontrollflusses wären:
 - Start, C1, A1, C2, A3, C3, A6, End
 - Start, C1, A1, C2, C3, A6, End
 - Start, C1, A2, A5, C3, End



Beitrag von Anweisungs- und Entscheidungstests

- Die Entscheidungsüberdeckung führt i. d. R. zu mehr Testfällen als die Anweisungsüberdeckung
 - Eine 100%ige Anweisungsüberdeckung ist in der 100%igen Entscheidungsüberdeckung enthalten!
 - Einige Kanten können mehrfach durchlaufen werden
- Schleifen werden einmal durchlaufen und führen zu **keinem zusätzlichen** Testfall
- Fehlerzustände im Code
 - Anweisungsüberdeckung kann Fehlerzustände finden, die mit anderen Tests nicht gefunden werden
 - Entscheidungsüberdeckung kann Fehlerzustände finden, deren Ausgänge (wahr / falsch) mit anderen Tests nicht ausgeführt werden

Übung IV.5, 6, 7: Abdeckungsmessung

(10 Min. Einzelarbeit / 15 Min. Ergebnisdiskussion)

Aus gegebenem Kontrollflussgraphen bzw. Pseudo Code sind:

- Die minimale Anzahl Testfälle für 100% Überdeckung zu bestimmen
 - Anweisungsüberdeckung
 - Entscheidungsüberdeckung
 - Pfadüberdeckung

288

Überdeckungsmessungen – Übersichtstabelle

Methode	Maß	erfüllte Bedingung	Durchführbarkeit
Anweisungsüberdeckung	C ₀	jede Anweisung wird mind. einmal ausgeführt	relativ einfach
Zweig-/Entscheidungsüberdeckung	C ₁	jede Kante wird mind. einmal durchlaufen	vertretbarer Aufwand
Pfadüberdeckung vollständig	C _{2a}	alle möglichen Pfade werden durchlaufen	unmöglich bei Schleifen
Einfachbedingung	C _{3a}	jede atomare Bedingung einmal true und false	einfach
Mehrfachbedingung	C _{3b}	jede true / false Bedingung der atomaren Bedingung wird getestet	sehr hoher Aufwand
Definierter Bedingungstest	C _{3c}	jede atomare und jede Gesamtbedingung wird einmal true und false getestet	hoher Aufwand

Zusammenfassung:

- Strukturorientierte oder White-Box-Testverfahren messen
 - Anweisungsüberdeckung
 - Entscheidungsüberdeckung
- Der Einsatz von Werkzeug gestützter **Überdeckungsmessung** findet **vornehmlich** auf den unteren Entwicklungsebenen wie **Komponenten-** und **Integrationstest** statt
- Das Konzept der Überdeckung lässt sich auch auf anderen Teststufen anwenden
 - **Integrationstest** – Anteil von Modulen, Komponenten oder Klassen, die durch eine Testsuite ausgeführt wurden
 - **Systemtest** – Überdeckung der Webseiten-Navigation, von Menüpunkten oder Funktionen
 - **Abnahmetest** – Überdeckung der Abläufe der Geschäftsprozesse

Kapitel IV – Testverfahren

- IV/1 Kategorien von Testverfahren
- IV/2 Black-Box-Testverfahren
- IV/3 White-Box-Testverfahren
- IV/4 Erfahrungsbasierte Testverfahren

Allgemeines

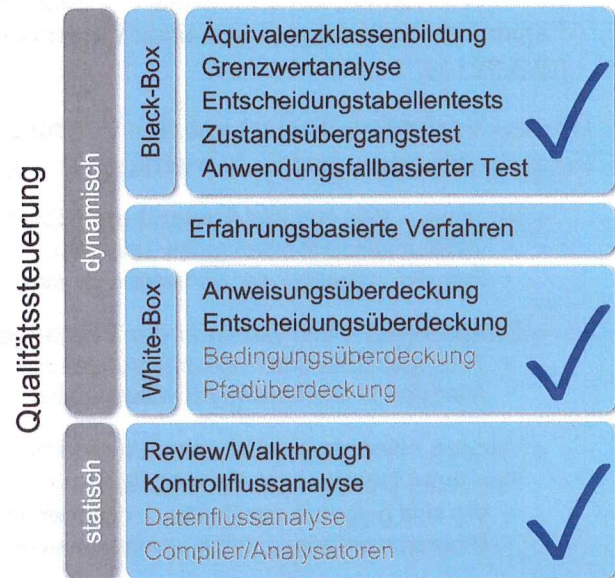
Systematische und erfahrungsbasierte Verfahren ergänzen sich

- Testfallermittlung **ohne systematische** Vorgehensweise auf Basis individueller Erfahrungen des Testers

Testfälle beruhen auf Intuition und Erfahrung

- Wo sind in der Vergangenheit Fehlerzustände aufgetreten?
- Wo könnten weitere Fehlerzustände auftreten?

Überdeckung schwer oder gar nicht messbar



Grundlagen

Bei der Anwendung von erfahrungsbasierten Testverfahren werden die Testfälle aus den Kenntnissen und der Intuition des Testers heraus entwickelt, sowie aus seiner Erfahrung mit ähnlichen Anwendungen und Technologien

Diese Verfahren können Testfälle identifizieren, die durch andere systematische Verfahren nicht so leicht zu identifizieren wären. Wird meist zur Ergänzung bereits vorhandener Testfälle genutzt

- Nicht ausreichend für das systematische Testen
- Ergibt aber oft zusätzliche Testfälle, die nicht über andere Verfahren zu erstellen sind, z. B.
 - Eingabe eines Schaltjahres nach 2060 (hier gab es früher mal Probleme)
 - Leere Menge in Eingabewerten (hier hatte ein ähnliches Programm einen Fehler)

Intuitive Testfallermittlung

1 von 2

- **Intuitive Testfallermittlung** (error guessing)

Ein Testverfahren, bei dem Tests auf Basis des Wissens der Tester über frühere Fehlerwirkungen oder allgemeines Wissen über Fehlerauswirkungen abgeleitet werden.
[ISO 29119]

Dieses Verfahren basiert auf der Vermutung vom Auftreten von Fehlhandlungen, Fehlerzuständen und Fehlerwirkungen, das in Testfälle umgesetzt wird

- Intuition – Wie hat die Anwendung früher funktioniert?
 - Intuition zeichnet einen guten Tester aus
 - Evtl. ein Vorgänger der Software oder ein in den Funktionen entsprechendes System
- Erfahrungen – Wo waren in der Vergangenheit oft Fehlerwirkungen?
 - Testergebnisse und Praxiserfahrungen mit ähnlichen Systemen
 - Alternativ können Listen mit wiederkehrenden Fehlerwirkungen herangezogen werden
- Wissen / Kenntnis – Wo speziell sind hier Fehlerzustände zu erwarten?
Spezielle Details zum Projekt fließen ein, z.B. Entwicklungsschwierigkeiten
 - Wo sind aufgrund des Zeitdruck oder der Komplexität erhöhte Fehlerzahlen zu erwarten?
 - Programmerstellung durch unerfahrenen Mitarbeiter

Intuitive Testfallermittlung – methodischer Ansatz

2 von 2

- Erstellen von Listen*, aus denen Testfälle entworfen werden
 - Mögliche / bekannte Fehlhandlungen → Fehlhandlungslisten
 - Mögliche / bekannte Fehlerzustände → Fehlerzustandslisten
 - Mögliche / bekannte Fehlerwirkungen → Fehlerwirkungslisten
- Mögliche Quellen
 - Erfahrungen über Fehlerzustände
 - Allgemeine Kenntnisse über ähnliche Systeme
 - Allgemeine Kenntnisse, warum Software fehlschlägt
 - Bekannte, in der Vergangenheit aufgetretene Fehlerwirkungen
 - Austausch von Erfahrungen hinsichtlich der Software in Anwenderkreisen

* Intuitive Testfallspezifikationen



Exploratives Testen

1 von 2

- **Exploratives Testen** (exploratory testing)
Ein Testansatz bei dem die Tester, basierend auf ihrem Wissen, der Erkundung des Testelements und dem Ergebnis früherer Tests, dynamisch Tests entwerfen und durchführen. (ISO 29119)

Vorgehensweise

- Erforschung der einzelnen Teile des Testobjektes anhand einer **Test-Charta**
- Ausführen weniger Testfälle, die sich auf die zu testenden Teile beschränken, dabei Einsatz von Error Guessing
- Analyse der Testergebnisse, dabei Erarbeitung eines groben Modells der Wirkungsweise des Testobjekts
- **Iteration**: Entwurf neuer Testfälle, dabei Nutzung des soeben erworbenen zusätzlichen Wissens um das Testobjekt
- Konzentration auf Auffälligkeiten bei der Durchführung der Testfälle sowie auf Erkundung weiterer Eigenschaften des Testobjekts
- **Mitschnittwerkzeuge** zur Protokollierung der Testaktivitäten nützlich

Exploratives Testen

2 von 2

Iteratives Vorgehen

- Kick-Off
 - Festlegen der neuen Testziele
- Vorbereitung
 - Prüfung der Rahmenbedingungen (HW, SW, etc.)
- Durchführung
 - "Entdecken" und "Erforschen" des Testobjektes während einer begrenzten Dauer (0.5 bis 2 Stunden)
- Abschluss
 - Sammeln und Dokumentieren der Resultate
- Review und Auswertung
 - Präsentieren der Resultate im Testteam

Ständiger Lernprozess

- Lerne das Produkt kennen
- Lerne, wie die Fehlerzustände in einer Software auftreten, und was die Fehlerzustände bewirken
- Lerne die Schwächen der Software kennen
- Lernen, wie die Software zu testen ist
- Teste die Software
- Melde die Probleme
- Erstelle neue Testfälle basierend auf dem, was du gerade gelernt hast



Checklistenbasiertes Testen

- **Checklistenbasiertes Testen** (checklist-based testing)

Ein erfahrungsbasiertes Testentwurfsverfahren, bei dem der erfahrene Tester eine **Liste von Kontrollpunkten** nutzt, die beachtet, überprüft oder in Erinnerung gerufen werden müssen, oder eine Menge von Regeln oder Kriterien gegen die ein Produkt verifiziert werden muss.

Vorgehensweise

- Abdeckung aller Testbedingungen aus einer Checkliste
- Tester erstellen neue Checklisten als Teil der Analyse oder erweitern bestehende
- Erstellung der Checklisten auf Basis von Erfahrungen; Wissen über das Wesentliche; Verständnis, warum und wie Software fehlschlägt

Checklisten

- Dienen zur Unterstützung funktionaler und nicht-funktionaler Testarten
- Sind eine gute Hilfestellung, um Konsens zu erzielen
- Lassen eine große Variabilität bei der Durchführung zu
- Liefern potentiell eine größere Überdeckung, aber geringe Wiederholbarkeit

Vergleich erfahrungsbasierte / systematische Verfahren

Erfahrungsbasierte Testfallermittlung ist eine sinnvolle Ergänzung zu den systematischen Verfahren

- Sie muss aber eine Ergänzung bleiben
- Sie kann nicht anhand von Kriterien auf Vollständigkeit geprüft werden – die Anzahl erfahrungsbasiert ermittelter Testfälle kann stark variieren

Der Testablauf entspricht dem Ablauf für systematisch erstellte Testfälle

- Es unterscheidet sich nur der Weg, auf dem der Testfall ermittelt / festgesetzt wurde

Über das erfahrungsbasierte Testen lassen sich häufig **zusätzliche** Fehlerzustände aufdecken, die man über das systematische Testen nicht gefunden hätte

Die **Dokumentation** des Vorgehens ist sehr wichtig, um die Testfälle **wiederverwendbar** zu machen für nachfolgende Releasezyklen und um ggf. einen Fehlernachtest durchführen zu können

Zusammenfassung

- Erfahrungsbasierte Verfahren können systematische Verfahren zur **Testfallermittlung ergänzen**
- Basieren stark auf den **individuellen Fähigkeiten** des jeweiligen Testers
- **Error Guessing**, fehlerbasierte intuitive Testfallermittlung
- **Exploratives Testen** auf Grundlage einer **Test-Charta**
- **Checklistenbasiertes Testen** auf Grundlage von Listen mit Kontrollpunkten

Schlüsselbegriffe

- Anweisungsüberdeckung
- Anwendungsfallbasierter Test
- Äquivalenzklassenbildung
- Black-Box-Testverfahren
- Checklistenbasiertes Testen
- Entscheidungstabellentests
- Entscheidungsüberdeckung
- Erfahrungsbasierte Testverfahren
- Exploratives Testen
- Grenzwertanalyse
- Intuitive Testfallermittlung
- Testverfahren
- Überdeckung
- White-Box-Testverfahren
- Zustandsübergangstest

V. Testmanagement » Agenda



Kapitel V – Testmanagement

- V/1 Testorganisation
- V/2 Testplanung und -schätzung
- V/3 Testüberwachung und -steuerung
- V/4 Konfigurationsmanagement
- V/5 Risiken und Testen
- V/6 Fehlermanagement

Lernziele für das Testmanagement

1 von 2

5.1 Testorganisation

- FL-5.1.1 (K2) Vor- und Nachteile unabhängigen Testens erklären können
- FL-5.1.2 (K1) Die Aufgaben eines Testmanagers und eines Testers benennen können

5.2 Testplanung und -schätzung

- FL-5.2.1 (K2) Den Zweck und Inhalt eines Testkonzepts zusammenfassen können
- FL-5.2.2 (K2) Zwischen verschiedenen Teststrategien unterscheiden können
- FL-5.2.3 (K2) Beispiele für mögliche Eingangs- und Endekriterien geben können
- FL-5.2.4 (K3) Wissen über Priorisierung sowie technische und logische Abhängigkeiten anwenden können, um die Testdurchführung für ein gegebenes Testfallset zu planen
- FL-5.2.5 (K1) Faktoren benennen können, die den Testaufwand beeinflussen
- FL-5.2.6 (K2) Den Unterschied zwischen zwei Schätzverfahren erklären können: das metrikbasierte Verfahren und das expertenbasierte Verfahren

Lernziele für das Testmanagement

2 von 2

5.3 Testüberwachung und -steuerung

- FL-5.3.1 (K1) Testmetriken wiedergeben können
- FL-5.3.2 (K2) Zweck, Inhalte und Zielgruppen für Testberichte zusammenfassen können

5.4 Konfigurationsmanagement

- FL-5.4.1 (K2) Zusammenfassen können, wie Konfigurationsmanagement das Testen unterstützt

5.5 Risiken und Testen

- FL-5.5.1 (K1) Risikostufe anhand der Wahrscheinlichkeit (des Eintritts) und Auswirkung (im Schadensfall) definieren können
- FL-5.5.2 (K2) Zwischen Projekt- und Produktrisiken unterscheiden können
- FL-5.5.3 (K2) Anhand von Beispielen beschreiben können, wie die Produktrisikanalyse Intensität und Umfang des Testens beeinflussen kann

5.6 Fehlermanagement

- FL-5.6.1 (K3) Einen Fehlerbericht schreiben können, der einen während des Testens gefundenen Fehler enthält

Kapitel V – Testmanagement

- V/1 Testorganisation
- V/2 Testplanung und -schätzung
- V/3 Testüberwachung und -steuerung
- V/4 Konfigurationsmanagement
- V/5 Risiken und Testen
- V/6 Fehlermanagement

Testmanagement ist Teil des Testprozesses

- **Testmanagement ist Projektmanagement** von Testprojekten
- Testen **beginnt** bereits mit der Erstellung der ersten **Dokumente** (in Form von Reviews)
- In jeder Entwicklungsphase liegen **unterschiedliche Testobjekte** vor, für die ebenfalls **unterschiedliche Testverfahren** eingesetzt werden
- Für jede dieser Teststufen müssen Tester bzw. **Testteams** zusammengestellt werden
 - Dabei sind zwischen Kosten und Nutzen abzuwägen und
 - Testteams auf Basis der Aufgaben auszuwählen

Unabhängiges Testen

1 von 3

Testaufgaben können von Personen in **verschiedenen Rollen** durchgeführt werden (von spezifischer Testrolle bis zur Rolle des Kunden / Anwenders). Hierbei erhöht die Unabhängigkeit des Testers die Effektivität des Testens

Vorteile der Unabhängigkeit von Testern (vom "Autoren" z.B. Entwicklern)

- Unabhängige Tester sind **unvoreingenommen**, erkennen andere Fehlerwirkungen als Entwickler, haben einen anderen Hintergrund und andere technische Sichtweisen
- Unabhängige Tester können die **Testbasis verifizieren, hinterfragen oder widerlegen**, auf der die Testobjekte hergestellt wurden. Sie können auch ohne (politischen) Druck objektiv, offen und ehrlich Probleme adressieren

Nachteile der Unabhängigkeit von Testern (vom "Autoren" z.B. Entwicklern)

- Isolation des Entwicklungsteams, verzögerte Rückmeldungen, feindliches Verhalten
- Unabhängige Tester werden verantwortlich für Engpässe und Verzögerungen von Releases gemacht
- Wichtige Informationen werden nicht an das Testteam weitergegeben
- Das Verantwortungsbewusstsein der Entwickler für Qualität geht verloren

Unabhängiges Testen – Grade der Unabhängigkeit

2 von 3

- Keine unabhängigen Tester
 - Entwickler testen ihren eigenen Code
- Unabhängige Entwickler oder Tester innerhalb des Entwicklungsteams
 - Entwickler testen die Arbeitsergebnisse ihrer Kollegen
- Unabhängiges Testteam innerhalb des Unternehmens
 - Eine Gruppe, die testet und an das Projektmanagement berichtet
- Unabhängige Tester innerhalb des Unternehmens aus dem Fachbereich
 - Getestet wird in der Linien- oder Fachabteilung des Auftraggebers
 - Spezialisten testen bestimmte Testarten (Gebrauchstauglichkeit, IT-Sicherheit, Performance, Regulatorik, Übertragbarkeit)
- Unabhängige Tester außerhalb des Unternehmens
 - Testen vor Ort (inhouse)
 - Testen außerhalb des Betriebes (outsourcing)

Unabhängiges Testen – Zusammenstellung des Testteams

3 von 3

- Je nach Teststufe, Projektumfeld, Testsituation sind verschiedene **Kombinationen** der oben erwähnten Testteam-Zusammenstellungen sinnvoll
 - **Komponententest** – Eher **entwicklernahe Testteams**, ggf. mit Projektvorgaben zu Vorgehensweisen und Testdokumentation
 - **Integrationstest** – Eher **Projektteams**, die die Module verschiedener Entwicklerteams integrieren und testen
 - **System / Abnahmetest** – Eher Teams des **Auftraggebers / Kunden**, mit vorab definierten Kriterien zur Abnahme / Freigabe des Systems
- **Abhängig von Softwareentwicklungslebenszyklus-Modell**
 - In der agile Entwicklung z.B.
 - Tester sind Teil des Entwicklerteams
 - Einsatz externer Tester im Entwicklungsteam
 - Product Owner führen Abnahmetests am Ende jeder Iteration durch zur Validierung von User-Stories



Profile der Testmitarbeiter

- Im Rahmen des Testens werden Mitarbeiter mit **unterschiedlichen Qualifikationen** benötigt
 - In der Praxis hat es sich als nützlich erwiesen, folgende **Profile** innerhalb des Testprozesses zu unterscheiden
 - **Testmanager**
 - **Testdesigner / Testanalyst**
 - **Testautomatisierer**
 - **Testadministrator / Testsystemadministrator**
 - **Tester**
 - **Technischer Experte**
- } **Tester**
- Im nachfolgenden Kapitel werden die Profile auf zwei Rollen reduziert:
Testmanager und **Tester**

Testmanager*

- Verantwortet und kontrolliert das Testprojekt
 - SW-Test und Qualitätsmanagement
 - Testplanung, Testfortschritt und Teststeuerung
 - Leitung der Testaktivitäten
- Die Rolle kann auch eingenommen werden von
 - Projektmanager, Entwicklungsmanager (Anm: keine gute Idee!)
 - Qualitätssicherungsmanager
 - Manager einer Testgruppe, professionellem Testmanager
- Die Rolle im agilen Umfeld
 - Die täglichen Tests werden von im Team integrierten Testern ausgeführt
 - Übergreifende Aufgaben über mehrere Teams können von Testmanagern außerhalb des Entwicklungsteams übernommen werden, sogenannte Test Coaches

* auch: Testleiter, Testkoordinator, Testcoach, leitender Tester

Testdesigner / Testanalyst

- Erstellt die Testfälle und legt die Reihenfolge der Durchführung fest
- Erfahrungen sind gefordert in den Bereichen
 - Entwicklungs- und Test-Know-how
 - SW-Engineering-Kenntnisse
 - Wissen über Spezifikationsmethoden, Kenntnisse fachlicher Anforderungen

Testautomatisierer

- Prüft die Möglichkeiten der Automatisierung und setzt diese um
- Erfahrungen sind gefordert in den Bereichen
 - Erfahrung als Tester
 - Know-how in Testdesign und -automatisierung, Programmiererfahrung
 - Sehr gute Kenntnisse der eingesetzten Tools



Testsystemadministrator

- Richtet die Testumgebung ein und betreibt diese
- Erfahrungen sind gefordert in den Bereichen
 - Systemadministration (oder Zugriffsmöglichkeit auf einen Systemadministrator)
 - Entwicklungs- und Testwerkzeuge
 - Datenbanksysteme und Netzwerke, falls notwendig
 - Installation und Betrieb der Systemumgebung

Tester

- Führt die Tests nach den Vorgaben resp. den Spezifikationen durch
- Erfahrungen sind gefordert in den Bereichen
 - Allgemeine IT-Grundlagen
 - Testbasiswissen, Kenntnisse über das Testobjekt
 - Bedienung / Parametrierung des eingesetzten Tools
 - Erfahrung in der Testdurchführung und Protokollierung



Technische Experten

- Ergänzen das Testteam je nach Bedarf
- Erfahrungen sind gefordert in den Bereichen
 - Datenbankadministratoren oder -designer
 - Experten zu User Interface
 - Netzwerkspezialisten
 - Themenspezifisches Know-How
- Je nach Problemstellung und Testumgebung etc. können weitere Spezialisten in die Testteams gebracht werden
 - Gefordert ist hier spezielles Fachwissen, das nichts mit dem Test-Know-how zu tun hat, z. B. Usability-Experten



Soft Skills – rollenübergreifend

- Zu den fachlichen Qualifikationen kommen u. a.
 - Teamfähigkeit, politisches bzw. diplomatisches Geschick
 - Bereitschaft, scheinbare Tatsachen zu hinterfragen
 - Durchsetzungskraft, sicheres Auftreten
 - Exaktheit und Kreativität
 - Sicherer Umgang mit komplexen Situationen
 - Schnelle Auffassungsgabe
- Ohne diese zusätzlichen Eigenschaften ist ein Tester, trotz hoher fachlicher Kompetenz nur bedingt erfolgreich
- Nicht jede Rolle muss von einer separaten Person ausgefüllt sein, in kleinen Teams kann **eine Person mehrere Rollen** übernehmen

TEAM: Toll Ein Anderer Macht's / Totaler Einsatz Aller Mitglieder

Aufgaben des Testmanagers – Überblick

- Erstellung und Prüfung einer **Testrichtlinie** für das Unternehmen
- Einbringen der Testperspektive in die Projektaktivitäten
- Anstoßen von Analyse, Entwurf, Realisierung und Durchführung von Tests
- Überwachen von Testfortschritt und Testergebnissen gegen Endekriterien
 - Definition von Metriken in Rücksprache mit den Stakeholdern
 - Konsolidieren der Ergebnisse und Erstellen von Berichten und Auswertungen
- Auswählen von Werkzeugen und Schulungen / Trainings
 - Empfehlung des Budgets, Zuordnung und Zeit für Pilotprojekte
- Entscheidung über Implementierung der Testumgebung
- Aufsetzen eines adäquaten Fehlermanagementsystems*
- Aufsetzen des Konfigurationsmanagements* der Testmittel
- Entwickeln und fördern des Berufsbildes „Tester“ in Organisation

* Das sind keine reinen Testaufgaben, sondern fürs gesamte Projekt notwendig.
Der Testmanager stellt die Anforderungen.

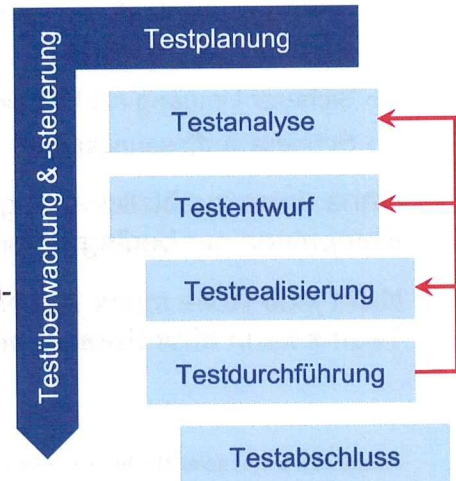
Aufgaben des Testmanagers in Testplanung / -steuerung

1 von 4

- Implementierung einer **Teststrategie** in einem spezifischen Projekt
- Initiierung, Überwachung und Steuerung von Tests und Testzyklen
- **Testzyklus - Durchlauf** durch den fundamentalen Testprozess auf **Basis** einer fixierten **Version** des Testobjekts

Testplanung

- **Grobplanung** der Testvorgehensweise:
zu **frühem Zeitpunkt** im Projekt erstellt,
im Testkonzept festgehalten
(Kontext, Risiken, Testdauer, Aufwand, Kosten)
- Erstellung eines **Planungsdokuments**, das u. a.
Umfang, Testvorgehensweise sowie Ressourcen-
und Zeitplanung der Tests beschreibt
- Definition der **Teststufen**
- **Testkonzept** erstellen, schreiben, aktualisieren
und abstimmen mit Beteiligten



Aufgaben des Testmanagers in Testplanung / -steuerung

2 von 4

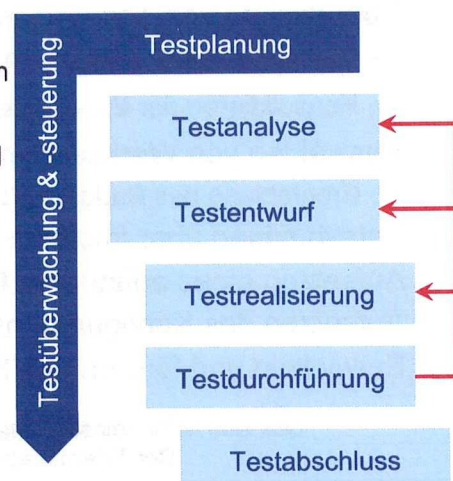
Testzyklusplanung

- **Feinplanung der Testzyklen** ist eine Ergänzung der Grobplanung, Abstimmung auf konkreten Testzyklus. Feinplanung ist **abhängig von der Projektsituation** (z. B. Entwicklungsstand, verfügbare Ressourcen)
- **Testplanung** beginnt mit **Start des Projekts**.
Hierin werden Ecktermine, Budget und Prioritäten
der verschiedenen Testaktivitäten berücksichtigt

Basis, um die Vorgaben für die Testanforderung
und die **Teststrategie** zu definieren

Ressourcen müssen eingeplant werden

- Sie sind meist **knapp** und müssen individuell
verteilt werden
- Während der Testzyklen kann es zu
Verschiebungen kommen, dann ist
Ressourcenplanung anzupassen
- Es ist an **Vertretungsregelungen** zu denken



Aufgaben des Testmanagers in Testplanung / -steuerung

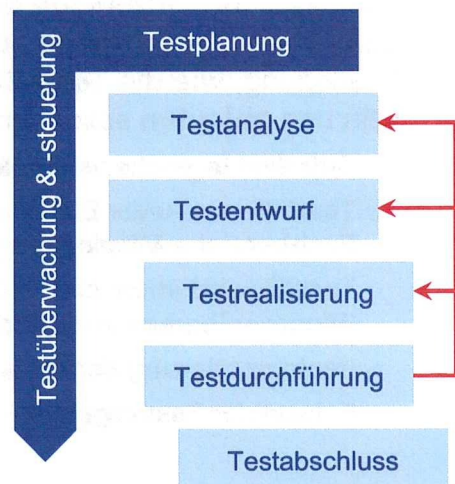
3 von 4

Projektentwicklung ist zu berücksichtigen

- Im Verlauf des Projektes kann es zu **Verzögerungen** kommen, so dass die gesamte zeitliche Planung in Gefahr ist
- Hierbei kann es sein, dass wegen verlorener Zeit **geplante Tests nicht mehr durchgeführt** werden können

Ergebnisse vorangegangener Tests **beeinflussen Testzyklusplanung**

- Auftretende **Probleme** können **Priorisierung** der Tests ändern
- **Korrigierte Fehlerzustände** machen **Nachtests** und eventuell Zusatztests erforderlich
- Tiefere **Analyse entdeckter Fehlerzustände** kann **Zusatztests** erforderlich machen



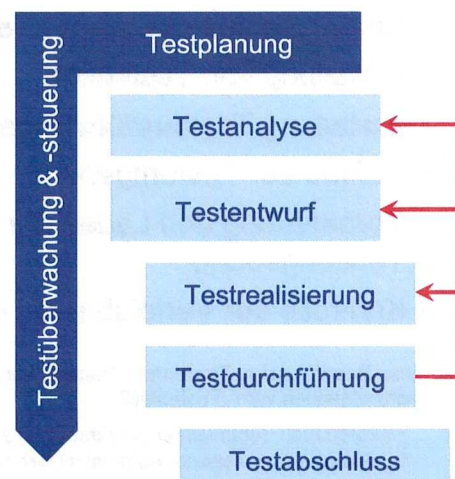
Aufgaben des Testmanagers in Testplanung / -steuerung

4 von 4

- **Erfolgskontrolle** und Überwachung der laufenden Tests geschieht **anhand von** im Testkonzept vereinbarten **Testmetriken**
- Dokumentation der Erfolgskontrolle im **Testfortschrittsbericht**
- **Testende** wird durch den Testmanager bestätigt, wenn die **Endekriterien** (bestimmt aus Testmetriken) **erfüllt** sind

Typische Endekriterien basieren bspw. auf

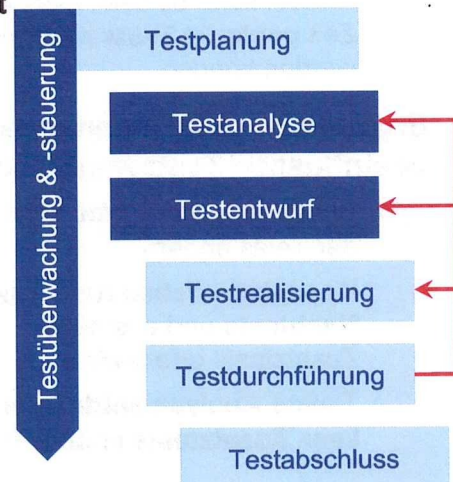
- Werte für Überdeckung, Funktionalität, Risiko
- Schätzungen über Restfehlerdichte oder Zuverlässigkeit
- Verbleibenden Risiken aus Restfehlerzuständen und -wirkungen oder fehlender Testüberdeckung
- Kosten, Termine



Aufgaben des Testmanagers in Testanalyse und Testentwurf

Ziel des Testens ist es, mit möglichst **geringem Aufwand** möglichst **viele** relevante **Fehlerzustände** zu finden!

- Initiierung der Analyse der Testbasis und der Spezifikation der Testfälle und Testdaten **auf Grundlage der im Testkonzept definierten Teststrategie**, d.h. auch Festlegung, **wie die Testfälle ausgestaltet** sein und **ablaufen** sollen, im Einzelnen
 - Definition **logische** vs. **konkrete** Testfälle
 - **Testfälle** sind unter Einbeziehung von **Fachbereichs-Mitarbeitern** zu erstellen
 - Testfälle sind im Hinblick auf ihre **Wiederholbarkeit** zu konzipieren
 - **Implementierung** der Tests / Testdurchführung
 - Entwurf der Testumgebung



Aufgaben des Testmanagers in Testrealisierung / -steuerung

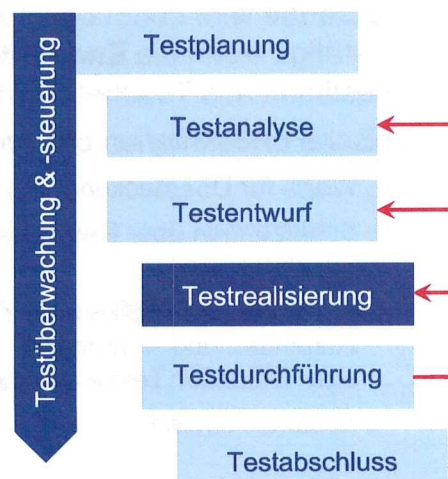
Initiierung der vorbereitenden Maßnahmen, wie

- Entwicklung Testabläufe
- Kontrolle der Softwareversion in Testumgebung
- Erstellung automatisierter Testskripte
- Erstellung von Testsuiten
- Erstellung **Testausführungspläne**
- Aufbau der Testumgebung
- Vorbereitung und Laden der Testdaten in die Testumgebung
- Kontrolle der Verfolgbarkeit (Traceability)

Diese Begriffspaare „Testskript / Testdrehbuch“ sind Spezialisierungen / Interpretationen vom „Testablauf“

...Testdrehbuch: (überwiegend) manueller Testablauf

...Testskript: (überwiegend) automatisierter Testablauf



Aufgaben des Testmanagers in Testdurchführung / -steuerung

Vergleich mit Plandaten während des Projektes

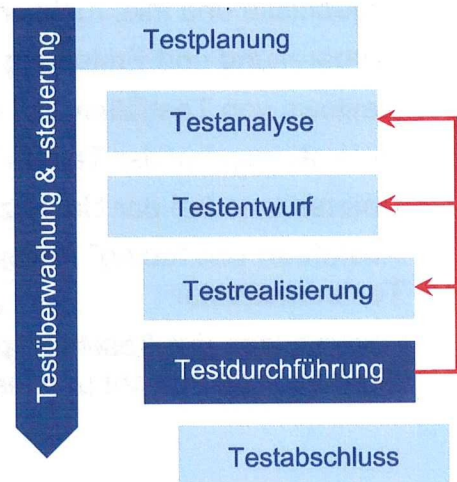
- Jeder einzelne **Testzyklus** muss mit dem Plan abgeglichen werden
- Liegen Ergebnisse im **vorausgesagten Bereich**? – Zeitbedarf, Anzahl gefundener Fehler, Korrekturaufwand, Nachtests usw.

Einhaltung des geplanten Aufwandes

- **Tooleinsatz** kann Abläufe effizient gestalten, muss aber **frühzeitig** erfolgen
- Weitgehende **Automatisierung** anstreben, wo sie mit vertretbarem Aufwand sinnvoll ist
- Risikobasiertes Vorgehen
- **Aber** bereits in der **Testplanung** muss **Automatisierung** berücksichtigt werden (langer Vorlauf)

Fehlermanagement

- Qualifizierung der **Abweichungen**
- Abgeleitete **Korrekturmaßnahmen**



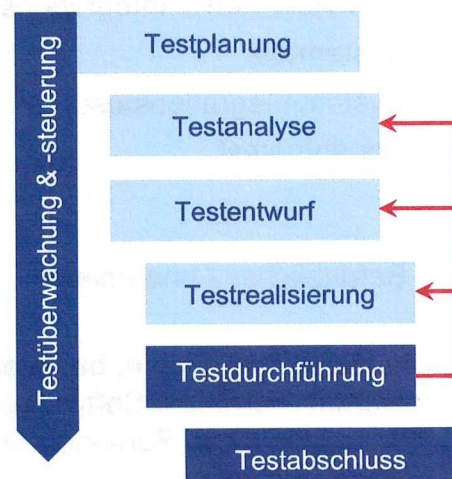
Aufgaben des Testmanagers in Testdurchführung und -abschluss

Die bei der Durchführung der Tests erstellten **Auswertungen** (z. B. Fehlerprotokolle, Zusammenfassungen, Einteilung in Fehlerklassen, Statistiken), die Fehlerverfolgungen, die Bewertung der Endkriterien **und das Reporting** an den Auftraggeber sind wichtige **Steuerungsmittel** für die Projektleitung und das **Management** (z. B. als eine Grundlage für die **Ressourcen- oder Zeitplanung**)

Schreiben von **Testabschlussberichten**, die die Informationen zusammenfassen, die während des Testens gesammelt wurden

Der Einsatz von **Tools** und standardisierten Templates erhöht die Qualität und **kann den Aufwand reduzieren**

Archivierung Testmittel, Übergabe IT-Betrieb, Lessons Learned, KVP



Aufgaben des Testers

- Mitarbeit bei der Testplanung und Testplan-Review
- Analyse, Review der Testbasis auf Testbarkeit
- Ausführen von Testfallentwurf und Testdurchführung (inkl. Auswertung der Ergebnisse und inkl. funktionale und nicht-funktionale Eigenschaften)
- Vorbereitung und Erstellung von Testdaten
- Reviews von Testfällen, die von anderen Testern geschrieben wurden
- Unterstützen bei der Testberichterstattung
- Unterstützen bei der Umsetzung der Testautomatisierung
- Ausführen von "doing" Aufgaben für die Testautomatisierung und Testinfrastruktur
- Unterstützen der Abschlussphase bei Rückfragen, Erstellung Testabschlussbericht und lessons learned

Wer übernimmt die „Tester“-Rolle?

Unterschiedliche Beteiligte übernehmen die Rolle „Tester“ für unterschiedliche Teststufen und -phasen. Tester, die Testanalyse, Testentwurf, spezifische Testarten oder Testautomatisierung übernehmen, sind oftmals Spezialisten

- | | | |
|-----------------------------------|---|--|
| • Komponenten- / Integrationstest | ➡ | Entwickler |
| • Systemtest | ➡ | unabhängiges Testteam |
| • Systemintegrationstest | ➡ | unabhängiges Testteam |
| • Abnahmetest | ➡ | Business Analysten, User, Fachexperten (Subject Matter Experts) des Kunden |
| • Betrieblicher Abnahmetest | ➡ | Operatoren und / oder Systemadministratoren des Kunden |

Die Rolle Tester kann, basierend auf den Risiken des Produkts, dem Projekt und dem gewählten Softwareentwicklungslebenszyklus-Modell, je Teststufe von unterschiedlichen Personen übernommen werden

Zusammenfassung

- **Testteams** sind um so **besser**, je **unabhängiger** sie sind, auch wenn sie sich aus Entwicklern der Testobjekte zusammensetzen
- **Der Testmanager**
 - Stellt sein **Team** frühzeitig zusammen
 - **Plant** die Tests und bereitet sie vor
 - Gibt die **Teststrategie** vor
 - Organisiert das **Fehlermanagement**
 - **Steuert** die Testdurchführung
 - **Werkzeugauswahl**, z. B. für Testautomatisierung
 - **Berichtet** an die Projektleitung und das Management
- **Der Tester** unterstützt die Testplanung, führt die Tests durch, erstellt die Fehlermeldungen und die Testdokumentationen und unterstützt ggf. bei der Testautomatisierung

Kapitel V – Testmanagement

- V/1 Testorganisation
- V/2 Testplanung und -schätzung
- V/3 Testüberwachung und -steuerung
- V/4 Konfigurationsmanagement
- V/5 Risiko und Testen
- V/6 Fehlermanagement

Grundlagen der Testplanung

Testplanung wird beeinflusst durch

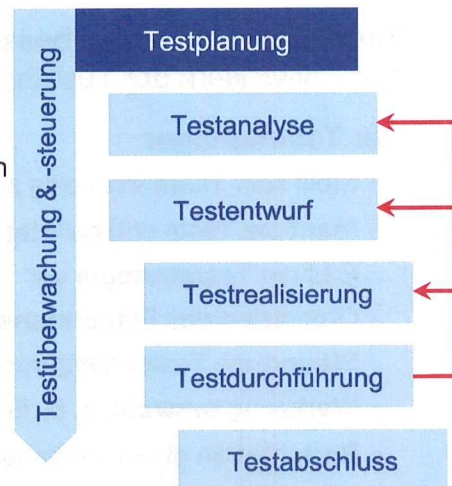
- Testtrichtlinie, Teststrategie des Unternehmens
- Entwicklungslebenszyklen, Verfahren
- Testumfang, Testziele, Risiken, Einschränkungen
- Kritikalität, Testbarkeit
- Verfügbarkeit von Ressourcen

Ist eine **kontinuierliche** Aktivität über den gesamten Produktlebenszyklus hinweg

- Die **Planung** ist laufend zu **prüfen** und ggf. auf Basis von Erkenntnissen aus laufenden Testaktivitäten **anzupassen**

Stufenweise Verfeinerung (level test plan)

- Bereits zu **Projektbeginn** ist eine Testplanung aufzustellen – sie ist Teil des Projektplans und somit die Entscheidungsgrundlage über das Projekt
- Im **Projektverlauf** werden sukzessive weitere Informationen erarbeitet, die zur laufenden Detaillierung der Testplanung einzusetzen sind



Testkonzept

Die Testplanung kann dokumentiert werden in

- a) einem Mastertestkonzept
- b) einzelnen Testkonzepten
 - je Teststufe (Systemtest, Abnahmetest)
 - für die einzelnen Testarten (Gebrauchstauglichkeit, Performance)

Das Testkonzept umfasst folgende Testplanungsaktivitäten

- Ermittlung von Budget, Umfang, Ziele und Risiken der Tests
- Festlegung der allgemeine Testvorgehensweise
- Integration und Koordination der Testaktivitäten in die SW-Lebenszyklusaktivitäten
- Entscheidungen treffen über
 - Was und wie ist zu testen?
 - Welche Personen und Ressourcen für die einzelnen Testaktivitäten nötig sind?
- Planung der Aktivitäten (sequentielle, agile, iterative Vorgehensmodelle)
- Definition der Metriken zur Testüberwachung und -steuerung
- Festlegung Detaillierungsgrad und Struktur der Testdokumentation (Vorlagen)

Mastertestkonzept nach ISO 29119 (vormals: IEEE 829)

- | | |
|--|--|
| 1. Dokumentenbezeichnung | 9. Testdokumentation |
| 2. Einführung | 10. Testaufgaben |
| 3. Testobjekte | 11. Testinfrastruktur |
| 4. Qualitätsmerkmale, die
getestet werden | 12. Verantwortlichkeiten,
Zuständigkeiten |
| 5. Qualitätsmerkmale, die
nicht getestet werden | 13. Personal, Einarbeitung,
Ausbildung |
| 6. Teststrategie | 14. Zeitplan, Arbeitsplan |
| 7. Abnahmekriterien und
Endekriterien | 15. Planungsrisiken und
Unvorhergesehenes |
| 8. Kriterien für den Testabbruch
und die Wiederaufnahme | 16. Genehmigung, Freigabe |



Teststrategie und Testvorgehensweise

Die **Teststrategie** beschreibt auf Produkt- oder Organisationsebene verallgemeinert den Testprozess

Häufig werden verschiedene Arten von Teststrategien kombiniert, sie ergänzen sich sinnvoll und ermöglichen dadurch ein effektiveres Testen, z.B. Risikobasiertes Testen (analytisch) mit explorativem Testen (reaktiv)

Die **Testvorgehensweise** ist die projekt- bzw. release-spezifische Anpassung der Teststrategie

• **Teststrategie** (test strategy)

Eine Dokumentation, die die generischen Anforderungen an das Testen in einem oder mehreren Projekten innerhalb einer Organisation beschreibt, einschließlich Details darüber, wie das Testen durchgeführt werden soll, und die an der Testrichtlinie ausgerichtet ist.

Gängige Arten von Teststrategien

1 von 3

- **Analytische** Teststrategie – z.B. risikobasiertes Testen
 - Analyse der **Testbasis** zur Identifikation der benötigten Testbedingungen
 - Ableitung der Testbedingungen aus den **Anforderungen oder Risiken**
 - z.B. Risikobasiertes Testen entwirft und priorisiert die Tests auf Basis der Risikostufe
- **Modellbasierte** Teststrategie – am Anforderungsprofil orientiert
 - Entwicklung eines **Modells** des Systems (tatsächliche / erwartete Situationen)
 - Modell auf Basis eines geforderten Produktaspektes
 - Funktion, Geschäftsprozess, interne Struktur, nicht-funktionale Eigenschaft
 - Berücksichtigung von Hardware, Software, Datenkapazität, Netzwerk und Infrastruktur der **erwarteten Produktionsumgebung**
 - z.B.: Geschäftsprozessmodell, Zustandsmodell, Zuverlässigkeitswachstumsmodelle

Gängige Arten von Teststrategien

2 von 3

- **Methodische** Teststrategie – z.B. qualitätsmerkmal-orientiert
 - **Vordefinierter** Satz von Tests oder Testbedingungen (Bsp. ISO 25010)
 - Checkliste oder Sammlung **generalisierter** Testbedingungen
 - Gängige oder wahrscheinliche Fehlerwirkungen, Liste von Q-Merkmalen
 - z.B. unternehmensweite Look-and-Feel Standards für mobile Apps oder Websites
- **Prozess- oder standardkonforme** Teststrategie (z.B. A-SPICE, MISRA)
 - z.B. für **medizinische** Systeme der Standard der Food & Drug Administration
 - Definiert von einem **Normungsausschuss** oder Expertengremium
 - Umfassen Dokumentation, Identifikation und Verwendung von Testbasis und Testorakel, Organisation des Testteams
 - **Scrum** verwendet **User-Stories**, wo Tester für jede Iteration den Testaufwand schätzen, Testbedingungen (Abnahmekriterien) identifizieren, Tests ausführen und den Status berichten

Gängige Arten von Teststrategien

3 von 3

- **Reaktive** Teststrategie – erst mit dem Erhalt des Testobjektes
 - Entwurf und Realisierung von Tests als Reaktion auf Erkenntnisse vorheriger Testergebnisse
 - **Exploratives** Testen, entwickeln von Test-Chartas für die relevanten Merkmale, Verteilung der Test-Chartas an die Tester, Überarbeitung der Test-Chartas für die nächste Testsitzung an Hand der Ergebnisse
- **Angeleitete oder beratende** Teststrategie – durch externes Testteam
 - Vorgehensweise auf Basis von Beratung, Anleitung oder Anweisungen von Stakeholdern, Fachexperten, die meist außerhalb des Testteams oder Unternehmens stehen
- **Regressionstestorientierte** Teststrategie – primär regressionsvermeidend
 - Wiederverwendung vorhandener Testmittel, Automatisierung von Regressionstests, Standardtestsuiten
 - Rückgang vorhandener Leistungsfähigkeit vermeiden

Testvorgehensweise – Auswahlkriterien

Aus der allgemeinen abstrakten Teststrategie wird durch den Testmanager die projektspezifische Testvorgehensweise

- Die **Testvorgehensweise** ist der Ausgangspunkt für die Auswahl von
 - Testverfahren, Teststufen, Testarten
 - Definition der Eingangs- und Endekriterien (Definition-of-Ready) (Definition-of-Done)
- Die **Entscheidungsbasis** für die Testvorgehensweise ist
 - Komplexität und Ziele des Projekts
 - Art des zu entwickelnden Produkts
 - Produktisikoanalyse
- Die ausgewählte **Vorgehensweise** wird bestimmt durch
 - Risiken, funktionale Sicherheit, verfügbare Ressourcen und Fähigkeiten
 - Technologie (bekannt / neu / einzigartig / etabliert)
 - Art des Systems (kundenspezifisch / Standardsoftware)
 - Testziele (Unternehmensziele / Qualitätsziele / Kundenziele)
 - Externe und interne Regularien / Vorschriften / Richtlinien

Eingangs- und Endekriterien (DoR) / (DoD)

Metriken für eine effektive Kontrolle der Qualität von Software und Tests sind

1. Eingangskriterien (Definition-of-Ready), bei Verfügbarkeit von
 - Testbaren Anforderungen, User-Stories, Modellen usw.
 - Testelementen (resp. Testobjekten) mit erfüllten Endekriterien der vorherigen Teststufen
 - Testumgebung, Testwerkzeugen, Testdaten, notwendigen Ressourcen
2. Endekriterien (Definition-of-Done), wenn folgendes gegeben ist
 - Geplante Tests wurden durchgeführt und festgelegte Überdeckung erreicht
 - Ungelöste Fehlerzustände innerhalb vereinbarter Grenze
 - Anzahl geschätzte Restfehler ist gering
 - Relevante Qualitätsmerkmale haben ausreichendes Niveau (Performance, Zuverlässigkeit, Zugriffssicherheit usw.)

Ein **Beginn ohne erfüllte Eingangskriterien** ist vermutlich schwieriger, teurer, risikoreicher, ein **Ende ohne erfüllte Endekriterien** ist üblich, das Risiko sollte geprüft und akzeptiert sein von den Fachverantwortlichen, Stakeholdern

Testausführungsplan

- **Finale** Festlegung der **Reihenfolge der durchzuführenden** manuellen und automatisierten Tests unter Berücksichtigung von Priorisierung und Risikoeinschätzung
- Berücksichtigt werden **Priorisierung, Abhängigkeiten**, die effizienteste Reihenfolge der Testdurchführung, Fehlernachtests und Regressionstests
- Dabei kann von der Reihenfolge streng nach der Priorität abgewichen werden
- Oft ein Kompromiss zwischen Effizienz der Testdurchführung und Einhaltung der Priorisierungen
- **Gute Testfälle** beschreiben die Abhängigkeiten von anderen Testfällen

Testaufwand beeinflussende Faktoren

1 von 2

- Testaufwandsschätzungen sind die **näherungsweise Beurteilung** des Testaufwandes auf der Basis von Erfahrungswerten und Informationen
- **Produkteigenschaften**
 - Produkttrisiken, Qualität der Testbasis
 - Größe und Komplexität des Systems unter Test (SUT)
 - Komplexität des Produkteinsatzbereichs
 - Besondere Qualitätsmerkmale (z.B. Zuverlässigkeit, Sicherheit)
 - Detaillierungsgrad der Testdokumentation (Dokumentationsstandards)
 - Gesetzliche oder regulatorische Anforderungen
- **Menschliche Eigenschaften**
 - Fähigkeiten und Erfahrungen der beteiligten Personen
 - Teamzusammenarbeit, Leitung

Testaufwand beeinflussende Faktoren

2 von 2

- **Entwicklungsprozesseigenschaften**
 - Stabilität und Reife der Organisation
 - Das genutzte Softwareentwicklungsmodell, der Testprozess
 - Vorgehensweise im Test
 - Anpassung oder Entwicklung von **Testwerkzeugen**, kundenspezifische Hardware
 - Außergewöhnliche Aufwände für Aufbau, Training und Einarbeitung
 - Zeitdruck (durch das Management, die Projektleitung)
- **Testergebnisse**
 - Fehlerhäufungen, Schweregrad der gefundenen Fehlerzustände
 - Anzahl der Änderungen (im Code, in den Anforderungen)

Testschätzverfahren

1 von 3

- Der Aufwand für die Vorbereitung und Durchführung von Tests ist lohnend, solange er niedriger ist als der Aufwand für die Fehlerkosten, die ohne Tests entstehen würden
- Die Entscheidung, wie viel Testaufwand noch wirtschaftlich ist, muss daher auf Annahmen und Schätzungen basieren
- Zwei der häufigsten Verfahren sind
 1. **Metrikbasierte Schätzung**
 2. **Expertenbasierte Schätzung**

Testschätzverfahren

2 von 3

1. Metrikbasierte Schätzung (analogie- und anteilsbasiert)

- Kategorisierung der durchzuführenden Testaufgabe und Lokalisieren eines bereits durchgeführten Projektes mit einer vergleichbaren Testaufgabe
- Übernahme des damals tatsächlich angefallenen Aufwands als Schätzgrundlage
- Auf Basis von Metriken (wie z.B. Lines Of Code, Anzahl Module, Anzahl Testfälle) hochrechnen auf das aktuelle Projekt
- Erfordert eine detaillierte **Historie** vergangener Projekte mit validen Ist-Aufwandsdaten, Organisationshistorie und Metriken sowie davon abgeleitete Modelle zum Schätzen von der Anzahl an Fehlern, Testzyklen und Testfällen
- **Durchschnittswerte** der Branche und Vorhersagemodelle, Anzahl Codezeilen, geschätzter Aufwand der Entwickler, etc.
- Beispiele
 - In **sequentiellen** Projekten: **Modelle der Fehlerentfernung** aus Vorprojekten mittels Erfassung von Fehlerzuständen und der Zeit, sie zu finden und beheben
 - In **agilen** Projekten: **Burndown-Charts**, dabei wird die Velocity (Story Points/Sprint) genutzt, um den Umfang der Aufgaben für die nächste Iteration zu schätzen

Testschätzverfahren

3 von 3

2. Expertenbasierte Schätzung (aufgabenbasiert)

- Der Schätzer ist der Experte
- Viel Erfahrung und Intuition seitens des Schätzers notwendig, um eine valide Schätzung durchzuführen, **Historiendaten** können helfen
- Übernahme des damals tatsächlich angefallenen Aufwands als Schätzgrundlage
- Mit genügend Erfahrung eingesetzt, ergeben sich in der Regel gute Näherungswerte für tatsächlich anfallenden Testaufwand
- Beispiele
 - In **sequentiellen** Projekten: **Breitband-Delphi-Schätzverfahren**, in dem Gruppen von Experten auf Basis ihrer Erfahrung schätzen
 - In **agilen** Projekten: **Planungspoker** (planning poker), Teammitglieder schätzen den Aufwand auf Basis ihrer Erfahrung (u.a. werden die geringste und höchste Schätzung dabei im Team erläutert)

Zusammenfassung

- Die Testplanung wird u.a. durch die Testrichtlinie und Teststrategie **des Unternehmens** beeinflusst
- Das **Testkonzept** als grundlegendes Element der Testplanung sollte zu **Beginn** der Tests erstellt werden (Muster für die Gliederung gibt ISO 29119)
- Arten von Teststrategien
 - Analytisch
 - Modellbasiert
 - Methodisch
 - Prozesskonform (oder standardkonform)
 - Angeleitet (oder beratend)
 - Regressionstestorientiert, leistungserhaltend
 - Reaktiv
- Die **Testaufwandsschätzung** kann mit verschiedenen Methoden durchgeführt werden: **metrik- oder expertenbasiert**, beide werden sowohl im agilen wie sequentiellen Umfeld genutzt

Kapitel V – Testmanagement

- V/1 Testorganisation
- V/2 Testplanung und -schätzung
- V/3 Testüberwachung und -steuerung
- V/4 Konfigurationsmanagement
- V/5 Risiko und Testen
- V/6 Fehlermanagement

V. Testmanagement » 3. Testüberwachung und -steuerung



Testüberwachung und -steuerung

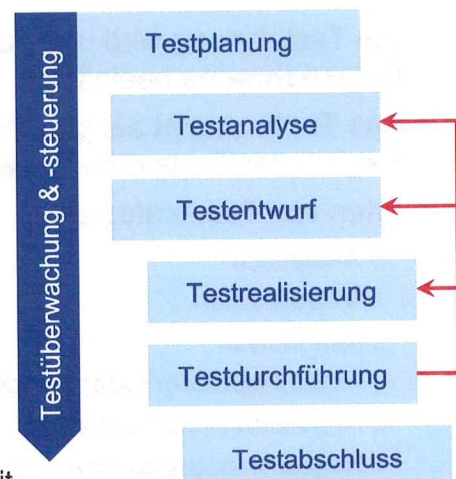
• Testüberwachung:

- Sammeln von Informationen über Testaktivitäten (manuell, automatisch)
- Beurteilung des Testfortschritts, mögliche Planabweichungen feststellen
- Prüfen, ob Testendekriterien (DoD) erfüllt sind

• Teststeuerung:

Leitende, korrigierende Maßnahmen auf Basis der gesammelten Metriken

- Neupriorisierung von Tests nach Risikoeintritt
- Anpassung des Testplans aufgrund Verfügbarkeit bzw. Nichtverfügbarkeit von Ressourcen oder der Testumgebung
- Neubewertung eines geänderten Testelements gegen Eingangs- und Endekriterien



Testüberwachung mittels Metriken

Metriken (messbare Größen, Kennzahlen) dienen der **Beurteilung** von

- Testfortschritt gegenüber dem Zeitplan, Budget; aktuelle Qualität des Testobjekts
- Angemessenheit des Testfortschritts, Effektivität der Testaktivitäten

Gängige Testmetriken sind

- Prozentsatz der durchgeführten Arbeiten in der **Testvorbereitung**, der **Testfallplanung** oder der Vorbereitung der **Testumgebung**
- Anzahl der durchgeführten/nicht durchgeführten **Testfälle** und der bestanden / nicht bestanden Testfälle
- **Testüberdeckung** der **Anforderungen**, Risiken oder des Codes
- **Fehlerinformationen**, z. B. **Fehlerdichte**, gefundene / behobene Fehlerzustände, Ausfallrate und Fehlernachtestergebnisse
- Testüberdeckung von Anforderungen, User-Stories, Abnahmekriterien oder Code
- **Testkosten**, inklusive der Kosten im Vergleich zum Nutzen durch das Auffinden des nächsten Fehlerzustandes oder für den nächsten Testdurchlauf

Testberichte: Zwecke, Inhalte und Zielgruppen

1 von 3

Testaktivitäten werden im Rahmen der Testberichterstattung verdichtet und kommuniziert (Testfortschrittsbericht, Testabschlussbericht) (ISO 29119-3)

- **Testfortschrittsberichte** werden regelmäßig vom Testmanager während der Testüberwachung und -steuerung erstellt
- Der **Testabschlussbericht** wird nach Erreichen der Endekriterien vom Testmanager erstellt (Zusammenfassung der durchgeführten Tests auf Basis des letzten Testfortschrittberichts und zusätzlicher relevanter Informationen)

Testberichte: Zwecke, Inhalte und Zielgruppen

2 von 3

Übliche Informationen in einem Testfortschritts- / abschlussbericht

- Zusammenfassung der durchgeführten Tests, Vorfälle während einer Testperiode
- Planabweichungen (Zeit, Dauer, Aufwand von Testaktivitäten) vom Testkonzept
- Erreichung von Meilensteinen (z. B. Abnahme von Testobjekten auf Teststufen), Stand der Tests, Produktqualität gegen Endkriterien oder DoD
- Benennen von Faktoren, die den Fortschritt blockiert haben oder noch blockieren
- Darstellen des Testfortschrittes anhand von Metriken gegenüber dem Testkonzept (Fehlerzustände, Testfälle, Testüberdeckung, Aktivitätsfortschritt, Ressourcenverbrauch)
- Aufzeigen der Restrisiken (neue / veränderte Risiken zum Vorbericht)
- Beurteilung der Qualität des Testobjekts
- Zusammenstellen der erzeugten wiederverwendbaren Testarbeitsergebnisse
- Darstellen der Aussichten, was für die nächste Testperiode geplant ist
- **Gesamtbewertung** / Status („Ampel“)

Testberichte: Zwecke, Inhalte und Zielgruppen

3 von 3

- **Varianten der Testberichterstattung** sind abhängig
 - Vom Projekt
 - Von den organisatorischen Anforderungen
 - Vom Softwareentwicklungslebenszyklus
- **Beispiele**
 - Ein **komplexes** Projekt mit vielen Stakeholdern, Regularien erfordert detaillierte, genaue Berichte
 - Bei der **agilen** Softwareentwicklung wird mit u.a. mit Task Boards (Kanban), Fehlerzusammenfassungen und Burn-Down-Charts gearbeitet, die in Daily's (tägliches Stand-up-Meeting) besprochen werden
 - Testberichte können auf die **Zielgruppe** angepasst sein
 - Technische Zielgruppe – detaillierte Informationen über Fehlerarten und Trends
 - Management (KLM) – abstrakt nach Fehlerstatusreport nach Priorität, Budget, Zeitplan, Testbedingungen erfüllt / nicht erfüllt / nicht getestet

Teststeuerung

1 von 2

Teststeuerung ist eine **Führungsaufgabe**

- Der Testmanager gehört zum Führungskader des Projekts
- Er benötigt die **Kompetenzen**, um z. B. **Überstunden** anzuordnen, externes Zusatzpersonal zu beauftragen usw.
- Er braucht Führungspersönlichkeit

Notwendigkeit von Teststeuerung

- Tests müssen **auf den Weg gebracht** werden
- Wenn der Weg **nicht eingehalten** wird oder nicht eingehalten werden kann, muss gesteuert werden
- Teststeuerung beinhaltet alle Führungs- und **Korrekturmaßnahmen**, die während der Testdurchführung ergriffen werden
- Teststeuerung kann sich ausschließlich auf die gemessenen **Testergebnisse** (siehe Testfortschrittsüberwachung) oder auf die **berichteten Informationen** (siehe Testberichterstattung) beziehen

Teststeuerung

2 von 2

- **Mehr Ressourcen (Zeit / Geld) bereitstellen**
 - Mehr Personal
 - Mehr Budget oder Umshiften
 - Tooleinsatz zur Mechanisierung von Tätigkeiten
- **Arbeitsaufwand reduzieren** (Abstriche an Qualität)
 - Varianten von Testfällen weglassen
 - Testtiefe von Testobjekten reduzieren
 - Tests mit weniger Testdaten durchführen
 - Testobjekte weglassen, Testsets weglassen
 - Funktionalität reduzieren / Stufenplan entwickeln
- Eingeleitete Teststeuerungsmaßnahmen sind zu **berichten**, um die Projektleitung / den Auftraggeber über möglicherweise veränderte Risiken der Produkteinführung zu informieren

Zusammenfassung

- Die **Testüberwachung** erfolgt anhand **messbarer Kriterien** und liefert so die Informationsbasis für die **Teststeuerung**
- Eine **regelmäßige Testberichterstattung** informiert die Projektleitung und das Management sowie den Kunden über den Teststatus, sie umfasst **Testfortschrittsberichte** und **Testabschlussberichte**
- Die **Varianten der Testberichte** hängt ab vom Projekt, von den organisatorischen Anforderungen und vom Softwareentwicklungslebenszyklus

Kapitel V – Testmanagement

- V/1 Testorganisation
- V/2 Testplanung und -schätzung
- V/3 Testüberwachung und -steuerung
- **V/4 Konfigurationsmanagement**
- V/5 Risiko und Testen
- V/6 Fehlermanagement

Konfigurationsmanagement

- Im **Verlauf** einer Softwareentwicklung werden **große Mengen** an Daten / Informationen / **Ergebnissen** erstellt
 - Anforderungen, Spezifikationen, Systementwürfe, Komponenten Systemversionen
- Die Integrität der Komponente, des Systems und der Testmittel sichern
 - Die Beziehungen untereinander und zueinander herzustellen
- Das Konfigurationsmanagement dient dazu
 - Alle Testelemente und Elemente der Testmittel
 - Sind eindeutig identifiziert, versionskontrolliert, Änderungen werden verfolgt
 - Stehen in Verbindung zueinander
 - Verfolgbarkeit ist zwischen den Elementen der Testmittel und den Testelementen möglich
 - Alle identifizierten Softwareelemente und Dokumente sind eindeutig in der Testdokumentation benannt
- Einführung eines Werkzeugs mit der Infrastruktur bereits in der Testplanung



Begriffe

Konfigurationsmanagement (KM)

Technische und administrative Maßnahmen zur Identifizierung und Dokumentation der fachlichen und physischen Merkmale eines Konfigurationselements, zur Überwachung und Protokollierung von Änderungen solcher Merkmale, zum Verfolgen des Änderungsprozesses, Umsetzungsstatus und zur Verifizierung der Übereinstimmung mit spezifizierten Anforderungen [IEEE 610]

- Das **Change-Management** dient der Verfolgung aller Aktivitäten (z. B. Änderung Quelltexte) im Rahmen eines Änderungsauftrages
- Das **Build-Management** beschreibt alle Schritte zum Aufbau der fertigen, auszuliefernden SW, sowohl für Komplettsysteme wie auch für durch Änderungen betroffene Teilsysteme
- Das **Release-Management** erlaubt die Klammerung versionierter Artefakte zu einem Gesamtprodukt für Test, Auslieferung u.Ä.
- Das **Versions-Management** hält für alle Artefakte (unter KM) fest, wer wann was angelegt oder geändert hat

Typische Probleme

Wird das Konfigurationsmanagement vernachlässigt wird es fast zwangsläufig z. B. zu folgenden Problemen kommen

- „**Versionsdurcheinander**“ – Unklarheit darüber, welche Dateiversionen zusammengehören, welche Versionen aktuell sind – es wird auf der Grundlage „alter“ Spezifikationen programmiert!
- **Wo und wann wurde etwas geändert?** Und von wem?
Paralleles Verändern von Dateien ist möglich, Änderungen werden u. U. wieder überschrieben
- **Welche Version** des Programms wurde getestet? Ohne klare Versionsangaben wird das Testen und die Testauswertung erschwert!

Anforderungen an das Konfigurationsmanagement

Aus der **Sicht des Testens** muss ein Konfigurationsmanagement **mindestens** die folgenden Anforderungen erfüllen

- **Versionenverwaltung / Versionskontrolle**
Katalogisieren, Speichern und Wiederabrufen von unterschiedlichen Versionen eines Objektes (V1.0, V1.1 etc.)
- **Protokollierung von Kommentaren** und Änderungsgründen
- **Konfigurationsverwaltung**
Bestimmung und Verwaltung aller Dateien in der jeweiligen Version, die zu einem Teilsystem zusammengesetzt werden können
- **Statusverfolgung**
Die Verfolgung von Fehlermeldungen und Änderungen, Aufzeichnung von Problembereichen und die Möglichkeit, deren Umsetzung nachzuvollziehen

Zusammenfassung

- Ein Konfigurationsmanagement wird benötigt, um die Testobjekte und deren **Versionsstände** geeignet zu **verwalten**
- Build- und Release-Informationen werden gespeichert, damit ältere Releasesstände später **rekonstruiert** werden können
- Konfigurationsmanagement wird für den **gesamten Softwareentwicklungslebenszyklus** genutzt, auch für den Testprozess
- Ohne **geeignete Werkzeuge** ist Konfigurationsmanagement kaum möglich

Kapitel V – Testmanagement

- V/1 Testorganisation
- V/2 Testplanung und -schätzung
- V/3 Testüberwachung und -steuerung
- V/4 Konfigurationsmanagement
- **V/5 Risiko und Testen**
- V/6 Fehlermanagement



Risiko – Begriffsdefinition

- Risiko (risk)

Ein Faktor, der zu negativen Konsequenzen in der Zukunft führen könnte; gewöhnlich ausgedrückt durch die Eintrittswahrscheinlichkeit und das Schadensausmaß.

- Ein Risiko ist die Wahrscheinlichkeit des Eintretens eines negativen Ereignisses multipliziert mit dem finanziellen Ausmaß
- Schadenshöhe × Eintrittswahrscheinlichkeit = Risikozahl
(Risikostufe meist gemessen in H=Hoch, M=Mittel, N=Niedrig)

- Risiko (aus: „Waltzing with bears“, Tom DeMarco / Timothy Lister*)

1. Ein mögliches, künftiges Ereignis, das zu unerwünschten Folgen führt
2. Diese Folgen selbst

- **Projekt- und Produktrisiken** sollten bei der Gestaltung und Planung der Tests, bei der **Auswahl und Priorisierung der Tests**, bei der Wahl der Testmethoden und bei der Testdurchführung berücksichtigt werden

* „Waltzing with Bears“, Tom DeMarco / Timothy Lister; Deutsch: Bärenango – Buchtipp

Projektrisiken – Kategorisierung

1 von 2

Projektrisiken beeinflussen Entwicklung und Test, gefährden Projektziele

- a) **Projektprobleme**

- Fehlerhafter Zeitplan des Projektes, späte Änderungswünsche, Überarbeitungen
- Verzögerungen bei Lieferungen, Aufgabenerfüllung, Endkriterien, Definition-of-Done
- Ungenaue Schätzungen, Umpriorisierungen von Mitteln, Kosteneinsparungen

- b) **Unternehmensprobleme**

- Zur Verfügung stehende Mitarbeiter, Qualifikation der zur Verfügung stehenden Mitarbeiter
- Zwischenmenschliche Probleme / Teambuilding (auch: Anerkennen von Testergebnissen)
- Mangelnde Kooperation beteiligter Abteilungen / Interessenkonflikte / Fachexperten

- c) **Politische Probleme**

- Tester und Testergebnisse werden nicht ausreichend wahrgenommen
- Erkenntnisse zur Testprozessverbesserung (Analyse von Tests und Reviews) werden nicht weiterverfolgt oder umgesetzt
- Keine Wertschätzung gegenüber Testern und Fehlerfindung während des Testens

Projektrisiken – Kategorisierung

2 von 2

d) Technische Probleme

- Anforderungen können nicht gut genug definiert sein (männlicher Internetauftritt, innovativ, ...)
- Anforderungen können unter den gegebenen Randbedingungen nicht erfüllt werden
- Bei Migrationen verspätete Datenkonvertierung, Migrationsplanung und -entwicklung, Tests der Datenkonvertierung oder Migrationswerkzeuge
- Schwächen im Entwicklungsprozess wie geringe Qualität des Designs, des Codes, der Konfigurationsdaten, der Testdaten und der Tests
- Schlechtes Fehlermanagement, kumulierte Fehlerzustände, technische Schulden
- Nicht rechtzeitige Verfügbarkeit von Testumgebungen

e) Lieferantenprobleme

- Zulieferung von Modulen / Software nicht in Time / Budget / Quality
- Vertragsaspekte bei Zulieferungen und Abnahmen
- Veränderungen von gesetzlichen oder sonstigen einzuhaltenden externen Regularien

Projektmanager verantworten den Umgang mit Projektrisiken, **Testmanager** den mit **testbezogenen** Projektrisiken

Produktrisiken – Qualitätsrisiken

Produktrisiken resultieren aus **Nichterfüllung** der Erwartungen des Nutzers

- Gelieferte **fehlerhafte** Software, gemäß Spezifikation
- Schlechte **Softwareeigenschaften** (funkt. und nicht-funkt. Qualitätsmerkmale)
- Das Produkt erfüllt nicht den geforderten Einsatzzweck und kann somit **nicht eingesetzt** werden
- Die Software verursacht **Sach- oder Personenschäden**
- Die Systemarchitektur unterstützt nicht-funktionale Anforderungen nicht angemessen
- Berechnungen nicht korrekt durchgeführt; Regelkreisverfahren nicht korrekt gebaut
- Antwortzeiten nicht angemessen
- Benutzermeldungen entsprechen nicht den Erwartungen (Fehlermeldungen)

Produktrisiken zu spezifischen Qualitätsmerkmalen, wie z.B. Funktionalität, Zuverlässigkeit oder Wartbarkeit werden auch **Qualitätsrisiken** genannt

Testen dient dazu, Produktrisiken zu vermindern oder zu vermeiden, also die **Eintrittswahrscheinlichkeit** von Produktrisiken zu senken

Risikobasiertes Testen und Produktqualität

1 von 2

- Risiken in **Risikoliste** erfassen – „Gefahr erkannt, Gefahr gebannt“, die Risikoliste muss regelmäßig aktualisiert werden
- Risiken und Testen werden genutzt, um
 - Den Testaufwand risikobasiert zu bündeln; was sollte wann mit Methodik und in welcher Intensität getestet werden
 - Risiken zu beherrschen, also eine Bewertung der identifizierten Risiken und Restrisiken (nicht gelösten Risiken) abzugeben
 - Die Eintrittswahrscheinlichkeit eines unerwünschten Ergebnisses und seiner Auswirkungen zu reduzieren (Reduzierung der Stufen des Produktrisikos)
- Überblick **Produktrisikanalyse**
 - Produktrisiko: Identifizierung, Beurteilung der Wahrscheinlichkeit und Auswirkung
 - Produktrisikoinformation hat Auswirkung auf die Lenkung von Testplanung, Spezifikation, Vorbereitung und Durchführung von Testfällen, Testüberwachung und Teststeuerung
 - **Eine frühe Analyse der Produktrisiken trägt zum Projekterfolg bei**

Risikobasiertes Testen und Produktqualität

2 von 2

Nutzung der Ergebnisse aus der **Produktrisikanalyse**

- **Testverfahren** geeignet auswählen, um die Risiken zu mindern
- Bestimmung der Teststufen und Testarten (IT-Sicherheitstests, Barrierefreiheit)
- **Testumfang**, optimaler Einsatz der Testressourcen zur Risikoreduktion
- **Priorisieren** der Tests, um kritische Fehlerzustände so früh wie möglich zu finden
- Prüfung, ob zusätzlich zum Testen noch weitere Aktivitäten notwendig sind wie **Schulungen** für unerfahrene Designer
- Risikomanagementaktivitäten

Risikomanagementaktivitäten von identifizierten Produktrisiken:

- Analyse und Neubewertung von Risiken, was kann falsch laufen
- Festlegung, welchen Risiken behandelt werden müssen
- Aktiv Maßnahmen ergreifen, die das Risiko vermindern
- Schaffung von Notfallplänen für den Fall des Risikoeintritts, Budget und Ressourcen bereithalten, sollte das Risiko eintreten

Zusammenfassung

- **Projekt- und Produktrisiken** gefährden den Projekterfolg und müssen **gemanagt** werden
- Produktrisiken beziehen sich auf das gelieferte **Softwareprodukt** und dadurch nicht erfüllte Bedürfnisse der Nutzer und Stakeholder
- Projektrisiken unterscheiden sich in **Projekt-, Unternehmens-, Politische, Technische und Lieferantenprobleme**
- Risikohöhe = Eintrittswahrscheinlichkeit × Schadenshöhe
- „Risikomanagement ist Projektmanagement für Erwachsene“

Kapitel V – Testmanagement

- V/1 Testorganisation
- V/2 Testplanung und -schätzung
- V/3 Testüberwachung und -steuerung
- V/4 Konfigurationsmanagement
- V/5 Risiko und Testen
- **V/6 Fehlermanagement**

Fehlerfindung während des Testens (defect management)

- Der Fehlermanagementprozess dient der Erkennung, Untersuchung, Maßnahmenergreifung und Behebung von Fehlerzuständen. Dazu gehört deren Protokollierung, Klassifizierung und Analyse der Auswirkung
- Das Finden von Fehlerzuständen ist eines der Ziele des Testens
- Fehlerzustände sollten aufgezeichnet werden, der Inhalt variiert
 - Abhängig vom Kontext wie Komponente, System, Teststufe, SW-Entw.-Modell
- Alle identifizierten Fehlerzustände werden untersucht
 - Entdeckung, Klassifizierung, Korrektur, Lösung, Nachtests, Bewertung
- Fehlermanagementprozess dient zur Verwaltung der Fehlerzustände
 - Workflow und Regeln für alle beteiligten Parteien (Designer, Entwickler, Tester, Product Owner), je nach Organisation formal bis informell

Fehlermanagementprozess

1 von 2

Im Fehlermanagement werden **alle gefundenen Fehlerwirkungen bzw. Abweichungen** im Projekt protokolliert

Neben der reinen Erfassung und Verwaltung werden noch **weitere Angaben** zu den einzelnen Fehlermeldungen hinterlegt

Alle Informationen werden in einem zentralen Fehlermanagementwerkzeug gespeichert

- Hier ist eine redundanzfreie Bearbeitung möglich
- Gewährleistung einer optimalen Übersicht über die gefundenen Fehlerzustände und deren Bearbeitung
- Übernimmt automatische Einträge von Kennung, Zuweisung, Aktualisierung des Status im Workflow und Benachrichtigung
- Abweichende Dokumentation bei Fehlerzuständen aus statischen Tests, Reviews

Anmerkung: Fehlerwirkungen entdecken und **melden** können Tester, Kunde, Entwickler, Projektleiter, usw.

Fehlermanagementprozess

2 von 2

Nur mit einem funktionierenden Fehlermanagement kann eine geordnete Fehlerbearbeitung erfolgen

- Aufbau des Fehlermanagements zu Beginn des Projektes

Ein korrekte **Fehler- / Abweichungsprotokollierung** umfasst unter anderem:

- Vorgabe einheitlicher Meldungen / Meldeformulare
- Festlegung, dass alle Fehlerzustände / Probleme dort erfasst werden – egal von welchem Beteiligten des Prozesses sie kommen und wo sie aufgetaucht sind
- Eine feste Vorgehensweise für die Behandlung von Fehlerzuständen

Standards für Klassifizierungen von Fehlern definieren

- Fehlerzustände können **auf verschiedenen Wegen** gefunden werden
 - Kodierung, statische Analyse, Reviews, dynamischem Testen
- Fehlerzustände können in unterschiedlichen **Arbeitsergebnissen** gefunden werden
 - Anforderungen, User-Stories, Abnahmekriterien, Testdokumenten, Installationsanleitungen
- Standards gewährleisten einen effektiven, effizienten Fehlermanagementprozess

Aufgabenverteilung

1 von 2

- **Tester**
 - **Führt Tests durch**, um Fehlerzustände aufzudecken
 - **Protokolliert** die Ergebnisse (Testprotokoll)
 - **Stellt die Fehlerwirkungen in die Datenbank / das Tool ein** (Fehler- bzw. Abweichungsbericht)
 - Falsch-positive Ergebnisse – Anomalie
 - Keine echten Fehlerwirkungen auf Basis von Fehlerzuständen
z.B. Unterbrechung der Netzwerkverbindung, Time-out
 - Tester sollten die Anzahl der falsch-positiven Fehlerberichte reduzieren
- **Testmanager**
 - **Wertet** die Fehlerberichte **aus** (Qualität der Arbeitsergebnisse)
 - Vergibt **Prioritäten** (Auswirkung auf das Testen)
(nach Abstimmung mit Projektleiter, Kunde, ...)
 - Liefert Ideen für die Verbesserung der Entwicklung und des Testprozesses

Aufgabenverteilung

2 von 2

- Entwickler
 - **Analysiert** Fehlerwirkungen und lokalisiert Fehlerzustände (~ Debugging)
 - **Korrigiert** die Fehlerzustände gemäß ihrer Priorität
 - Führt alle genehmigten **Änderungen** durch
- Aufgaben werden in einem **iterativen** Prozess erledigt
 - Tester
 - Testmanager
 - Änderungsmanagement (oder CCB)
 - Entwickler
 - Tester...
- Änderungsmanagement oder **Change Control Board (CCB)**
 - Trifft **Entscheidung** über Änderungen und deren Priorisierung

Aufbau des Fehlerberichts (incident logging)

1 von 2

Ein Fehlerbericht während des dynamischen Tests beschreibt die **Fehlerwirkung**, nicht die Fehlerursache!

- Muster für einen Fehlerbericht: ISO 29119-3 (Anomaly Report)

Der Fehlerbericht sollte mind. folgende Punkte umfassen

- **Fehlerdaten**
 - Eindeutige Identifikation (z. B. fortlaufend zu vergeben)
 - Titel und kurze Zusammenfassung des Fehlerzustands
 - Angaben zum Testobjekt (Name, Version) und der Testumgebung
 - Meldungsdatum des Fehlerberichts, Verfasser, Organisation
- **Einordnung des Fehlerzustandes**
 - **Fehlerklasse** (Schwere der Fehlerwirkung)
 - **Priorität** (Einschätzung zur Dringlichkeit)
 - **Fehlerstatus** (Neuer Fehlerzustand, Nachtest etc.)

Aufbau des Fehlerberichts

2 von 2

Der Fehlerbericht sollte mindestens folgende Punkte umfassen

- **Beschreibung**

- Referenzen wie den ausgeführten Testfall (liefert alle Angaben zu den Bedingungen)
- Fehlerwirkung (anhand der Darstellung der vorausgesagten Ergebnisse und der abweichenden Ist-Ergebnisse)
- Beschreibung des Fehlerzustands zur Reproduzierbarkeit (Screenshots, Protokolle, Datenbank-Dumps, Aufnahmen)
- Phase des Entwicklungslebenszyklus, in der der Fehlerzustand gefunden wurde
- Querverweise zu verwandten Meldungen
- Schlussfolgerungen, Empfehlungen, Freigaben
- Allgemeine Probleme und andere Bereiche, die ggf. durch die Änderung betroffen sind, die aus diesem Fehlerzustand resultiert

- **Verwaltungsdaten**

- Änderungshistorie, Reihenfolge der Aktionen, die ausgeführt wurden, um den Fehlerzustand zu isolieren, reparieren und als behoben zu bestätigen (Veränderungen im Lebenszyklus des Fehlerberichts sollten automatisch von der Fehlerdatenbank mitgeführt werden)

Einordnung des Fehlerzustands

1 von 2

- **Fehlerklasse**

- Die **Schwere** eines Fehlers wird in der **Fehlerklasse** ausgedrückt
- Fehlerklassen können frei gebildet werden (meist drei bis vier Fehlerklassen)
- Fehlerklassen beschreiben Stufen wie: kritische, schwere, mittlere, leichte Fehler
- Grundlage für die Klassifizierung kann der Grad der **Beeinträchtigung des Produkteinsatzes** sein (DIN 66271)

- **Beispiel**

Klasse	Bedeutung
1 – Kritisch	Systemabsturz mit ggf. Datenverlust; das Testobjekt ist in dieser Form nicht einsetzbar „nichts geht mehr“
2 – Hoch	Wesentliche Funktion ist fehlerhaft; Anforderung nicht beachtet oder falsch umgesetzt; das Testobjekt ist nur mit großen Einschränkungen einsetzbar „ich kann die Maske nicht nutzen“
3 – Mittel	Funktionale Abweichungen bzw. Einschränkungen; Anforderungen fehlerhaft oder nur teilweise umgesetzt; System kann mit Einschränkungen genutzt werden, „ich kann weiterarbeiten“
4 – niedrig	Geringfügige Abweichung; System kann ohne Einschränkung genutzt werden (Beseitigung hat mit dem nächsten Release zu erfolgen) „besser nutzbar wäre es, wenn...“
5 – Kosmetisch	Schönheitsfehler; System kann ohne Einschränkungen genutzt werden (Beseitigung kann mit einem der nächsten Releases erfolgen) „schöner wäre es, wenn...“

Einordnung des Fehlerzustands

2 von 2

- Dringlichkeit, Priorität der Behebung
 - Fehlerschwere, Bewertung der **Auswirkungen** des Fehlerzustandes
 - Einfluss auf Projektverlauf etc., wird das weitere Vorgehen im Prozess behindert?
 - Umfang und Grad der Auswirkungen auf Stakeholder-Interessen
 - Möglichkeit der unmittelbaren Behebung bzw., mit der nächsten Überarbeitung des Testobjektes etc.
 - Die **Abarbeitung / Korrektur** der Fehlerzustände erfolgt **entsprechend der gesetzten Prioritäten**

Priorität	Bedeutung
Umgehend	Der Arbeitsablauf beim Anwender ist blockiert oder die laufenden Tests können nicht fortgesetzt werden. Das Problem muss unmittelbar, ggf. provisorisch, behoben werden; ein Patch ist zu erstellen
mit nächster Version	Die Fehlerkorrektur erfolgt mit der nächsten regulären Produktversion oder mit der nächsten Testobjektlieferung
a.s.a.p.	Die Fehlerkorrektur erfolgt, sobald die betroffenen Systemteile ohnehin überarbeitet werden
offen	Korrekturplanung ist noch zu treffen

Einordnung des Fehlerzustands – Fehlerstatus

1 von 3

- Der Fehlerstatus gibt an, in welcher Bearbeitungsstufe sich der Fehlerzustand befindet, er beschreibt somit den **Fehlerlebenszyklus**
- Mögliche Stufen sind dabei z.B.
 - **Offen, neu** – Fehlerzustand wurde vom Tester neu angelegt
 - **Offen, qualifiziert** – Meldung wurde vom Testmanager freigegeben
 - **Zurückgewiesen** – Meldung wurde vom Testmanager abgewiesen
 - **Wartet auf Behebung** – Entwickler versucht den Fehlerzustand zu identifizieren
 - **Behebung** – Fehlerzustand ist gefunden und wurde für die Korrektur freigegeben (andernfalls wird der gefundene Fehlerzustand abgewiesen)
 - **Wartet auf Fehlernachtest** – wird vom Entwickler nach Korrektur vergeben
 - **Geschlossen / Erledigt** – wird vom Tester vergeben, wenn der Fehlerzustand laut Nachtest beseitigt ist
 - **Offen, alt** – wird vom Tester vergeben, wenn der Fehlerzustand weiterhin auftritt

Einordnung des Fehlerzustands – Fehlerstatus

2 von 3

Folgende Zustandsübergänge können vergeben werden (Beispiel)



Einordnung des Fehlerzustands – Fehlerstatus

3 von 3

Nur der Tester kann einen Fehlerbericht als **erledigt** bzw. geschlossen abhaken!

Ob ein Fehler **zurückgewiesen** oder **bearbeitet** wird, entscheidet der **Testmanager** bzw. das **Änderungsmanagement** auf Basis der dann vorliegenden Informationen zum Fehlerzustand und den Kosten für dessen Beseitigung

Alle **Veränderungen** werden mit Kommentaren im **Fehler / Abweichungsmanagement** erfasst

- Eine ständige Kontrolle über die Abarbeitung ist gegeben
- Der weitere Testverlauf lässt sich bestimmen
- Ggf. müssen **weitere Testfälle** generiert werden, wenn der betreffende Fehlerzustand nicht über interne Tests entdeckt wurde, sondern z. B. **vom Kunden** gemeldet wurde

Auswertungen der Fehlerberichte

Die Fehlerberichte werden systematisch ausgewertet, um den Abarbeitungsstand der Fehlerzustands zu verfolgen und daraus Informationen über die **Einhaltung der Projekt- und Testpläne** sowie das **erreichte Qualitätsniveau** der Software zu gewinnen

Typische Berichte / Auswertungen

- Zeigt sich eine **Zunahme oder Abnahme** der neu **gemeldeten / offenen Fehler** im Projektverlauf?
- In welchen Testobjekten **häufen** sich die Fehler? In welchen sind **unterdurchschnittlich** viele Fehler gefunden worden?
- Wie viele schwere **Fehler / Fehler mit hoher Priorität** sind noch offen?
- Wie lang sind die **Fehlerbearbeitungszeiten** / Durchlaufzyklen?
- Wie kann die Entwicklung und der Testprozess verbessert werden?
- Wie hoch ist die Qualität der Arbeitsergebnisse und welche Auswirkung hat das auf das Testen?



Übung IV.1: Fehlermanagement

(10 Min. Einzelarbeit / 15 Min. Ergebnisdiskussion)

- Schreiben Sie einen Fehlerbericht über den gesamten Fehlerverlauf
- Thema
Der Geldautomat hat Ihnen trotz korrekter PIN kein Geld ausgezahlt

Zusammenfassung

- Fehlermanagement ist die **Verwaltung** der beim Test gefundenen **Abweichungen** (Inhalte der Meldungen, Prozesse zur Behebung)
- Die Behandlung von Fehlerzuständen / Abweichungen ist ein **eigener Prozess** mit einem hierfür angepassten **Workflow**
- Für das Fehlermanagement stehen **leistungsstarke Tools** zur Verfügung, die auch Änderungsmanagement unterstützen
- Der Begriff **Abweichungsmanagement** wird synonym zu **Fehlermanagement** verwendet

Schlüsselbegriffe

- Eingangskriterien
- Endekriterien
- Fehlerbericht
- Fehlermanagement
- Konfigurationsmanagement
- Produktrisiko
- Projektrisiko
- Risiko
- Risikobasiertes Testen
- Risikostufe
- Testabschlussbericht
- Tester
- Testfortschrittsbericht
- Testkonzept
- Testmanager
- Testplanung
- Testschätzung
- Teststeuerung
- Teststrategie
- Testüberwachung
- Testvorgehensweise

VI. Werkzeugunterstützung für das Testen » Agenda



Kapitel VI – Werkzeugunterstützung für das Testen

- VI/1 Überlegungen zu Testwerkzeugen
- VI/2 Effektive Nutzung von Werkzeugen

Lernziele für Testwerkzeuge

6.1 Überlegungen zu Testwerkzeugen

- FL-6.1.1 (K2) Testwerkzeuge gemäß ihrem Zweck und den Testaktivitäten, die sie unterstützen, klassifizieren können
- FL-6.1.2 (K1) Nutzen und Risiken der Testautomatisierung identifizieren können
- FL-6.1.3 (K1) Sich an besondere Gesichtspunkte von Testdurchführungs- und Testmanagementwerkzeugen erinnern können

6.2 Effektive Nutzung von Werkzeugen

- FL-6.2.1 (K1) Die Hauptprinzipien für die Auswahl eines Werkzeugs identifizieren können
- FL-6.2.2 (K1) Sich an Ziele für die Nutzung von Pilotprojekten zur Einführung von Werkzeugen erinnern können
- FL-6.2.3 (K1) Erfolgsfaktoren für die Evaluierung, Implementierung, Bereitstellung und kontinuierliche Unterstützung von Testwerkzeugen in einem Unternehmen identifizieren können

Kapitel VI – Werkzeugunterstützung für das Testen

- VI/1 Überlegungen zu Testwerkzeugen
- VI/2 Effektive Nutzung von Werkzeugen

Allgemeines

Testaktivitäten können durch **Testwerkzeuge** unterstützt werden

- a) Werkzeuge, die für Tests genutzt werden
(Testausführungswerkzeuge, Testdateneditoren, Testdatengeneratoren)
- b) Werkzeuge, die beim **Verwalten helfen** von
 - Anforderungen, Testfälle, Testabläufe, automatisierte Testskripte, Testergebnisse, Testdaten, Abweichungen, Fehlerzuständen, u.v.m.**bzw. Unterstützung leisten** im oder beim
 - Berichtswesen zur Fortschrittsüberwachung
 - Steuerung der Testdurchführung
- c) Werkzeuge, die zur Untersuchung und Bewertung genutzt werden
- d) Werkzeuge, die in irgendeiner Form das Testen unterstützen können
(wie Tabellenkalkulationsprogramme)

Klassifizierung von Testwerkzeugen – Zwecke

- Verbesserung der Effizienz der Testaktivitäten
 - **Verlagerung von manuellen Aktivitäten** auf das Werkzeug, Automatisierung
 - Erhebliche Einsparung von Ressourcen bei sich **wiederholenden Tätigkeiten**
- Effizienzverbesserung durch **Unterstützung von manuellen Testaktivitäten**
- Verbesserung der **Qualität der Testdurchführung**
 - **Wiederholungen** identischer Tätigkeiten, Regressionstests
 - Bessere Konsistenz und höhere Fehlerreproduzierbarkeit
- Automatisierung von **manuell nicht durchführbaren Tests**
(z.B. Performancetest)
- **Verbesserte Zuverlässigkeit** des Testens
 - Datenhaltung in Testwerkzeugen ermöglicht **vielfältige Auswertungen**
 - Automatisierter Abgleich großer Datenmengen
 - Vereinfachter Zugriff auf **Informationen** über durchgeführte Tests wie Statistiken, Testfortschritt, Fehlerrate, etc.

Klassifizierung von Testwerkzeugen – Beispiele

1 von 2

- Kriterien zur Klassifizierung von Werkzeugen
 - Zweck, Preis
 - Lizenzmodell: kommerziell / kostenlos / Open-Source / Shareware
 - Verwendete Technologie
- Testwerkzeuge können mehr als einer Aktivität dienen
 - **Einzel-Tools**, die eine fest umrissene Aufgabe haben also eine **dedizierte Testaktivität** unterstützen
 - **Tool-Suiten**, die mehrere Aufgabenbereiche abdecken und bei denen die Einzel-Tools **miteinander integriert** sind
- Intrusive Testtools haben Einfluss auf das Testobjekt
 - **Intrusive** Tools nehmen direkt Einfluss auf das Testobjekt
 - Bei **Performance-Tests** kann es daher Beeinflussungen geben
 - Bei **Systemtests** sollen die Testobjekte im **Produktions-Timing** miteinander arbeiten, intrusive Werkzeuge sollten dann abgeschaltet sein

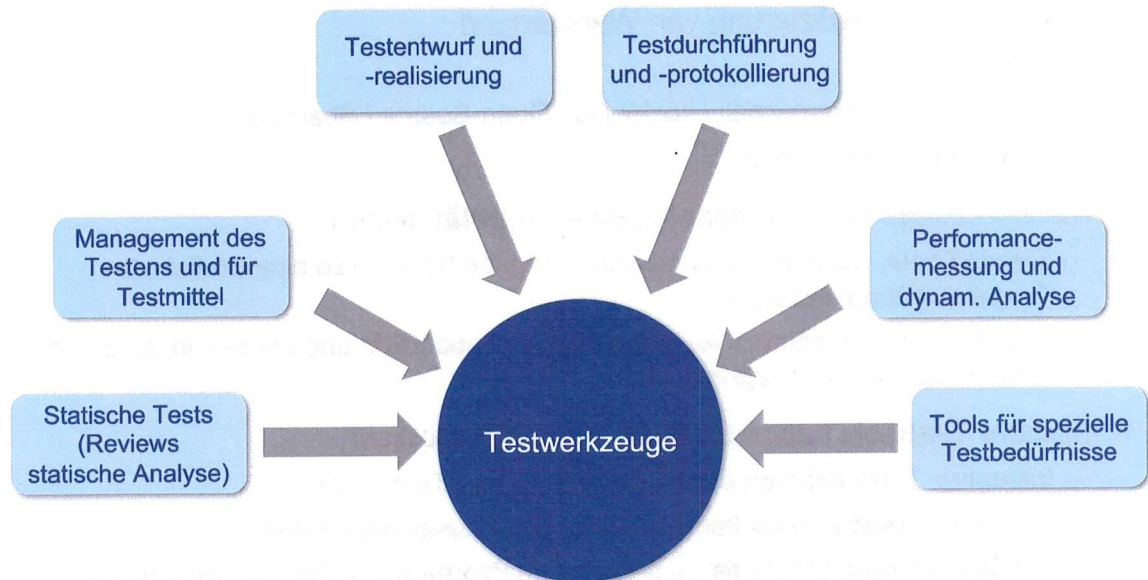
Klassifizierung von Testwerkzeugen – Beispiele

2 von 2

- Testwerkzeuge für spezifische Anwendungsbereiche, z.B.
 - Für bestimmte **Applikationstypen** wie **Webanwendungen**
 - Für bestimmte **Entwicklungsumgebungen** wie **Java**
 - Für bestimmte **nicht-funktionale** Tests wie **Sicherheitsprüfungen**
- Selbst erstellte Testwerkzeuge, z.B.
 - Excel-Tabellen
 - SQL-Skripte
 - spezifische Datenbanken zur Testdatenverwaltung
 - spezifische Komparatoren zum Soll-Ist-Abgleich

Hinweis: Werkzeuge zur Unterstützung von **Entwicklern** werden im folgenden als (Entwicklerwerkzeug) gekennzeichnet

Typen von Testwerkzeugen



Werkzeugunterstützung für statische Tests

- **Statische Analysewerkzeuge** (Entwicklerwerkzeug)
 - Einhaltung von **Programmierrichtlinien** / Konventionen
 - Unterstützung bei der Planung und auch bei der Risikoanalyse
 - **Strukturanalyse** von Quellcode
(z. B. prüfen von Links auf Gültigkeit im HTML-Code)
 - **Analysator** zur Auswertung von Quellcode, ermitteln von Metriken (z. B. der **zyklomatischen Zahl**) oder Identifizierung von Anomalien (Toter Code, **Datenflussanomalien**, **Kontrollflussanomalien**)

Werkzeuge für das Management des Testens und für Testmittel

1 von 3

Testmanagementwerkzeuge und ALM-Werkzeuge*

- Erfassung, Kategorisierung und **Verwaltung** von **Testfällen**
- Auswertungen / Erhebung von **Metriken**, die auf Testfällen beruhen
- Zeit- und **Ressourcenplanung**, Budgetplanung
- **Schnittstellen** zu Testauswertungs-, Fehlerverfolgungs- und Anforderungsmanagement-Werkzeugen
- unabhängige Versionskontrolle oder Schnittstelle zum entsprechenden Tool
- Verfolgbarkeit von Tests, Testergebnissen, Spezifikationen
- Erstellung von **Berichten** zur Testplanung
- Aufzeichnung von Testergebnissen
- Generierung von Fortschrittsberichten
- Quantitative Analyse (Metriken) zum Steuern und Verbessern des Testprozesses

* Application Lifecycle Management

Werkzeuge für das Management des Testens / Testmittel

2 von 3

• Anforderungsmanagementwerkzeuge

- strukturierte **Erfassung** von Systemanforderungen (Requirements)
- **Priorisierung** von Anforderungen
- **Verknüpfung** der Anforderungen mit Testfällen (Verfolgbarkeit zu Testobjekten)
- **Konsistenzprüfungen** / Auswertungen (z. B. zu Überdeckungen)

• Fehlermanagementwerkzeuge

- Erfassung und **Verfolgung von Fehlerberichten** (Fehlerzustände, Änderungsanforderungen (change requests), Fehlerwirkungen und wahrgenommene Probleme und Anomalien)
- Priorisierung, Kategorisierung, Sortierung von Abweichungen
- **Auswertungen** wie Testmetriken, **Testfortschrittsgrade**
- **Workflow** zur Abweichungs-Bearbeitung (Statuswechsel, Zuständigkeiten)

Werkzeuge für das Management des Testens / Testmittel

3 von 3

• Konfigurationsmanagementwerkzeuge

- Informationen über **Versionen** von **Komponenten**
- Verwaltung von Komponenten bzw. deren **Sourcecode**
- Verfolgbarkeit von Software-Produktkomponenten, den Varianten und Testmitteln, **Verknüpfung** mit Testmanagement / Anforderungsmanagement
- Verwaltung von mehreren Konfigurationen, Hard- und Softwareumgebungen (Bibliotheken, Browser, Rechner, etc.)

• Werkzeuge zur continuous integration (Entwicklerwerkzeug)

- Statische Codeanalyse und Lieferung der Ergebnisse
- Kompilieren und Linken des Codes, Erstellung einer ausführbaren Datei (Automatisierung des Buildprozesses)
- Durchführung von Unit-Tests, Prüfung der Codeüberdeckung
- Werkzeuge zur Versionskontrolle
- Regelmäßige Pflege der Werkzeuge ist notwendig

Werkzeugunterstützung für Testentwurf und -realisierung

1 von 2

• Testentwurfswerkzeuge, die aus den Anforderungen, der graphischen Benutzeroberfläche (GUI), dem Entwurfsmodell (Zustands-, Daten- oder Objektmodell) oder aus dem Code Folgendes **generieren** können

- Testeingaben, ausführbare Tests, Testorakel

• Testwerkzeuge für modellbasiertes Testen (MBT)

- **Nutzung formaler Modelle**, um das **Systemverhalten** zu beschreiben
- Bereitstellen von Funktionen zum **Ausführen der Modelle**
- Generieren großer Mengen unterschiedlicher Ausführungssequenzen
- Exportieren der Ausführungssequenzen als Testfälle
- Testen mit **anderer Sichtweise** auf das Testobjekt
- Gefunden werden Fehlerzustände, die bei anderen Tests übersehen werden
- Die **Erstellung der automatisierten Tests erfolgt (teil-)automatisiert**
- Die Testfallentstehung ist steuerbarer und transparenter
- Die Testqualität kann **personenunabhängiger** gestaltet werden

Werkzeugunterstützung für Testentwurf und -realisierung

2 von 2

- **Testdatengeneratoren und -editoren**
- Erzeugung und Bearbeitung von Testdaten, z.B. zur **Anonymisierung** zur Einhaltung des **Datenschutzes**
- **Datenbankbasierte Testdatengeneratoren**
 - Erkennen **Strukturen** und Inhalte aus **Datenbanken** oder Dateien und leiten Testdaten daraus ab
- **Codebasierte Testdatengeneratoren**, Erzeugen Testdaten aus dem **Quellcode**
- **Schnittstellenbasierte Testdatengeneratoren**
 - Erzeugen Testdaten auf Basis von Schnittstellenparametern
 - Leiten aus den **Definitionsbereichen Äquivalenzklassen und Grenzwerte** ab
- **Spezifikationsorientierte Testdatengeneratoren**
 - Erzeugen Testdaten aus einer formalen Notation einer **Anforderung (Case-Tools)**

Werkzeuge für Testdurchführung und -protokollierung

1 von 3

Es gibt verschiedene Tools, die den Tester in Abhängigkeit der jeweiligen **Teststufe** bei der Testdurchführung entlasten können

- **Testausführungswerkzeuge** übernehmen folgende Aufgaben
 - **Datenanlieferung** und Datenempfang oder Protokollierung des **Ausgabeverhaltens**
 - **Protokollierung** der Testdurchführung
 - Halb- / automatisierte Testdurchführung mit gespeicherten Daten
 - Unterstützung von Skriptsprachen
 - GUI-basierte Konfigurationen zur Parametrisierung der Daten oder für spezifische Anwendungen
 - Nützlich für **Regressionstests**

Werkzeuge für Testdurchführung und -protokollierung

2 von 3

- **Testrahmen / Simulatoren / Komponententestrahmen**
(Entwicklerwerkzeug)

- Tool, das die **Produktivumgebung** (oder einen Teil davon) nachbildet, nach Möglichkeit **eins-zu-eins**
- Unittest-Framework oder Testrahmen erleichtern den Test einer Komponente oder eines Teilsystems
- Simulation der Umgebung des Testobjekts
- Scheinobjekte (Simulatoren) als Treiber und / oder Platzhalter

Werkzeuge für Testdurchführung und -protokollierung

3 von 3

- **Komparatoren / Vergleichswerkzeuge**

- Tools für den Vergleich von **vorausgesagten und Ist-Ergebnissen**
- Basis des Vergleichs können Datenbanken oder Dateien verschiedener Formate sein
- Relevante Daten für den Vergleich werden über **Filterfunktionen** etc. aussortiert

- **Werkzeuge zur Überdeckungsmessung** (Entwicklerwerkzeug)

- Tools, die die jeweilige Überdeckung für den Test liefern (White-Box-Tests)
- Über eine **Instrumentierung** werden **Zähler** installiert, die jeden Zugriff registrieren
- Nach Testende können die Tools anhand der Zählerstände auswerten, welcher **Überdeckung** für den Test vorliegt (Anweisungs-, Entscheidungsüberdeckung)

Werkzeugunterstützung zur Performancemessung und dynamischen Analyse

1 von 2

• Last- / Performance- / Stresstestwerkzeuge

- Tools, die das **Laufzeitverhalten** des Testobjekts in verschiedenen Situationen überwachen
- Oft arbeiten diese Tools mit parametergesteuerten **Wiederholungen von Testfällen**, die automatisch ausgeführt werden
- Performancetestwerkzeuge werden in der Regel bei Testobjekten eingesetzt, die verteilt sind (auf mehrere Tasks oder Systeme) und z. B. über Netzwerke kommunizieren
- Meist ist dabei die Testumgebung nicht vollständig isoliert und unterliegt Einflüssen, die bei der Testvorbereitung und -durchführung nicht genau bekannt sind. Dadurch können die Testergebnisse verfälscht werden
- Die Wiederholbarkeit von Tests mit identischen Testergebnissen ist oftmals nicht gegeben
- Der Einsatz in komplexeren Systemumgebungen erfordert **Spezialistenwissen**, um keine verfälschten Ergebnisse zu erhalten (z. B. durch Netzwerk-Konstellationen)

Werkzeugunterstützung zur Performancemessung und dynamischen Analyse

2 von 2

• Dynamische Analysewerkzeuge (Entwicklerwerkzeug)

- Tools für die unterstützende Analyse **dynamischer Tests**
- Überwachung und Protokollierung des inneren Zustandes des Testobjektes (z. B. die **Speicherverwendung**, Speicherlecks, wilde Zeiger, Systemleistung, Netzwerkverhalten)
- Einsatz im Komponententest oder bei Integrationstests

• Sonstiges zu Testwerkzeugen

Neben den Werkzeugen welche die Hauptaufgaben des Testers unterstützen gibt es Werkzeuge speziell für den **Test nicht-funktionaler Anforderungen**, wie zum Beispiel:

- Benutzbarkeit, Sicherheit, Effizienz, Zuverlässigkeit

Testautomatisierung – Nutzen und Risiken

1 von 3

- **Neue Werkzeuge** garantieren nicht automatisch Erfolg
- **Aufwand** ist erforderlich, um echten und dauerhaften Nutzen zu erzielen
- **Potentieller Nutzen und die Risiken sollten betrachtet werden!**
- **Erwartungen** an das eingesetzte Werkzeug – **Nutzen**
 - **Reduktion** sich wiederholender Arbeiten
 - **Regressionstests** zur Sicherstellung der Altfunktionalität
 - **Wiederholbarkeit** von Arbeiten
 - Testdatenerstellung in immer gleicher Art und Weise
 - Durchführung von Tests in immer gleichen Abläufen
 - Generierte Metriken zur **objektiveren Beurteilung** (Überdeckung)
 - Generierung von Informationen für Statistiken, Graphiken, Testfortschritt, Fehlerraten
 - Durchführung und Wiederholen von Arbeiten, die manuell nicht oder nur mit erheblichem Aufwand zu leisten sind, wie Messungen von Performance, Last

Testautomatisierung – Nutzen und Risiken

2 von 3

- **Erwartungen** an das eingesetzte Werkzeug werden **nicht erfüllt**
 - **Funktionalität** des Werkzeugs **nicht wie erwartet**
 - **Benutzerfreundlichkeit** des Werkzeugs nicht wie erwartet
 - Es lässt sich doch nicht so leicht anwenden oder anpassen
- **Kein Ersatz** für manuellen Testentwurf und Testdurchführung
- Fehler bei der Schätzung von **Zeit, Kosten und Aufwand** der Einführung
 - **Schulungen** sind notwendig, externe Fachkenntnisse, kein Knowhow vor Ort
 - **Aufwand** der Beschaffung, der Einführung oder des Betriebs **unterschätzt**
 - **Prozesse** müssen angepasst werden, der Werkzeugeinsatz ist festgeschrieben
 - **Ohne** vorher Abläufe / **Testprozesse** selbst zu durchdenken und für das eigene Projekt geeignet festzulegen
- Aufwand für Wartung der Testarbeitsergebnisse
 - Abläufe ändern sich, **Schnittstellen** werden geändert, Datenstrukturen erweitert

Testautomatisierung – Nutzen und Risiken

3 von 3

- Die Kontrolle der **Testvorteile** durch das Werkzeug wird **vernachlässigt**
- Vernachlässigen der **Versionskontrolle** von Testgegenständen im Testwerkzeug
- Keine Berücksichtigung von **Schnittstellen** zu anderen wichtigen Werkzeugen anderer Hersteller wie Anforderungsmanagement, Fehlermanagement, Konfigurationsmanagement etc.
- Das Risiko, dass der Hersteller „**pleite**“ geht, das Werkzeug einzieht oder es an einen anderen Hersteller veräußert
- Mangelhafte Unterstützung durch den Hersteller beim **Support**, Updates oder der Fehlerbereinigung, Eigentumsverhältnisse ungeklärt
- Risiko der **Einstellung** von open-source oder freeware Werkzeugen
- Unvorhergesehenes wie keine Möglichkeit der **Übernahme** des Tools in eine andere **Umgebung** oder **neue Technologie**

Besondere Betrachtung der Testdurchführungs- und Testmanagementwerkzeuge

Testausführungswerkzeuge

- **Führen** Testobjekte aus
- Nutzen **automatisierte** Testskripte
- Erfordern oft einen erheblichen **Aufwand**, um Nutzen zu bringen
- Verwenden unterschiedliche Testautomatisierungsansätze

Die wichtigsten **Testautomatisierungsansätze** sind

- Aufgezeichnete Skripte
- Datengetriebener Ansatz
- Schlüsselwortgetriebener Ansatz

Testausführungswerkzeuge

1 von 3

Aufgezeichnete Testskripte (capturing test approach)

- Die einfache **Aufzeichnung** der Aktionen eines manuellen Testers **genügt** meist **nicht** – **Testskripte** müssen **erstellt** bzw. **erweitert** werden
- Oft eine **lineare Repräsentation** mit spezifischen Daten und Aktionen (linear scripting)
- Soll dasselbe Testscenario mit anderen Daten durchgeführt werden, müssen mit viel Aufwand **alle Daten im Script** einzeln gesucht und ersetzt werden
- Ungeeignet für eine große Anzahl von Testskripten
- Beim Auftreten **unerwarteter Ereignisse** können Skripte dieser Art **instabil** werden
- Setzen **technische Kenntnisse** in den Skriptsprachen voraus
- Benötigen **erwartete Ergebnisse** für jeden Test zum späteren Vergleich

Eine stetige Wartung der Skripte ist erforderlich, besonders wenn Benutzerschnittstellen oder Abläufe weiterentwickelt werden

Testausführungswerkzeuge

2 von 3

Datengetriebener Ansatz (data driven approach)

- Die Skripte werden so erstellt, dass sie die Programmfunktionen des Testobjekts bedienen. Die **Daten**, die dabei verwendet werden, holt sich das Skript aus einer **externen Datei** / Tabellenblatt / Datenbank
- Tester, die neue oder veränderte **Testfälle** ausführen möchten, müssen nicht die Skripte, sondern nur die **externe Datei** anpassen
- Problem: veränderte Daten können die **Reaktionen** des Testobjekts **verändern**, die ausführenden Skripte müssen dies verkraften
- **Problem**: Wenn sich die GUI des Testobjekts im Laufe des Projekts ändert, kann es zu **Ablaufschwierigkeiten** der Skripte kommen

Durch **Trennung der Kontrollinformationen von den Daten** kann der Wartungsaufwand reduziert werden

Bei einer **erneuten Ausführung** des Skriptes mit anderen Daten muss nur das **Datenfile ersetzt** werden, am Skript sind keine Änderungen nötig

Testausführungswerkzeuge

3 von 3

Schlüsselwortgetriebener Ansatz (keyword driven approach)

- Die Skripte werden so modularisiert, das jedes einzelne Skript möglichst nur noch eine **atomare Benutzerinteraktion** des Testobjekts auslöst
- Zusätzlich zu den in einer **externen Datei** hinterlegten **Testdaten** werden auch noch die aufzurufenden Funktionen extern hinterlegt
- Ein **Steuerskript** wertet diesen extern hinterlegten Testablauf aus und ruft die Funktionen (inkl. Daten) programmgesteuert ab
- Die Tester können so, **ohne die Skripte selbst zu modifizieren**, sehr **flexibel Testabläufe** der Testroboter steuern
- Extern zuzusteuern Daten werden schnell sehr komplex

Für den Einsatz von Testrobotern ist auf jeden Fall **Entwicklungs-Know-how / Skripting-Know-how** notwendig

Die **vorausgesagten Ergebnisse** der Tests sollten ebenfalls für maschinelle Auswertungen **verfügbar** sein, ansonsten nutzt man nicht das gesamte zur Verfügung stehende Potential

Testmanagementwerkzeuge

Einsatz integrierter Testmanagementwerkzeuge (z.B. ALM)

Schnittstellen zu anderen Werkzeugen (Testauswertungs-, Fehlerverfolgungs- und Anforderungsmanagement-Werkzeugen, Tabellen)

- Sammlung wichtiger Informationen, abgestimmt auf die Organisation
- Aufzeichnung von Testergebnissen
- Generierung von Fortschrittsberichten
- unabhängige Versionskontrolle, Schnittstellen zum Konfigurationsmanagementtool
- Verfolgbarkeit von Tests, Testergebnissen zu Anforderungen

Bei Nutzung durch verschiedene Gruppen sollte aus verschiedenen Aspekten auf Schnittstellen geachtet werden

- Projektplanungs- und -steuerungsinformationen
- Budgetinformationen
- Quantitative Analyse (Metriken) zum Steuern und Verbessern des Testprozesses

Kapitel VI – Werkzeugunterstützung für das Testen

- VI/1 Typen von Testwerkzeugen
- VI/2 Effektive Nutzung von Werkzeugen

Hauptgrundsätze für die Auswahl von Werkzeugen

1 von 2

Grundsätzliche Überlegungen zur Auswahl von Testwerkzeugen

- **Bewertung der Reife des Unternehmens**
 - Wo hat die Testorganisation Schwächen, wo Stärken?
 - Wo kann der Testprozess durch ein Werkzeug unterstützt und verbessert werden?
- Kompatibilität des Werkzeuges mit der eingesetzten Technologie und den Testobjekten
- Passt das Werkzeug zu bereits eingesetzten Werkzeugen für Build und continuous integration?
- **Anforderungsdefinition:** Anforderungen an das Werkzeug spezifizieren, objektive, messbare Kriterien verwenden (z. B. Anforderungsmatrix mit **gewichteten Anforderungen**)
- Steht das Werkzeug für eine kostenlose Testperiode zur Verfügung und wie lange?
- **Installation** des Werkzeugs in der Systemumgebung
- Ggf. Anpassung des Werkzeugs bzw. **Parametrierung**
- Sicherstellen des Systembetriebs, **Administrationsaufwand**

Hauptgrundsätze für die Auswahl von Werkzeugen

2 von 2

Grundsätzliche Überlegungen zur Auswahl von Testwerkzeugen

- **Evaluation der Anbieter und des Schulungsbedarfs:** Herstellerqualifikation prüfen (z. B. Trainingsunterstützung, Support, kommerzielle und vertragliche Aspekte)
- Gibt es interne Anforderungen nach Coaching und Anleitung zur Nutzung der Werkzeuge
- Identifizieren des Schulungsbedarfs (sind Test- und Testautomatisierungskennntnisse vorhanden)
- Abwägen verschiedener Lizenzmodelle (kommerziell / open source)
- **Beurteilung** der Kosten-Nutzen-Analyse anhand eines konkreten Geschäftsprozesses
- **Machbarkeitsstudie** (proof-of-concept) durch Nutzen des Werkzeugs während der Evaluierungsphase

Pilotprojekte für die Einführung von Werkzeugen im Unternehmen

Nach der Auswahl eines Werkzeugs **startet die Einführungsphase im Rahmen eines Pilotprojektes** mit folgenden Zielen

- Detailliertes **Kennenlernen** des Werkzeugs und seiner **Stärken und Schwächen**
- **Evaluierung:** In welchem Maße müssen bestehende Prozesse und Methoden angepasst und erweitert oder verändert werden
- Entscheidung über **Standardisierung** und Wartung des Werkzeuges
 - Verwaltung, Speicherung der Daten und Testbestände
 - Namenskonventionen, Programmierrichtlinien (Tests!), Bibliotheken, Module
- Einschätzung, ob sich die erwarteten **Nutzenpotenziale** einstellen und ob der erwartete **Aufwand** der Nutzung im Rahmen bleibt (ROI)
- **Metriken** verstehen und das Werkzeug konfigurieren: Welche Metriken das Werkzeug liefert, wie sie erfasst, aufgezeichnet, gesammelt und ausgewertet werden

Erfolgsfaktoren für Werkzeuge

Nach dem Pilotprojekt erfolgt die **schrittweise** Einführung im **gesamten Unternehmen**, nicht nur in einem Bereich

- Der Werkzeugeinsatz muss in den betreffenden Arbeitsabläufen / **Prozessen** festgeschrieben werden
- **Benutzerrichtlinien** für den Werkzeugeinsatz sind notwendig
- Schulungsmaßnahmen, Coaching und **Support** nicht nur bei der Einführung, sondern auch während des laufenden Einsatzes
- **Erfahrungen** aus dem Einsatz **aller Benutzer** messen und sammeln
- Die **tatsächliche Nutzung** und der Nutzen des Werkzeugs sind zu verfolgen, ggf. sind geeignete **Steuerungsmaßnahmen** zu ergreifen, um die Nutzung sicherzustellen
- **Integration in den Softwareentwicklungslebenszyklus**, technisch, organisatorisch und organisationsübergreifend

Zusammenfassung

- Es gibt unterschiedlichste Typen von **Testwerkzeugen**, die die **verschiedenen Aufgabengebiete** des Tests unterstützen
 - Werkzeuge für das Management des Testens und für Testmittel
 - Werkzeuge für statisches Tests
 - Werkzeuge für Testentwurf und -realisierung
 - Werkzeuge für Testdurchführung und -protokollierung (z. B. Komparatoren)
 - Werkzeuge zur Performancemessung und dynamischen Analyse
 - Werkzeuge für spezielle Testbedürfnisse
 - Werkzeuge, die keine reinen Testwerkzeuge sind
- Der Einsatz von Testwerkzeugen soll auf Basis einer Bewertung der Unternehmensreife und **Kosten-Nutzen-Analyse** wohlüberlegt erfolgen
- Die **Verwendung** eines neuen Testwerkzeugs muss **gezielt vorbereitet, durchgeführt** und fortgeführt werden, um erfolgreich zu sein

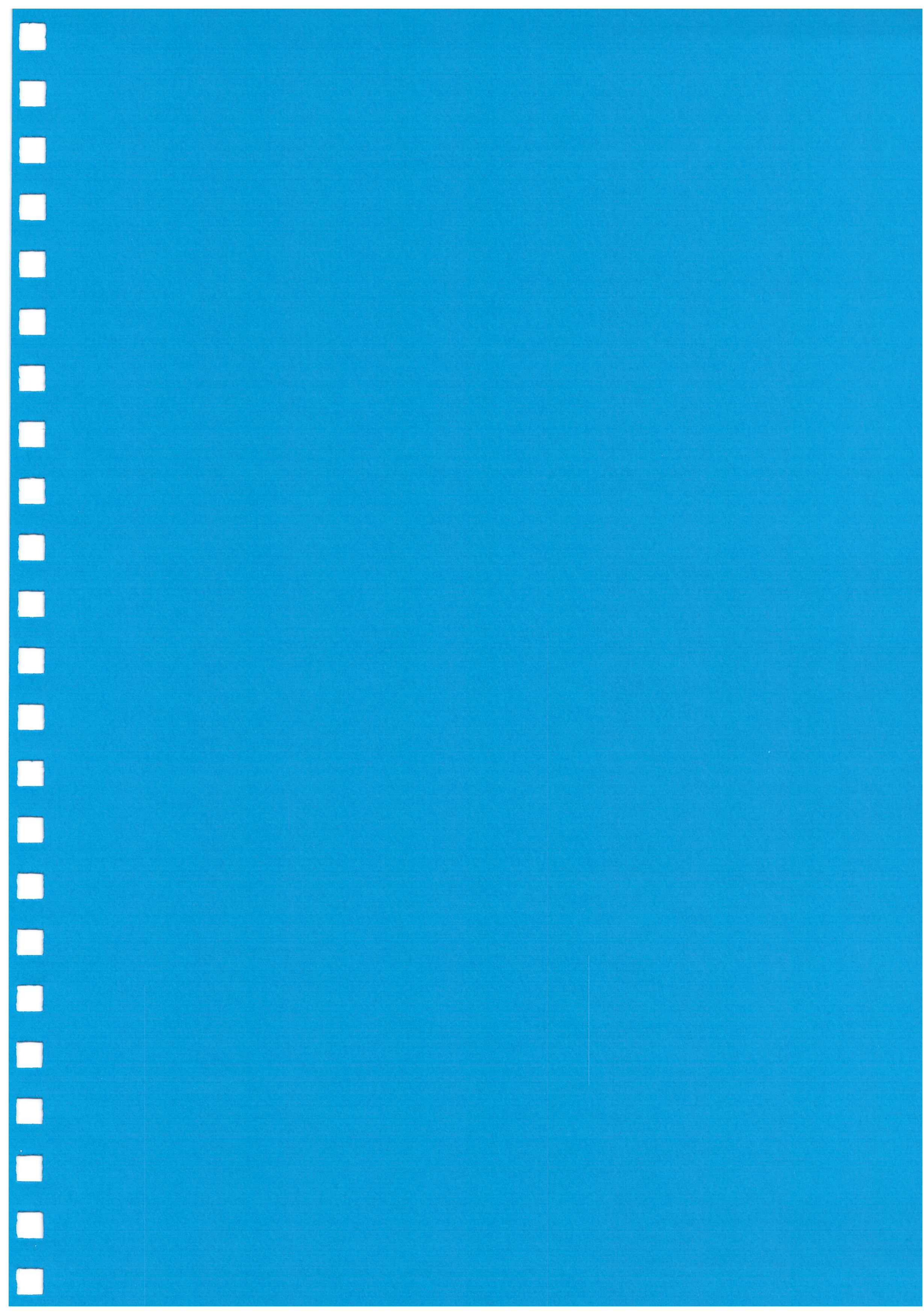
Schlüsselbegriffe

- Datengetriebener Ansatz
- Schlüsselwortgetriebener Ansatz
- Testautomatisierung
- Testausführungswerkzeug
- Testmanagementwerkzeug

Danke, dass Du dabei warst!

**Wir wünschen Dir eine
erfolgreiche Prüfung zum**

ISTQB® Certified Tester – Foundation Level



**ISTQB® Certified Tester Foundation Level 2018 -CORE-
Sample Exam Paper ("Probeprüfung"), SET B**



Familienname, Vorname: _____

Firmenadresse: _____

Telefon: _____

Fax: _____

E-Mail-Adresse: _____

Rechnungsanschrift: _____

Schulungsunternehmen: _____

Referent: _____

**Foundation Level CORE Probeprüfung
SET B (ISTQB V.1.3)**

CTFL CORE Lehrplan Version 2018 (V.3.1)

ISTQB Glossar V.3.3

ISTQB® Certified Tester Foundation Level

(Hinweis: Wenn nicht anders gekennzeichnet ist nur eine der vorgegebenen Antworten jeweils zutreffend.)

Einführung

Dies ist eine Probeprüfung. Sie hilft den Kandidaten bei ihrer Vorbereitung auf die Zertifizierungsprüfung. Enthalten sind Fragen, deren Format der regulären ISTQB®¹ / GTB Certified Tester Foundation Level Prüfung ähnelt.

Es ist strengstens verboten, diese Prüfungsfragen in einer echten Prüfung zu verwenden.

- 1) Jede Einzelperson und jeder Schulungsanbieter kann diese Probeprüfung in einer Schulung verwenden, wenn ISTQB® als Quelle und Copyright-Inhaber der Probeprüfung anerkannt wird.
- 2) Jede Einzelperson oder Gruppe von Personen kann diese Probeprüfung als Grundlage für Artikel, Bücher oder andere abgeleitete Schriftstücke verwenden, wenn ISTQB® als Quelle und Copyright-Inhaber der Probeprüfung bestätigt wird.
- 3) Jedes vom ISTQB® anerkannte nationale Board kann diese Probeprüfung übersetzen und öffentlich zugänglich machen, wenn ISTQB® als Quelle und Copyright-Inhaber der Probeprüfung bestätigt wird.
- 4) Zu fast jeder Frage wird genau eine zutreffende Lösung erwartet. Bei den Ausnahmen wird explizit auf die Möglichkeit mehrerer Antworten hingewiesen.

Allgemeine Angaben zur Probeprüfung:

Anzahl der Fragen: 40

Dauer der Prüfung: 60 Minuten

Gesamtpunktzahl: 40 (ein Punkt pro Frage)

Punktzahl zum Bestehen der Prüfung: 26 (oder mehr)

Prozentsatz zum Bestehen der Prüfung: 65% (oder mehr)

¹ International Software Testing Qualifications Board

**Fragen zum Thema
"Grundlagen des Testens"**

Frage 1	Keywords	K1	Punkte 1.0
----------------	-----------------	-----------	-------------------

Welche der folgenden Aussagen ist die korrekte Definition des Begriffes „Testfall“?

Wählen Sie genau EINE korrekte Option aus!

a)	Teilmenge des Wertebereichs innerhalb einer Komponente oder eines Systems, für die aufgrund der Spezifikation erwartet wird, dass alle Werte gleichartig behandelt werden.	☐
b)	Menge von Vorbedingungen, Eingaben, Aktionen, erwarteten Ergebnissen und Nachbedingungen, welche auf Basis von Testbedingungen entwickelt wurden.	☐
c)	Arbeitsergebnis, welches während des Testprozesses erstellt wird und dazu gebraucht wird, um die Tests zu planen, zu entwerfen, auszuführen, auszuwerten und darüber zu berichten.	☐
d)	Informationsquelle zur Ermittlung des erwarteten Ergebnisses, um es mit dem tatsächlichen Ergebnis eines Systems unter Test zu vergleichen.	☐

Frage 2	FL-1.1.1	K1	Punkte 1.0
----------------	-----------------	-----------	-------------------

Welche der folgenden Aussagen ist ein typisches Ziel des Testens von Software?

Wählen Sie genau EINE korrekte Option aus!

a)	Fehlerwirkungen und Fehlerzustände aufdecken	☐
b)	Validierung von Projektplänen	☐
c)	Sicherstellen von vollständigen Tests	☐
d)	Vergleich der Istergebnisse mit den erwarteten Ergebnissen	☐

Frage 3	FL-1.2.3	K2	Punkte 1.0
---------	----------	----	------------

Welches der folgenden Beispiele ist eine Fehlerwirkung in einem Tempomat eines Autos?

Wählen Sie genau EINE korrekte Option aus!

a)	Der Entwickler des Systems hat vergessen, Variablen nach einem Ausschneiden und Einfügen umzubenennen.	☐
b)	Nicht benötigter Code, der beim Rückwärtsfahren einen Alarm auslöst, wurde in das System aufgenommen.	☐
c)	Das System hält die eingestellte Geschwindigkeit nicht mehr ein, wenn die Radiolautstärke erhöht oder verringert wird.	☐
d)	Die System-Entwurfsspezifikation gibt die Geschwindigkeit falsch an.	☐

Frage 4	FL-1.2.4	K2	Punkte 1.0
---------	----------	----	------------

Welche der folgenden Aussagen ist eher ein Fehlerzustand als eine Grundursache für einen Fehlerzustand in einem Fitness-Tracker?

Wählen Sie genau EINE korrekte Option aus!

a)	Der Anforderungsmanager war mit der Domäne des Fitnesstrainings nicht vertraut und ging zu Unrecht davon aus, dass die Benutzer die Herzschlag-Frequenz in Schlägen pro Stunde ablesen wollen.	☐
b)	Der Tester des Smartphone-Interfaces war nicht im zustandsbasierten Testen geschult und hat daher einen signifikanten Fehler übersehen.	☐
c)	Eine vom Entwickler für die GPS-Funktion fehlerhaft implementierte Konfigurationsvariable kann während der Sommerzeit zu Standortproblemen führen.	☐
d)	Die Designerin der Benutzeroberfläche hat noch nie an tragbaren Geräten gearbeitet und missverstand deshalb die Auswirkungen von reflektiertem Sonnenlicht.	☐

Frage 5	FL-1.3.1	K2	Punkte 1.0
---------	----------	----	------------

Als Ergebnis der Risikoanalyse werden mehr Tests auf die Bereiche des Systems unter Test angewendet, in denen die ersten Tests mehr Fehler als in den anderen Bereichen aufgedeckt haben.

Welcher der folgenden Grundsätze des Testens wird angewendet?

Wählen Sie genau EINE korrekte Option aus!

a)	Vorsicht vor dem Pestizid-Paradoxon.	☐
b)	Das Testen ist kontextabhängig.	☐
c)	Trugschluss: Keine Fehler bedeutet brauchbares System.	☐
d)	Häufung von Fehlerzuständen.	☐

Frage 6	FL-1.4.2	K2	Punkte 1.0
---------	----------	----	------------

Gegeben seien die folgenden Listen von Testaktivitäten (A.-D.) und Testaufgaben (1.-4.):

- A. Testentwurf
- B. Testrealisierung
- C. Testdurchführung
- D. Testabschluss

- 1. Erfassung von Änderungsanforderungen für offene Fehlerberichte
- 2. Identifizierung von Testdaten zur Unterstützung der Testfälle
- 3. Priorisierung von Testabläufen und Vorbereitung von Testdaten
- 4. Analyse von Abweichungen, um deren Ursache zu ermitteln

Welches ist die korrekte Paarung von Testaktivitäten und Testaufgaben?

Wählen Sie genau EINE korrekte Option aus!

a)	A-2, B-3, C-4, D-1	☐
b)	A-2, B-1, C-3, D-4	☐
c)	A-3, B-2, C-4, D-1	☐
d)	A-3, B-2, C-1, D-4	☐

Frage 7	FL-1.4.4	K2	Punkte 1.0
---------	----------	----	------------

Welche der folgenden Aussagen beschreibt AM BESTEN, wie ein Mehrwert durch Aufrechterhaltung und Wartung der Verfolgbarkeit zwischen Testbasis und Testartefakten erzielt wird?

Wählen Sie genau EINE korrekte Option aus!

a)	Wartungstests können basierend auf Änderungen der ursprünglichen Anforderungen vollständig automatisiert werden.	☐
b)	Es kann festgestellt werden, ob ein neuer Testfall eine höhere Abdeckung der Anforderungen erreicht.	☐
c)	Testmanager können feststellen, welche Tester die Fehler mit dem höchsten Schweregrad gefunden haben.	☐
d)	Bereiche, die möglicherweise durch Seiteneffekte einer Änderung beeinflusst werden, können durch Regressionstests gezielt überprüft werden.	☐

Frage 8	FL-1.5.2	K2	Punkte 1.0
---------	----------	----	------------

Welche der folgenden Eigenschaften findet man EHER in der Denkweise eines Testers als in der Denkweise eines Entwicklers?

Wählen Sie genau EINE korrekte Option aus!

a)	Die Leistung eines Testers wächst und reift mit steigender Erfahrung.	☐
b)	Die Fähigkeit zu erkennen, was an Lösungen falsch sein könnte.	☐
c)	Gute Kommunikation mit Teammitgliedern.	☐
d)	Aufmerksamkeit für Details.	☐

Fragen zum Thema

"Testen im Softwareentwicklungslebenszyklus"

Frage 9	FL-2.1.1	K2	Punkte 1.0
---------	----------	----	------------

Betrachten Sie die folgenden Aussagen über die Beziehungen zwischen Softwareentwicklungsaktivitäten und Testaktivitäten im Softwareentwicklungslebenszyklus:

1. Für jede Entwicklungsaktivität sollte es eine zugehörige Testaktivität geben.
2. Reviewaktivitäten sollten starten, sobald die finale Version der Dokumente verfügbar ist.
3. Testentwurf und Implementierung der Tests sollten während der entsprechenden Entwicklungsaktivitäten starten.
4. Testaktivitäten sollten schon in frühen Phasen des Softwareentwicklungslebenszyklus beginnen.

Welche der folgenden Optionen zeigt KORREKT, welche dieser Aussagen wahr und welche falsch sind?

Wählen Sie genau EINE korrekte Option aus!

a)	Wahr – 1, 2; Falsch – 3, 4	☐
b)	Wahr – 2, 3; Falsch – 1, 4	☐
c)	Wahr – 1, 2, 4; Falsch – 3	☐
d)	Wahr – 1, 4; Falsch – 2, 3	☐

Frage 10	FL-2.2.1	K2	Punkte 1.0
----------	----------	----	------------

Ein durchgeführter Test hat folgende Eigenschaften:

- Er basiert auf Schnittstellenspezifikationen.
- Der Schwerpunkt liegt auf dem Finden von Fehlerwirkungen in der Kommunikation.
- Die Testvorgehensweise wendet sowohl funktionale als auch strukturelle Testarten an.

Auf welcher der folgenden Teststufen wird dieser Test **AM WAHRSCHEINLICHSTEN** durchgeführt?

Wählen Sie genau **EINE** korrekte Option aus!

a)	Integrationstest.	☐
b)	Abnahmetest.	☐
c)	Systemtest.	☐
d)	Komponententest.	☐

Frage 11	FL-2.3.2	K1	Punkte 1.0
----------	----------	----	------------

Welche der folgenden Aussagen über Testarten und Teststufen ist ZUTREFFEND?

Wählen Sie genau EINE korrekte Option aus!

a)	Funktionaler und nicht-funktionaler Test können auf den Teststufen System- und Abnahmetest durchgeführt werden, während der White-Box-Test auf Komponenten- und Integrationstests beschränkt ist.	☐
b)	Funktionaler Test kann auf jeder Teststufe durchgeführt werden, während der White-Box-Test auf Komponententest beschränkt ist.	☐
c)	Es ist möglich, funktionalen, nicht-funktionalen und White-Box-Test in jeder Teststufe durchzuführen.	☐
d)	Funktionaler und nicht-funktionaler Test können auf jeder Teststufe durchgeführt werden, während der White-Box-Test auf Komponenten- und Integrationstests beschränkt ist.	☐

Frage 12	FL-2.3.3	K2	Punkte 1.0
----------	----------	----	------------

Welche der folgenden Aussagen vergleicht die Zwecke der Fehlernachtests und Regressionstests AM BESTEN miteinander?

Wählen Sie genau EINE korrekte Option aus!

a)	Der Regressionstest stellt sicher, dass alle früher durchgeführten Tests immer noch korrekt laufen, während der Fehlernachtest sicherstellt, dass Korrekturen an einem Teil des Systems die anderen Teile nicht negativ beeinflussen.	☐
b)	Der Fehlernachtest prüft, dass ein vorher gefundener Fehlerzustand korrigiert wurde, während der Regressionstest sicherstellt, dass die Korrektur keine anderen Teile des Systems negativ beeinflusst hat.	☐
c)	Der Regressionstest stellt sicher, dass Korrekturen an einem Teil des Systems die anderen Teile nicht negativ beeinflussen, während der Fehlernachtest prüft, dass alle früher durchgeführten Tests immer noch die gleichen Ergebnisse produzieren.	☐
d)	Der Fehlernachtest bestätigt, dass die Änderungen am System erfolgreich durchgeführt wurden, während der Regressionstest Tests durchführt, die vorher fehlgeschlagen sind, um sicherzustellen, dass sie jetzt korrekt funktionieren.	☐

Frage 13	FL-2.4.2	K2	Punkte 1.0
----------	----------	----	------------

**Welche der folgenden Aussagen beschreibt eine Aufgabe der
Auswirkungsanalyse im Wartungstest KORREKT?**

Wählen Sie genau EINE korrekte Option aus!

a)	Die Auswirkungsanalyse unterstützt bei der Entscheidung, ob sich eine Fehlerkorrektur beim zu wartenden System lohnt.	☐
b)	Die Auswirkungsanalyse identifiziert, wie Daten in das gewartete System zu migrieren sind.	☐
c)	Die Auswirkungsanalyse unterstützt bei der Entscheidung, welche Hot Fixes den meisten Nutzen für den Benutzer haben.	☐
d)	Die Auswirkungsanalyse unterstützt die Ermittlung der Effektivität neuer Wartungstestfälle.	☐

**Fragen zum Thema
"Statistischer Test"**

Frage 14	FL-3.1.2	K2	Punkte 1.0
----------	----------	----	------------

**Welche der folgenden Aussagen gibt den Nutzen des statischen Tests
KORREKT wieder?**

Wählen Sie genau EINE korrekte Option aus!

a)	Nach der Einführung von Reviews stellten wir fest, dass sich sowohl die Qualität der Spezifikationen als auch die für Entwicklung und Test benötigte Zeit erhöht haben.	☐
b)	Durch die Anwendung von statischem Test können wir den Test besser steuern und haben ein günstigeres Fehlermanagement, weil sich Fehlerzustände später im Lebenszyklus leichter finden lassen.	☐
c)	Da wir jetzt statische Analyse nutzen, haben fehlende Anforderungen abgenommen und die Kommunikation zwischen Testern und Entwicklern hat sich verbessert.	☐
d)	Seitdem wir statische Analysen eingeführt haben, finden wir Programmierfehler, die wir allein durch dynamischen Test möglicherweise nicht gefunden hätten.	☐

Frage 15	FL-3.2.1	K2	Punkte 1.0
----------	----------	----	------------

Welche der folgenden Aussagen zur Anwendung von Checklisten bei einem formalen Review ist KORREKT?

Wählen Sie genau EINE korrekte Option aus!

a)	Im Rahmen der Planung des Reviews erstellen die Reviewer die für das Review benötigten Checklisten.	☐
b)	Im Rahmen der Befundkommunikation füllen die Reviewer die für das Review bereitgestellten Checklisten aus.	☐
c)	Im Rahmen der Reviewsitzung erstellen die Reviewer auf Basis der für das Review bereitgestellten Checklisten Fehlerberichte.	☐
d)	Im Rahmen des Reviewbeginns (Kick-Off) erhalten die Reviewer die für das Review benötigten Checklisten.	☐

Frage 16	FL-3.2.2	K1	Punkte 1.0
----------	----------	----	------------

Welche der folgenden Optionen gibt die Rollen und Verantwortlichkeiten in einem formalen Review KORREKT wieder?

Wählen Sie genau EINE korrekte Option aus!

a)	Management – Entscheidet über die Durchführung von Reviews	☐
b)	Reviewleiter – Stellt den erfolgreichen Ablauf von Reviewsitzungen sicher	☐
c)	Protokollant – Behebt Fehlerzustände im Arbeitsergebnis, das einem Review unterzogen wurde	☐
d)	Moderator – Überwacht die stetige Kosteneffizienz	☐

Frage 17	FL-3.2.3	K2	Punkte 1.0
----------	----------	----	------------

Die Reviews in Ihrer Organisation haben die folgenden Eigenschaften:

- Es gibt die Rolle des Protokollanten
- Der Zweck ist es, potenzielle Fehlerzustände zu entdecken
- Die Reviewsitzung wird vom Autor geleitet
- Die Reviewer finden potenzielle Fehlerzustände durch individuelles Review
- Es wird ein Reviewbericht erstellt

Welche der folgenden Reviewarten wird hier **AM WAHRSCHEINLICHSTEN** verwendet?

Wählen Sie genau **EINE** korrekte Option aus!

a)	Informelles Review	☐
b)	Walkthrough	☐
c)	Technisches Review	☐
d)	Inspektion	☐

Frage 18	FL-3.2.4	K3	Punkte 1.0
----------	----------	----	------------

Sie wurden gebeten, an einem checklistenbasierten Review des folgenden Auszuges aus einer Anforderungsspezifikation eines Bibliothekssystems teilzunehmen:

Bibliothekare können:

1. Neue Nutzer registrieren.
2. Bücher von Nutzern zurücknehmen.
3. Mahngebühren für Nutzer erheben.
4. Neue Bücher mit deren ISBN, Autor und Titel dem System hinzufügen.
5. Bücher aus dem System löschen.
6. Systemrückmeldungen innerhalb von 5 Sekunden erhalten.

Nutzer können:

7. Maximal drei Bücher gleichzeitig ausleihen.
8. Die Historie ihrer ausgeliehen/reservierten Bücher anschauen.
9. Mit einer Mahngebühr wegen Nichtrückgabe eines Buches innerhalb von 3 Wochen belegt werden.
10. Systemrückmeldungen innerhalb von 3 Sekunden erhalten.
11. Ein Buch kostenfrei für maximal 4 Wochen ausleihen.
12. Bücher reservieren (falls sie ausgeliehen sind).

Alle Benutzer (Bibliothekare und Nutzer):

13. Können Bücher nach ISBN, Autor oder Titel suchen.
14. Können den Systemkatalog durchstöbern.
15. Das System soll innerhalb von 3 Sekunden auf Benutzeranfragen reagieren.
16. Die Benutzungsschnittstelle soll einfach zu bedienen sein.

Ihnen wurde der Checklisteneintrag zugewiesen, der ein Review der Spezifikation auf Inkonsistenzen zwischen einzelnen Anforderungen vorsieht (beispielsweise Konflikte zwischen Anforderungen).

Welche folgenden Kombinationen weisen Inkonsistenzen zwischen Anforderungspaaren RICHTIG auf?

Wählen Sie genau EINE korrekte Option aus!

a)	6-10, 6-15, 7-12	☐
b)	6-15, 9-11	☐
c)	6-10, 6-15, 9-11	☐
d)	6-15, 7-12	☐

Fragen zum Thema
"Testverfahren"

Frage 19	Schlüsselbegriff	K1	Punkte 1.0
----------	------------------	----	------------

Welche der folgenden Aussagen beschreibt AM BESTEN exploratives Testen?

Wählen Sie genau EINE korrekte Option aus!

a)	Eine Testvorgehensweise/ein Testansatz, bei der eine intensive Untersuchung des Hintergrunds des Testobjekts dazu genutzt wird, mögliche Schwachstellen zu identifizieren, die durch Testfälle untersucht werden.	☐
b)	Eine Testvorgehensweise/ein Testansatz bei dem die Tester, basierend auf ihrem Wissen, der Erkundung des Testelements und dem Ergebnis früherer Tests, dynamisch Tests entwerfen und durchführen.	☐
c)	Eine Testvorgehensweise/ein Testansatz, bei dem die Testaktivitäten - insbesondere Testanalyse und Testentwurf - als unterbrechungsfreie Sitzungen geplant werden, oft in Verbindung mit checklisten-basiertem Testen.	☐
d)	Eine Testvorgehensweise/ein Testansatz, das auf der Erfahrung, dem Wissen und der Intuition des Testers basiert.	☐

Frage 20	FL-4.1.1	K2	Punkte 1.0
----------	----------	----	------------

Welche der folgenden Zuordnungen von Beschreibungen zu verschiedenen Kategorien von Testverfahren trifft AM BESTEN zu?

1. Überdeckung wird auf Basis einer ausgewählten Struktur des Testobjektes gemessen.
2. Verarbeitung innerhalb des Testobjekts wird überprüft.
3. Tests basieren auf der Wahrscheinlichkeit von Fehlerzuständen und deren Verteilung.
4. Abweichungen von Anforderungen werden überprüft.
5. User-Stories werden als Testbasis herangezogen.

Verwendete Notation für die folgenden 4 Optionen:

Black – Black-Box-Testverfahren

White – White-Box-Testverfahren

Erfahrung – erfahrungsbasiertes Testverfahren

Wählen Sie genau EINE korrekte Option aus!

a)	Black – 4, 5; White – 1, 2; Erfahrung – 3	☐
b)	Black – 3; White – 1, 2; Erfahrung – 4, 5	☐
c)	Black – 4; White – 1, 2; Erfahrung – 3, 5	☐
d)	Black – 1, 3, 5; White – 2; Erfahrung – 4	☐

Frage 21	FL-4.2.1	K3	Punkte 1.0
----------	----------	----	------------

Eine Fitness-App misst die Anzahl der täglich gelaufenen Schritte und gibt dazu Feedback, um den Anwender zu motivieren, fit zu bleiben.

Das Feedback zu den verschiedenen Schrittzahlen soll sein:

- | | |
|------------------------|------------------------|
| Bis zu 1000 | - Couch Potato! |
| Über 1000, bis zu 2000 | - Faulpelz! |
| Über 2000, bis zu 4000 | - Die Richtung stimmt! |
| Über 4000, bis zu 6000 | - Nicht schlecht! |
| Über 6000 | - Super! |

Welche der folgenden Gruppen von Testeingabewerten würde für die BESTE Überdeckung von Äquivalenzklassen sorgen?

Wählen Sie genau EINE korrekte Option aus.

a)	0,	1000,	2000,	3000,	4000	☐
b)	1000,	2001,	4000,	4001,	6000	☐
c)	123,	2345,	3456,	4567,	5678	☐
d)	666,	999,	2222,	5555,	6666	☐

Frage 22

FL-4.2.1

K3

Punkte 1.0

Ein Gerät zur Messung des täglichen Strahlungseinfalls für Pflanzen ermittelt einen Einstrahlungswert für Sonnenschein. Dieser ergibt sich aus der Kombination der Anzahl der Stunden, in denen eine Pflanze der Sonne ausgesetzt ist (unter 3 Stunden, 3 bis 6 Stunden, über 6 Stunden) und der durchschnittlichen Intensität des Sonnenscheins (sehr niedrig, niedrig, mittel, hoch). Die folgenden Testfälle existieren bereits:

	Dauer (Stunden)	Intensität	Einstrahlung
T1	1,5	sehr niedrig	10
T2	7,0	mittel	60
T3	0,5	sehr niedrig	10

Wie viele Testfälle müssen mindestens noch erzeugt werden, um eine vollständige Überdeckung ALLER GÜLTIGEN Eingabe-Äquivalenzklassen zu gewährleisten?

Wählen Sie genau EINE korrekte Option aus.

a)	1	☐
b)	2	☐
c)	3	☐
d)	4	☐

Frage 23	FL-4.2.2	K3	Punkte 1.0
----------	----------	----	------------

Eine Smart-Home-App misst die durchschnittliche Temperatur im Haus während der vergangenen Woche und gibt den Bewohnern basierend auf diesem Wert Informationen zur Umweltfreundlichkeit ihres Verhaltens. Das Feedback für die verschiedenen Durchschnittstemperaturbereiche (gerundet auf die nächsten ganzen °C) soll lauten:

Bis zu 10°C	Eiskalt!
11°C bis 15°C	Ganz schön schattig!
16°C bis 19°C	Cool!
20°C bis 22°C	Zu warm!
Über 22°C	Sauna!

Welches der folgenden Testsets liefert die höchste Überdeckung von Grenzwerten, wenn die Grenzwertanalyse ausschließlich unter Verwendung der Minimal- und Maximalwerte verwendet wird?

Wählen Sie genau EINE korrekte Option aus.

a)	0°C,	11°C,	20°C,	22°C,	23°C	☐
b)	9°C,	15°C,	19°C,	23°C,	100°C	☐
c)	10°C,	16°C,	19°C,	22°C,	23°C	☐
d)	14°C,	15°C,	18°C,	19°C,	21°C 22°C	☐

Frage 24

FL-4.2.3

K3

Punkte 1.0

Ein System zur Berechnung der Strafe für Geschwindigkeitsübertretungen im Straßenverkehr wird mit folgender Entscheidungstabelle spezifiziert:

Regeln		R1	R2	R3	R4
Bedingungen	Geschwindigkeit > 50	J	J	N	N
	Schul-Zone	J	N	J	N
Aktionen	250€ Geldstrafe	-	X	-	-
	Führerscheinentzug	X	-	-	-

Für die beiden in der obigen Tabelle dargestellten Regeln R1 und R4 wurden bereits folgende zwei Testfälle generiert:

ID	Eingabe	Erwartetes Ergebnis
TF1	Geschwindigkeit = 65; Schulzone = Ja	Führerscheinentzug
TF2	Geschwindigkeit = 45; Schulzone = Nein	Keine Strafe

Die nächste Tabelle zeigt vier zusätzliche Testfälle TF3, TF4, TF5 und TF6:

ID	Eingabe	Erwartetes Ergebnis
TF3	Geschwindigkeit = 55; Schulzone = Ja	Führerscheinentzug
TF4	Geschwindigkeit = 44; Schulzone = Ja	Keine Strafe
TF5	Geschwindigkeit = 66; Schulzone = Ja	Führerscheinentzug
TF6	Geschwindigkeit = 77; Schulzone = Nein	250€ Geldstrafe

Siehe nächste Seite!

Welche zwei der zusätzlichen Testfälle würden eine 100% Überdeckung der gesamten Entscheidungstabelle erreichen (in Kombination mit den Testfällen TF1 und TF2)?

Wählen Sie genau EINE korrekte Option aus.

a)	TF3, TF4	☐
b)	TF4, TF5	☐
c)	TF4, TF6	☐
d)	TF5, TF6	☐

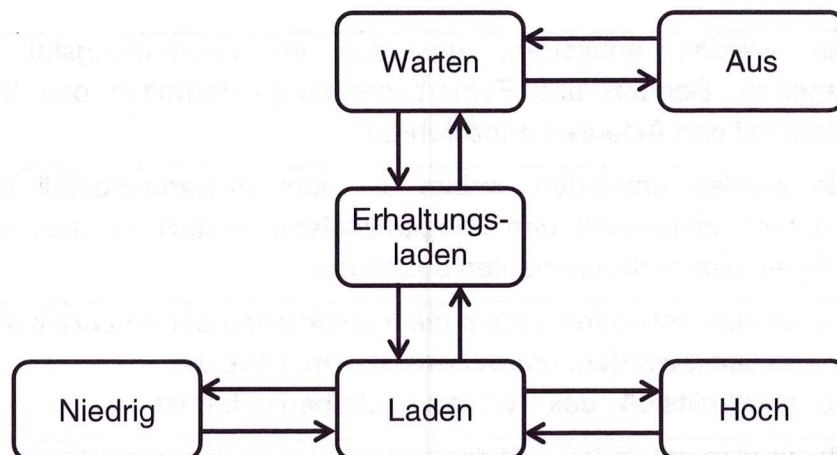
Frage 25

FL-4.2.4

K3

Punkte 1.0

Gegeben sei folgendes Zustandsübergangsdiagramm für die Software eines Batterie-Ladegerätes:



Welche der folgenden Zustandsübergangs-Sequenzen ergibt die höchste Überdeckung der Zustandsübergänge?

Wählen Sie genau EINE korrekte Option aus!

a)	Aus → Warten → Aus → Warten → Erhaltungsladen → Laden → Hoch → Laden → Niedrig	☐
b)	Warten → Erhaltungsladen → Warten → Aus → Warten → Erhaltungsladen → Laden → Niedrig → Laden	☐
c)	Hoch → Laden → Niedrig → Laden → Erhaltungsladen → Warten → Erhaltungsladen → Warten → Erhaltungsladen	☐
d)	Warten → Erhaltungsladen → Laden → Hoch → Laden → Erhaltungsladen → Warten → Aus → Warten	☐

Frage 26	FL-4.2.5	K2	Punkte 1.0
----------	----------	----	------------

Welche der folgenden Aussagen beschreibt AM BESTEN, wie Testfälle aus Anwendungsfällen entworfen werden?

Wählen Sie genau EINE korrekte Option aus!

a)	Testfälle werden entworfen, um das im Anwendungsfall definierte grundlegende, Sonder- und Fehlerbehandlungs-Verhalten des Systems in Interaktion mit den Akteuren auszuführen.	☐
b)	Testfälle werden entworfen, indem die vom Anwendungsfall betroffenen Komponenten identifiziert und Integrationstests erstellt werden, welche die Interaktionen dieser Komponenten ausführen.	☐
c)	Testfälle werden entworfen, indem die Interaktionen der Akteure mit dem System analysiert werden, um sicherzustellen, dass die Benutzungsschnittstelle des Systems leicht bedienbar ist.	☐
d)	Testfälle werden entworfen, mit denen alle Entscheidungspunkte im Geschäftsprozess des Anwendungsfalls ausgeführt werden, um eine 100%-ige Entscheidungsüberdeckung zu erreichen.	☐

Frage 27	FL-4.3.1	K2	Punkte 1.0
----------	----------	----	------------

Welche der folgenden Beschreibungen der Anweisungsüberdeckung ist korrekt?

Wählen Sie genau EINE korrekte Option aus!

a)	Die Anweisungsüberdeckung ist ein Maß für die Anzahl der Quellcodezeilen (ohne Kommentare), die im Test ausgeführt wurden.	☐
b)	Die Anweisungsüberdeckung ist ein Maß für den prozentualen Anteil der ausführbaren Anweisungen im Quellcode, die im Test ausgeführt wurden.	☐
c)	Die Anweisungsüberdeckung ist ein Maß für den prozentualen Anteil der Quellcodezeilen (ohne Kommentare), die im Test ausgeführt wurden.	☐
d)	Die Anweisungsüberdeckung ist ein Maß für die Anzahl der ausführbaren Anweisungen im Quellcode, die im Test ausgeführt wurden.	☐

Frage 28	FL-4.3.2	K2	Punkte 1.0
----------	----------	----	------------

Welche der folgenden Beschreibungen der Entscheidungsüberdeckung ist zutreffend?

Wählen Sie genau EINE korrekte Option aus!

a)	Die Entscheidungsüberdeckung ist ein Maß für den prozentualen Anteil möglicher Pfade durch den Quellcode, die im Test ausgeführt wurden.	☐
b)	Die Entscheidungsüberdeckung ist ein Maß für den prozentualen Anteil der Geschäftsabläufe durch die Komponente, die im Test ausgeführt wurden.	☐
c)	Die Entscheidungsüberdeckung ist ein Maß für die „IF-Anweisungen“ im Quellcode, die im Test sowohl mit dem Ergebnis „WAHR“ als auch mit „FALSCH“ ausgeführt wurden.	☐
d)	Die Entscheidungsüberdeckung ist ein Maß für den Anteil der Entscheidungsergebnisse, die im Test ausgeführt wurden.	☐

Frage 29	FL-4.4.1	K2	Punkte 1.0
----------	----------	----	------------

Welche der folgenden Optionen beschreibt AM BESTEN das Konzept der intuitiven Testfallermittlung?

Wählen Sie genau EINE korrekte Option aus!

a)	Die intuitive Testfallermittlung erfordert, dass Sie sich vorstellen, der Benutzer des Testobjekts zu sein und dass Sie Fehler erraten, die der Benutzer bei der Interaktion damit machen könnte.	☐
b)	Die intuitive Testfallermittlung bezieht Ihre persönlichen Entwicklungserfahrungen und die Fehler mit ein, die Sie als Entwickler gemacht haben.	☐
c)	Die intuitive Testfallermittlung verwendet Ihre Kenntnisse und Erfahrungen mit Fehlerzuständen, die in der Vergangenheit gefunden wurden, sowie mit typischen Fehlhandlungen von Entwicklern.	☐
d)	Die intuitive Testfallermittlung erfordert, dass Sie die Entwicklungsaufgabe schnell selbst wiederholen, um die Art von Fehlern zu identifizieren, die Entwickler dabei möglicherweise machen könnten.	☐

**Fragen zum Thema
"Testmanagement"**

Frage 30	FL-5.1.1	K2	Punkte 1.0
----------	----------	----	------------

Welche der folgenden Aussagen beschreibt am BESTEN einen Vorteil von unabhängigem Testen?

Wählen Sie genau EINE korrekte Option aus!

a)	Die Verwendung eines unabhängigen Testteams erlaubt dem Projektmanagement die Verantwortung für die Qualität des finalen Arbeitsergebnisses auf das Testteam zu übertragen. Somit ist jedem bewusst, dass die Qualität in der Gesamtverantwortung des Testteams liegt.	☐
b)	Wenn ein Testteam außerhalb der Organisation zur Verfügung gestellt werden kann, hat dies deutliche Vorteile, da dieses externe Team nicht so leicht von den Bedenken des Projektmanagements und der Notwendigkeit der Einhaltung strenger Lieferfristen beeinflusst wird.	☐
c)	Ein unabhängiges Testteam kann vollkommen separat von den Entwicklern arbeiten, muss sich nicht von sich ändernden Projektanforderungen ablenken lassen und kann die Kommunikation mit den Entwicklern auf das Verfassen von Fehlerberichten über das Fehlermanagementsystem beschränken.	☐
d)	Wenn Spezifikationen Mehrdeutigkeiten und/oder Inkonsistenzen enthalten, werden Annahmen zu deren Interpretation getroffen. Ein unabhängiger Tester kann hilfreich sein, um die vom Entwickler getroffenen Annahmen und vorgenommenen Interpretationen in Frage zu stellen.	☐

Frage 31	FL-5.1.2	K1	Punkte 1.0
----------	----------	----	------------

Welche der folgenden Aufgaben wird AM WAHRSCHEINLICHSTEN vom Testmanager ausgeführt?

Wählen Sie genau EINE korrekte Option aus!

a)	Erstellen von Testabschlussberichten auf der Grundlage der während des Tests gesammelten Informationen.	☐
b)	Tests prüfen, die von anderen entwickelt wurden.	☐
c)	Testdaten vorbereiten und beschaffen.	☐
d)	Anforderungen, Spezifikationen und Modelle auf Testbarkeit analysieren, prüfen und beurteilen.	☐

Frage 32	FL-5.2.3	K2	Punkte 1.0
----------	----------	----	------------

Gegeben seien die folgenden Beispiele für Eingangs- und Endekriterien:

1. Das ursprüngliche Testbudget von 30.000 US-Dollar wurde ausgegeben.
2. 96% der geplanten Tests wurden ausgeführt.
3. Die Testumgebung für den Performanz-Test wurde entworfen, eingerichtet und verifiziert.
4. Derzeit gibt es keine kritischen Fehlerzustände und zwei Fehlerzustände mit hoher Priorität.
5. Die Designspezifikationen wurden einem Review unterzogen und nachgebessert.
6. Die Komponente für die Berechnung des Steuersatzes hat die Unit-Tests bestanden.

Welche der folgenden Kombinationen kategorisiert sie AM BESTEN als Eingangs- und Endekriterien?

Wählen Sie genau EINE korrekte Option aus!

a)	Eingangskriterien – 5, 6; Endekriterien – 1, 2, 3, 4	☐
b)	Eingangskriterien – 2, 3, 6; Endekriterien – 1, 4, 5	☐
c)	Eingangskriterien – 1, 3; Endekriterien – 2, 4, 5, 6	☐
d)	Eingangskriterien – 3, 5, 6; Endekriterien – 1, 2, 4	☐

Frage 33	FL-5.2.4	K3	Punkte 1.0
----------	----------	----	------------

Gegeben sind die folgenden Prioritäten und Abhängigkeiten von Testfällen:

Testfall	Priorität	Technische Abhängigkeit von:	Logische Abhängigkeit von:
TF1	Hoch	TF4	
TF2	Niedrig		
TF3	Hoch		TF4
TF4	Mittel		
TF5	Niedrig		TF2
TF6	Mittel	TF5	

Welcher der folgenden Testausführungspläne berücksichtigt AM BESTEN die Prioritäten sowie technische und logische Abhängigkeiten?

Wählen Sie genau EINE korrekte Option aus!

a)	TF1 – TF3 – TF4 – TF6 – TF2 – TF5	☐
b)	TF4 – TF3 – TF1 – TF2 – TF5 – TF6	☐
c)	TF4 – TF1 – TF3 – TF5 – TF6 – TF2	☐
d)	TF4 – TF2 – TF5 – TF1 – TF3 – TF6	☐

Frage 34	FL-5.2.6	K2	Punkte 1.0
----------	----------	----	------------

Welche der folgenden Aussagen über Testschätzverfahren ist korrekt?

Wählen Sie genau EINE korrekte Option aus!

a)	Beim metrikbasierten Verfahren basiert die Schätzung auf Testmaßnahmen aus dem Projekt, so dass diese Schätzung erst nach Beginn des Tests verfügbar ist.	☐
b)	Beim expertenbasierten Verfahren empfiehlt eine vom Kunden identifizierte Gruppe von Experten das erforderliche Testbudget.	☐
c)	Beim expertenbasierten Verfahren schätzen die für die verschiedenen Testaktivitäten verantwortlichen Testmanager den erwarteten Testaufwand.	☐
d)	Beim metrikbasierten Ansatz wird ein Durchschnitt der Testkosten, die aus mehreren vergangenen Projekten ermittelt wurden, als Testbudget verwendet.	☐

Frage 35	FL-5.5.1	K1	Punkte 1.0
----------	----------	----	------------

Welche der folgenden Aussagen definiert AM BESTEN die Risikostufe (Höhe des Risikos)?

Wählen Sie genau EINE korrekte Option aus!

a)	Die Risikostufe wird berechnet, indem die Wahrscheinlichkeiten aller Problemsituationen und der daraus resultierende finanzielle Schaden addiert werden.	☐
b)	Die Risikostufe wird geschätzt, indem die Wahrscheinlichkeit einer Bedrohung des Systems multipliziert wird mit der Wahrscheinlichkeit, dass die Bedrohung auftritt und finanzielle Schäden verursacht.	☐
c)	Die Risikostufe wird bestimmt durch eine Kombination der Wahrscheinlichkeit eines unerwünschten Ereignisses und der erwarteten Auswirkung dieses Ereignisses.	☐
d)	Die Risikostufe ist die Summe aller potenziellen Gefahren für ein System multipliziert mit der Summe aller potenziellen Verluste aus diesem System.	☐

Frage 36	FL-5.5.2	K2	Punkte 1.0
----------	----------	----	------------

Welcher der folgenden Aussagen ist AM EHESTEN ein Beispiel für ein Produktrisiko?

Wählen Sie genau EINE korrekte Option aus!

a)	Die erwarteten IT-Sicherheitsmerkmale werden von der Systemarchitektur möglicherweise nicht unterstützt.	☐
b)	Die Entwickler haben möglicherweise nicht die Zeit, alle vom Testteam gefundenen Fehler zu beheben.	☐
c)	Die Testfälle decken die spezifizierten Anforderungen möglicherweise nicht vollständig ab.	☐
d)	Die Umgebung für den Performanztest ist möglicherweise nicht einsatzbereit, bevor das System zur Auslieferung ansteht.	☐

Frage 37	FL-5.5.3	K2	Punkte 1.0
----------	----------	----	------------

Welche der folgenden Aussagen zum Zusammenhang von Produktrisiko und Testmanagement ist am wenigsten sinnvoll?

Wählen Sie genau EINE korrekte Option aus!

a)	Die potenziellen Auswirkungen von IT-Sicherheitsmängeln wurden als besonders hoch eingestuft, so dass IT-Sicherheitstests vor einigen anderen Testaktivitäten priorisiert wurden.	☐
b)	Die Tests haben ergeben, dass die Qualität des Netzwerkmoduls besser ist als erwartet, so dass nun zusätzliche Tests in diesem Bereich durchgeführt werden.	☐
c)	Die Benutzer hatten Probleme mit der Benutzeroberfläche des bisherigen Systems, so dass zusätzliche Usability-Tests für das Ersatzsystem geplant sind.	☐
d)	Die Zeit, die zum Laden von Webseiten benötigt wird, ist entscheidend für den Erfolg der neuen Website, weshalb für dieses Projekt ein Experte für Leistungstests eingesetzt wurde.	☐

Frage 38	FL-5.6.1	K3	Punkte 1.0
----------	----------	----	------------

Sie führen Systemtests für ein Zugbuchungssystem durch. Basierend auf den durchgeführten Testfällen haben Sie festgestellt, dass das System gelegentlich meldet, dass keine Züge verfügbar sind, obwohl dies eigentlich der Fall sein sollte. Sie haben den Entwicklern eine Zusammenfassung des Fehlers und der Version des getesteten Systems zur Verfügung gestellt. Diese erkennen die Dringlichkeit des Fehlers und warten nun darauf, dass Sie weitere Details angeben.

Abgesehen von den bereits aufgeführten Informationen sind folgende weitere Informationen gegeben:

1. Grad der Auswirkung (Schwere) des Fehlers.
2. Identifikation des Testelements.
3. Details der Testumgebung.
4. Dringlichkeit/Priorität für die Behebung.
5. Istergebnisse.
6. Referenz auf die Testfallspezifikation.

Welche dieser Informationen sind **AM SINNVOLLSTEN**, um sie in den Fehlerbericht mit aufzunehmen?

Wählen Sie genau **EINE** korrekte Option aus!

a)	1, 2, 6	☐
b)	1, 4, 5, 6	☐
c)	2, 3, 4, 5	☐
d)	3, 5, 6	☐

**Fragen zum Thema
"Testwerkzeuge"**

Frage 39	FL-6.1.1	K2	Punkte 1.0
----------	----------	----	------------

Gegeben seien folgende Testaktivitäten und Testwerkzeuge:

1. Performanzmessung und dynamische Analyse.
 2. Testdurchführung und Protokollierung.
 3. Management des Testens und Testmittel.
 4. Testentwurf.
-
- A. Werkzeuge für Anforderungsüberdeckung.
 - B. Dynamische Analysewerkzeuge.
 - C. Werkzeuge zur Vorbereitung von Testdaten.
 - D. Fehlermanagementwerkzeuge.

Welcher der folgenden Kombinationen passt am besten zu den Aktivitäten und Tools?

Wählen Sie genau EINE korrekte Option aus!

a)	1 – B, 2 – C, 3 – D, 4 – A	☐
b)	1 – B, 2 – A, 3 – C, 4 – D	☐
c)	1 – B, 2 – A, 3 – D, 4 – C	☐
d)	1 – A, 2 – B, 3 – D, 4 – C	☐

Frage 40	FL-6.2.2	K1	Punkte 1.0
----------	----------	----	------------

Welcher der folgenden Punkte wird AM WAHRSCHEINLICHSTEN als Grund für die Verwendung eines Pilotprojekts zur Einführung eines Werkzeugs in einem Unternehmen verwendet?

Wählen Sie genau EINE korrekte Option aus!

a)	Die Notwendigkeit zu bewerten, wie das Werkzeug zu bestehenden Prozessen und Vorgehensweisen passt und zu bestimmen, was geändert werden muss.	☐
b)	Die Notwendigkeit, die Fähigkeiten zur Testautomatisierung sowie die Trainings-, Mentoring- und Coaching-Bedürfnisse der Tester zu bewerten, die das Werkzeug nutzen werden.	☐
c)	Die Notwendigkeit zu bewerten, ob das Werkzeug die erforderliche Funktionalität bietet und bestehende Testwerkzeuge nicht dupliziert.	☐
d)	Die Notwendigkeit, den Werkzeughersteller zu bewerten in Bezug auf die Schulung und andere Unterstützung, die er anbietet.	☐

ISTQB® Certified Tester Foundation Level 2018 -CORE- Sample Exam Paper ("Probeprüfung"), SET B



Platz für Ihre Notizen:

(Diese werden bei der Korrektur weder gelesen noch bewertet)

**ISTQB® Certified Tester Foundation Level 2018 -CORE-
Sample Exam Paper (“Probeprüfung”), SET B**



Platz für Ihre Notizen:

(Diese werden bei der Korrektur weder gelesen noch bewertet)

**ISTQB® Certified Tester Foundation Level 2018 -CORE-
Sample Exam Paper (“Probeprüfung”), SET B**



Platz für Ihre Notizen:

(Diese werden bei der Korrektur weder gelesen noch bewertet)

**ISTQB® Certified Tester Foundation Level 2018 -CORE-
Sample Exam Paper (“Probeprüfung”), SET B**



Platz für Ihre Notizen:

(Diese werden bei der Korrektur weder gelesen noch bewertet)

Informationen zu den Übungen:

Die Übungen sind zum Teil als Einzelarbeiten mit Ergebnisdiskussion, als Gruppenarbeit mit Ergebnisdiskussion oder als gemeinsame Arbeiten gestaltet.

Nicht alle Übungen werden während des Seminars durchgeführt. Manche Übungen werden als Hausarbeiten aufgegeben oder alternativ durchgeführt – d.h. von zwei möglichen Übungsszenarien wird nur eines ausgewählt.

Zu manchen Übungsthemen wird im Rahmen einer gemeinsamen Diskussion das Übungsergebnis erarbeitet.

Zu allen Übungen gibt es Lösungsbeispiele oder Lösungsansätze in einem separaten Dokument bzw. in einem separaten Bereich dieses Dokumentes.

Version 0.6 (Dezember 2021)

Inhaltsverzeichnis:

	Seite
1. Übungen	1
2. Lösungen	14
3. Normen und Standards	28
4. Quellenverzeichnis	29

I. Grundlagen des Softwaretestens

1. Warum ist Testen notwendig?

Übung I.1: Wie viel Testaufwand ist notwendig?

Aufgabe	▪ Welche Faktoren sind für die Bestimmung des Testaufwands entscheidend?
Zeitrahmen	10 Min. gemeinsame Erarbeitung am Flipchart

Deine Lösung:

I. Grundlagen des Softwaretestens

1. Warum ist Testen notwendig?

Übung I.2: Zusammenfassung von Beispielen

Aufgabe	Welche Beispiele für die folgenden Themen gibt es: ▪ Softwarefehler können verheerende Auswirkungen haben und sehr teuer sein ▪ Durch Fehlhandlungen können Fehlerzustände/Defekte in das Testobjekt kommen, die sich mit Fehlerwirkungen zeigen ▪ Norm ISO 9126 regelt den Begriff der Softwarequalität, funktionale und nicht-funktionale Qualitätsmerkmale bestimmen die Gesamtqualität ▪ Konstruktive Software QS befasst sich mit Fehlervermeidung ▪ Analytische Software QS mit Fehlerfindung,-korrektur und Qualitätssicherung ▪ Für Tests sind vorab Endekriterien zu definieren, mit deren Erfüllung der Test abgeschlossen ist
Zeitraumen	10 Min. gemeinsame Erarbeitung am Flipchart

Deine Lösung:

III. Statischer Test

2. Reviewprozess

Übung III.1: Review Durchführen

Aufgabe	Führe ein Walkthrough für die gegebene Spezifikation durch. Anforderungsspezifikation (Online-Shop): ▪ Bis einschließlich 500 Euro Warenwert gibt es keinen Rabatt. Ab Euro 500 bis 1.000,- gibt es 2,5 % Rabatt. Über € 1.000 bis 5.000 Euro gibt es 5,0 % Rabatt, darüber 8,5 %. ▪ Bis 11 kg Gewicht kostet der Postversand 6 Euro, der Expressversand 12 Euro. Bis einschl. 30 kg Gewicht liegen die Versandpreise bei 11 Euro (normal) und 18 Euro (Express). Über 30 kg liegen die Preise entsprechend bei 17 bzw. 25 Euro. Auch Selbstabholung ist möglich. ▪ Über 1.200 Euro Warenwert erfolgt der normale Paketversand versandkostenfrei. Bei Zahlung per Nachnahme werden 3 Euro Gebühr zusätzlich berechnet (normal wird per Vorkasse gezahlt). Bei über 1.000 Euro Warenwert entfällt auch die Nachnahme-Gebühr.
Zeitraumen	15 Min. Gruppenarbeit

Deine Lösung:

IV. Testverfahren

2. Black-Box-Testverfahren

Übung IV.1: Äquivalenzklassen

Aus der gegebenen Spezifikation (Online-Shop) sind:
Die Eingabewerte zu bestimmen
Äquivalenzklassen für die Eingabe-Wertebereiche des Warenwertes zu definieren
Gültige Äquivalenzklassen
Ungültige Äquivalenzklassen

Aufgabe	Anforderungsspezifikation (Online-Shop): ▪ Bis einschließlich 500 Euro Warenwert gibt es keinen Rabatt. Ab Euro 500 bis 1.000,- gibt es 2,5 % Rabatt. Über € 1.000 bis 5.000 Euro gibt es 5,0 % Rabatt, darüber 8,5 %. ▪ Bis 11 kg Gewicht kostet der Postversand 6 Euro, der Expressversand 12 Euro. Bis einschl. 30 kg Gewicht liegen die Versandpreise bei 11 Euro (normal) und 18 Euro (Express). Über 30 kg liegen die Preise entsprechend bei 17 bzw. 25 Euro. Auch Selbstabholung ist möglich. ▪ Über 1.200 Euro Warenwert erfolgt der normale Paketversand versandkostenfrei. Bei Zahlung per Nachnahme werden 3 Euro Gebühr zusätzlich berechnet (normal wird per Vorkasse gezahlt). Bei über 1.000 Euro Warenwert entfällt auch die Nachnahme-Gebühr.
Zeitraumen	15 Min. Gruppenarbeit

Deine Lösung:

IV. Testverfahren

2. Black-Box-Testverfahren

Übung IV.2: Entscheidungstabellentest

Aufgabe	Der Besuch im Zoo. Als Tierpfleger im Krefelder Zoo bist Du für die Betreuung der Nashornbabys <i>Navu</i> und <i>Thabo</i> zuständig. Es gibt einige Regeln für die Besuche: ▪ Besucher mit ansteckenden Krankheiten dürfen gar nicht zu den Tieren ▪ Besucher außerhalb der Besuchszeiten dürfen nur für max. 15 Min. nach vorheriger Anmeldung zu den Tieren ▪ Besucher in großen Gruppen dürfen nur mit einem Pfleger zu den Tieren Die Regeln sind der Wichtigkeit nach angeordnet. In Abhängigkeit von den Ursachen werden die (ggf. kombinierten) Wirkungen ermittelt.
Zeitraumen	20 Min. gemeinsame Erarbeitung am Flipchart

Deine Lösung:

IV. Testverfahren

2. Black-Box-Testverfahren

Übung IV.3: Zustandsbasierter Test

Aufgabe	▪ Ein Zustandsübergangsdiagramm für die möglichen unterschiedlichen Familienstände aus Sicht eines deutschen Standesamtes wird erarbeitet ▪ Die Umformung des Diagrammes in einen Zustandsbaum wird gemeinsam durchgeführt
Zeitraumen	10 Min. Gruppenarbeit

Deine Lösung:

IV. Testverfahren

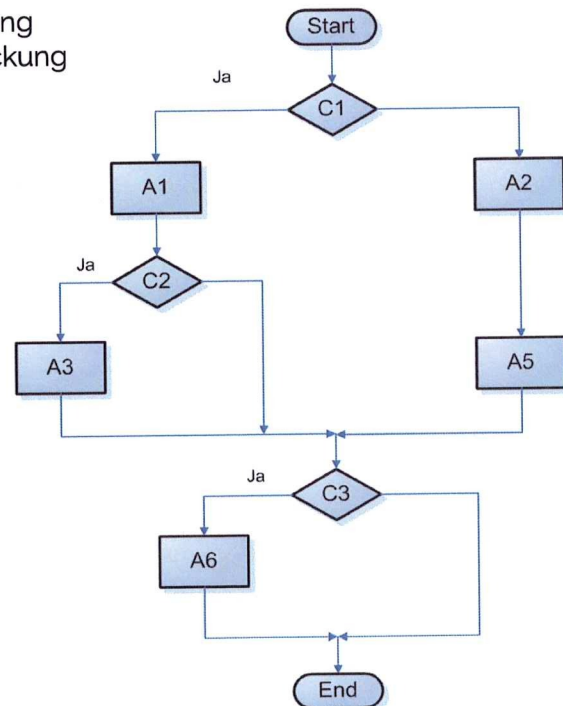
3. White-Box-Testverfahren

Übung IV.4: Abdeckungsmessung, Beispiel 1

Aufgabe

Aus gegebenem Kontrollflussgraphen bzw. Pseudo Code sind:
die minimale Anzahl Testfälle für 100% Überdeckung zu bestimmen

- Anweisungsüberdeckung
- Entscheidungsüberdeckung
- Pfadüberdeckung



Zeitraumen

10 Min. gemeinsame Erarbeitung am Flipchart

Deine Lösung:

Anweisungsüberdeckung	
Entscheidungsüberdeckung	
Pfadüberdeckung	

IV. Testverfahren

3. White-Box-Testverfahren

Übung IV.5: Abdeckungsmessung, Beispiel 2

Aufgabe	Erstelle für das in Pseudocode dargestellte Programm einen Kontrollflussgraphen. Bilde die minimal notwendige Anzahl von Testfällen, die 100% Überdeckungsgrad erzeugen für: ▪ Anweisungsüberdeckung ▪ Entscheidungsüberdeckung ▪ Pfadüberdeckung <pre>1 Start 2 IF V1 > 0 THEN 3 DO A1 4 IF V2 < 5 THEN 5 do A2 6 ELSE 7 do A3 8 END-IF 9 END-IF 10 IF V3 = V1 THEN 11 do A4 12 END-IF 13 End</pre>
Zeitraumen	10 Min. Einzelarbeit, 15 Min. Ergebnisdiskussion

Deine Lösung:

Anweisungsüberdeckung	
Entscheidungsüberdeckung	
Pfadüberdeckung	

IV. Testverfahren

3. White-Box-Testverfahren

Übung IV.6: Abdeckungsmessung, Beispiel 3.

Aufgabe	Erstelle für das in Pseudocode dargestellte Programm einen Kontrollflussgraphen. Bilde die minimal notwendige Anzahl von Testfällen, die 100% Überdeckungsgrad erzeugen für: ▪ Anweisungsüberdeckung ▪ Entscheidungsüberdeckung ▪ Pfadüberdeckung <pre>1 Start 2 if V1 > 0 then 3 do A1 4 if V2 < 5 then 5 do A2 6 until V3 = 0 7 endif 8 Endif 9 End</pre>
Zeitraumen	10 Min. Einzelarbeit, 15 Min. Ergebnisdiskussion

Deine Lösung:

Anweisungsüberdeckung	
Entscheidungsüberdeckung	
Pfadüberdeckung	

IV. Testverfahren

3. White-Box-Testverfahren

Übung V.7: Abdeckungsmessung, Beispiel 4

Aufgabe	Erstelle für das in Pseudocode dargestellte Programm einen Kontrollflussgraphen. Bilde die minimal notwendige Anzahl von Testfällen, die 100% Überdeckungsgrad erzeugen für: ▪ Anweisungsüberdeckung ▪ Entscheidungsüberdeckung ▪ Pfadüberdeckung <pre> 1 Sortier-Programm (array DATEI, int EOF) 2 boolean Sort-Ende 3 int i, Hilf 4 Sort-Ende = false 5 WHILE (Sort-Ende = false AND EOF > 1) do 6 Sort-Ende = true 7 FOR (i=1; i<=(EOF-1); i=i+1) do 8 IF DATEI[i] > DATEI[i+1] then 9 Sort-Ende=false 10 Hilf = DATEI[i] 11 DATEI[i+1] = DATEI[i] 12 DATEI[i] = Hilf 13 END-IF 14 END-FOR 15 END-WHILE 16 return DATEI </pre>
Zeitraumen	10 Min. Einzelarbeit, 15 Min. Ergebnisdiskussion

Deine Lösung:

Anweisungsüberdeckung	
Entscheidungsüberdeckung	
Pfadüberdeckung	

IV. Testverfahren

3. White-Box-Testverfahren

Übung 27 aus TrainingQuestionsGerman-V02_003a

Aufgabe	Welches ist die minimale Anzahl der benötigten Testfälle, um 100% Entscheidungsüberdeckung zu erreichen. Neue Übung mit Schleife
Zeitraumen	10 Min. Einzelarbeit, 15 Min. Ergebnisdiskussion

```
1 Start
2 Input Number_of_Coins
3 Total = 0
4 While Number_of_Coins > 0
5     Input Value_of_Coins
6     Total = Total + Value_of_Coin
7     Number_of_Coins = Number_of_Coins - 1
8 endloop
9 Print "Ihre Münzen haben den Wert: & Total"
10 End
```

Deine Lösung:

Anweisungsüberdeckung	
Entscheidungsüberdeckung	
Pfadüberdeckung	

V. Testmanagement

6. Fehlermanagement

Übung V.1: Fehlermanagement

Aufgabe	Du möchtest am Geldautomaten mit Deiner EC-Karte von Deinem Girokonto Geld abheben. Der Geldautomat hat Deine PIN-Eingabe als richtig quittiert und Dich nach dem gewünschten Betrag gefragt. Du hast den Betrag ausgewählt. Nach einer längeren Pause geht der Geldautomat außer Betrieb und Du hast Deine EC-Karte zurückerhalten, aber es erfolgte keine Auszahlung. Bei der Umsatzabfrage Deines Kontos stellst Du fest, dass der Betrag trotzdem Deinem Konto belastet wurde. Schreibe einen Fehlerbericht über den gesamten Fehlerverlauf.
Zeitraumen	10 Min. Einzelarbeit, 15 Min. Ergebnisdiskussion

Deine Lösung:

INFORMATIONEN ZU DEN LÖSUNGEN/LÖSUNGSANSÄTZE:

Die hier angeführten Lösungsansätze stellen meist nur eine der möglichen Lösungen dar. In vielen Fällen sind andere Lösungen auch denkbar und sinnvoll.

Alle gefundenen Lösungsansätze sind stark von der jeweiligen Erfahrung des Bearbeiters geprägt.

Zu Diskussions-Übungen werden keine Lösungen präsentiert; es werden hier jedoch einige typische Argumente/Gesprächspunkte angeführt, die in den Diskussionen mit den Schulungsteilnehmern immer wieder auftraten.

Es gibt auch Übungen ohne niedergeschriebene Lösung.

I. Grundlagen des Softwaretestens

1. Warum ist Testen notwendig?

Übung I.1: Wie viel Testaufwand ist notwendig?

Aufgabe

- Welche Faktoren sind für die Bestimmung des Testaufwands entscheidend?

Zeitraumen

10 Min. gemeinsame Erarbeitung am Flipchart

Deine Lösung:

I. Grundlagen des Softwaretestens

1. Warum ist Testen notwendig?

Übung I.2: Zusammenfassung von Beispielen

Aufgabe	Welche Beispiele für die folgenden Themen gibt es: ▪ Softwarefehler können verheerende Auswirkungen haben und sehr teuer sein ▪ Durch Fehlhandlungen können Fehlerzustände/Defekte in das Testobjekt kommen, die sich mit Fehlerwirkungen zeigen ▪ Norm ISO 9126 regelt den Begriff der Softwarequalität, funktionale und nicht-funktionale Qualitätsmerkmale bestimmen die Gesamtqualität ▪ Konstruktive Software QS befasst sich mit Fehlervermeidung ▪ Analytische Software QS mit Fehlerfindung, -korrektur und Qualitätssicherung ▪ Für Tests sind vorab Endekriterien zu definieren, mit deren Erfüllung der Test abgeschlossen ist
Zeitraumen	10 Min. gemeinsame Erarbeitung am Flipchart

Deine Lösung:

III. Testverfahren

2. Black-Box-Testverfahren

Übung III.1: Review durchführen

Aufgabe	Führe ein Walkthrough für die gegebene Spezifikation durch. Anforderungsspezifikation (Online-Shop): ▪ Bis einschließlich 500 Euro Warenwert gibt es keinen Rabatt. Ab Euro 500 bis 1.000,- gibt es 2,5 % Rabatt. Über € 1.000 bis 5.000 Euro gibt es 5,0 % Rabatt, darüber 8,5 %. ▪ Bis 11 kg Gewicht kostet der Postversand 6 Euro, der Expressversand 12 Euro. Bis einschl. 30 kg Gewicht liegen die Versandpreise bei 11 Euro (normal) und 18 Euro (Express). Über 30 kg liegen die Preise entsprechend bei 17 bzw. 25 Euro. Auch Selbstabholung ist möglich. ▪ Über 1.200 Euro Warenwert erfolgt der normale Paketversand versandkostenfrei. Bei Zahlung per Nachnahme werden 3 Euro Gebühr zusätzlich berechnet (normal wird per Vorkasse gezahlt). Bei über 1.000 Euro Warenwert entfällt auch die Nachnahme-Gebühr.
Zeitraumen	15 Min. Gruppenarbeit

Lösung:

Neben allgemeine Bemerkungen können u.a. folgende spezifische Themen diskutiert, bzw. als Befund angemerkt werden:

Zum ersten Abschnitt:

- *ist Null Euro ein gültiger Warenwert, was ist mit negativen Werten (mehrdeutig, fehlende Info)*
 - *„Bis 500 Euro“ ist das einschließlich oder nicht (mehrdeutig)*
 - *mal wird „bis“ verwendet, mal „bis einschließlich“ – gibt's da Unterschiede? (mehrdeutig, gleiche Sachen erfordern gleiche Bezeichner)*
 - *unterschiedliche Währungsbezeichnungen: EUR, Euro, € (gleiche Sachen erfordern gleiche Bezeichner)*
 - *ist „Postversand“ und „normaler Paketversand“ dasselbe? (mehrdeutig, gleiche Sachen erfordern gleiche Bezeichner)*
- was ist das kleinste Inkrement (ein Euro? ein Cent?). Das ist aus der Schreibweise nicht ableitbar: 500 1000,- 1000,00 (fehlende Info)*

Zum zweiten Abschnitt:

- was ist das kleinste Gewichtsinkrement - 1 Gramm, 100 Gramm? (fehlende Info)
- mal wird „bis“ verwendet, mal „bis einschließlich“ – gibt's da Unterschiede? (mehrdeutig, gleiche Sachen erfordern gleiche Bezeichner)
- wie hoch sind Versandkosten bei Selbstabholung? (mehrdeutig, fehlende Info)

Zum dritten Abschnitt:

- hier kann schon eine kleine Entscheidungstabelle vorgestellt werden

IV. Testverfahren

2. Black-Box-Testverfahren

Übung IV.1: Äquivalenzklassen

Aus der gegebenen Spezifikation (Online-Shop) sind:

Die Eingabewerte zu bestimmen

Äquivalenzklassen für die Eingabe-Wertebereiche des Warenwertes zu definieren

Gültige Äquivalenzklassen

Ungültige Äquivalenzklassen

Lösung:

ÄK	Warenwert	Versand	Rabatt-%
+	$0 < x \leq 500$	Versand	./.
+	$500 < x \leq 1.000$	Versand	2,5 %
+	$1.000 < x \leq 1.200$	Versand	5 %
+	$1.200 < x \leq 5.000$	./.	5 %
+	$5.000 < x \leq \infty$	./.	8,5 %
-	$x \leq 0$		
-	x nicht numerisch		

Was fehlt?

Die „Spezifikation“ wird untersucht auf mangelhafte Angaben wie z.B. fehlende Untergrenzen, ungenaue Grenzangaben, Größe der Beträge für Warenwert (double?), Nachkommastellen, Rundung, größter Warenwert. Darüber hinaus wird durch die Einzelarbeit deutlich, welche Annahmen intuitiv getroffen werden, die dort gar nicht spezifiziert sind. Kritischer Umgang mit Spezifikationen und wie die ÄK dabei wesentlich unterstützen.

IV. Testverfahren

2. Black-Box-Testverfahren

Übung IV.2: Entscheidungstabellentest

Aufgabe	Der Besuch im Zoo. Als Tierpfleger im Krefelder Zoo bist Du für die Betreuung der Nashornbabys <i>Navu</i> und <i>Thabo</i> zuständig. Es gibt einige Regeln für die Besuche: ▪ Besucher mit ansteckenden Krankheiten dürfen gar nicht zu den Tieren ▪ Besucher außerhalb der Besuchszeiten dürfen nur für max. 15 Min. nach vorheriger Anmeldung zu den Tieren ▪ Besucher in großen Gruppen dürfen nur mit einem Pfleger zu den Tieren Die Regeln sind der Wichtigkeit nach angeordnet. In Abhängigkeit von den Ursachen werden die (ggf. kombinierten) Wirkungen ermittelt.
Zeitraumen	20 Min. gemeinsame Erarbeitung am Flipchart

Lösung:

U	ansteckende Krankheit	J	N	N	N	N
U	außerhalb Besuchszeit + Anm.	-	J	N	J	N
U	große Gruppe	-	N	J	J	N
W	kein Besuch	J	N	N	N	N
W	Besuchszeit max. 5 Min.	N	J	N	J	N
W	mit Pfleger	N	N	J	J	N
W	Normalbesuch	N	N	N	N	J

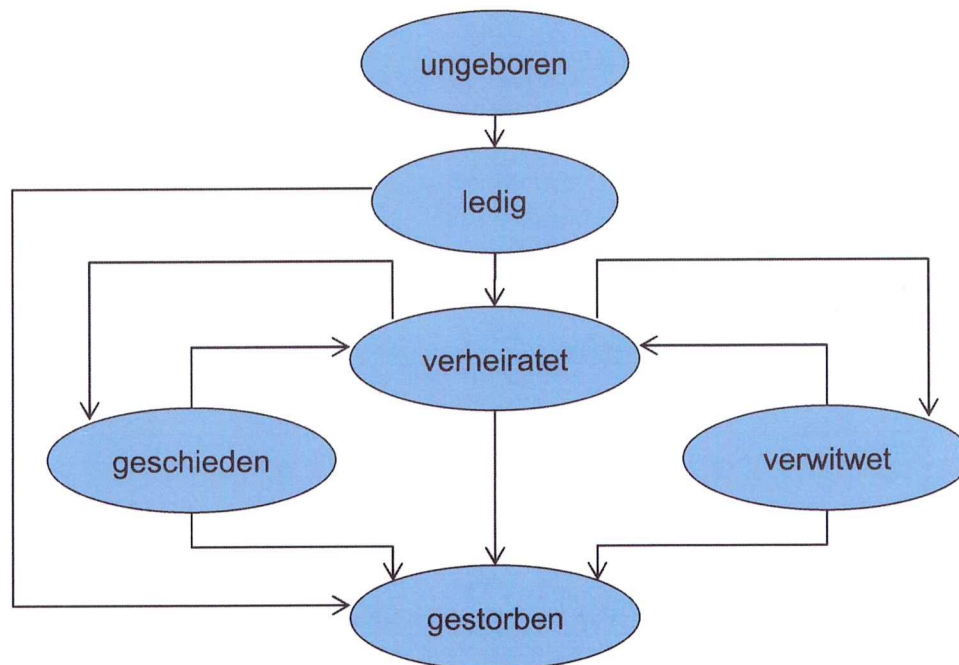
IV. Testverfahren

2. Black-Box-Testverfahren

Übung IV.3: Zustandsbasierter Test

Aufgabe	10 Min. Gruppendiskussion zur gemeinsamen Erstellung (Flip-Chart) des Zustandsdiagramms über die Zustände, die ein Mensch aus Sicht eines Standesbeamten im Laufe seines Lebens einnehmen kann (ledig, verheiratet, geschieden, verwitwet). Aufgabe: Ermitteln sie für das dargestellte Zustandsübergangsdiagramm die Testfälle: ▪ direkt ▪ durch Umwandlung in einen Zustandsbaum
Zeitraumen	10 Min. Gruppenarbeit

Lösung:



IV. Testverfahren

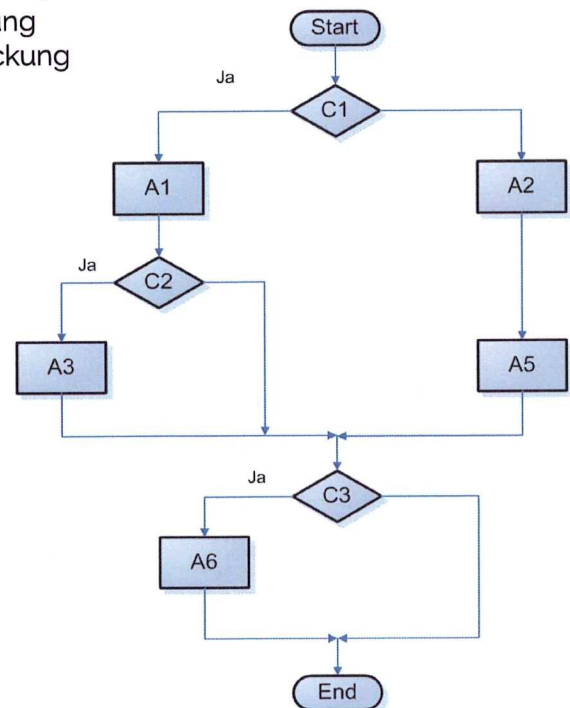
3. White-Box-Testverfahren

Übung IV.4: Abdeckungsmessung, Beispiel 1

Aufgabe

Aus gegebenem Kontrollflussgraphen bzw. Pseudo Code sind:
die minimale Anzahl Testfälle für 100% Überdeckung zu bestimmen

- Anweisungsüberdeckung
- Entscheidungsüberdeckung
- Pfadüberdeckung



Zeitraumen

10 Min. gemeinsame Erarbeitung am Flipchart

Lösung:

Anweisungsüberdeckung	2
Entscheidungsüberdeckung	3
Pfadüberdeckung	6

IV. Testverfahren

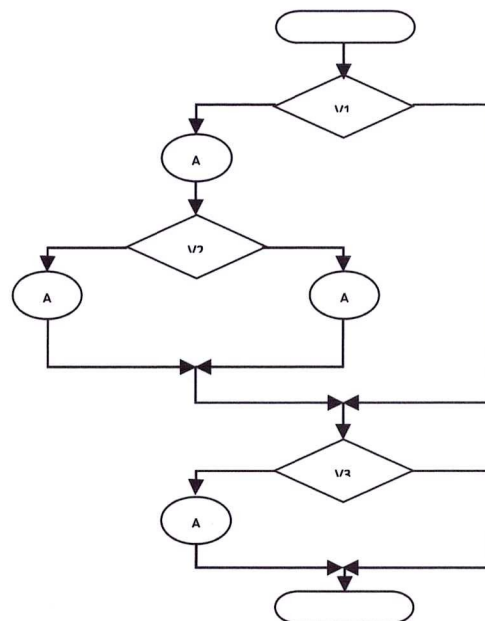
3. White-Box-Testverfahren

Übung IV.5: Abdeckungsmessung, Beispiel 2

Aufgabe	Erstelle für das in Pseudocode dargestellte Programm einen Kontrollflussgraphen. Bilde die minimal notwendige Anzahl von Testfällen, die 100% Überdeckungsgrad erzeugen für: - Anweisungsüberdeckung - Entscheidungsüberdeckung - Pfadüberdeckung
Zeitraumen	10 Min. Einzelarbeit, 15 Min. Ergebnisdiskussion

```

1  Start
2  IF V1 > 0 THEN
3    DO A1
4    IF V2 < 5 THEN
5      do A2
6    ELSE
7      do A3
8    END-IF
9  END-IF
10 IF V3 = V1 THEN
11   do A4
12 END-IF
13 End
  
```



Lösung:

Anweisungsüberdeckung	2
Entscheidungsüberdeckung	3
Pfadüberdeckung	6

IV. Testverfahren

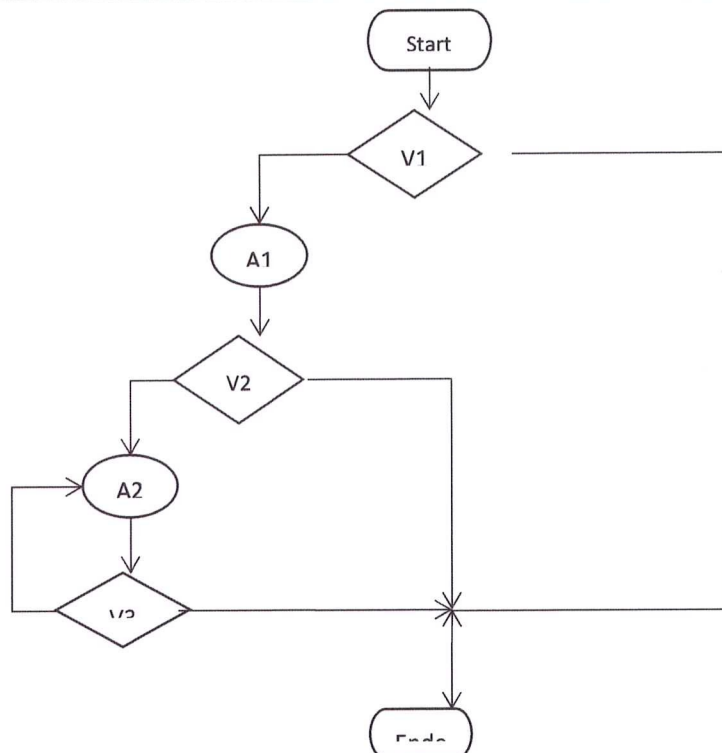
3. White-Box-Testverfahren

Übung IV.6: Abdeckungs-messung, Beispiel 3

Aufgabe	Erstelle für das in Pseudocode dargestellte Programm einen Kontrollflussgraphen. Bilden Sie die minimal notwendige Anzahl von Testfällen, die 100% Überdeckungsgrad erzeugen für: - Anweisungsüberdeckung - Entscheidungsüberdeckung - Pfadüberdeckung
Zeitraumen	10 Min. Einzelarbeit, 15 Min. Ergebnisdiskussion

```

1  Start
2  if V1 > 0 then
3      do A1
4          if V2 < 5 then
5              do A2
6                  until V3 = 0
7          endif
8      endif
9  End
    
```



Lösung:

Anweisungsüberdeckung	1
Entscheidungsüberdeckung	3
Pfadüberdeckung	≥ 4

IV. Testverfahren

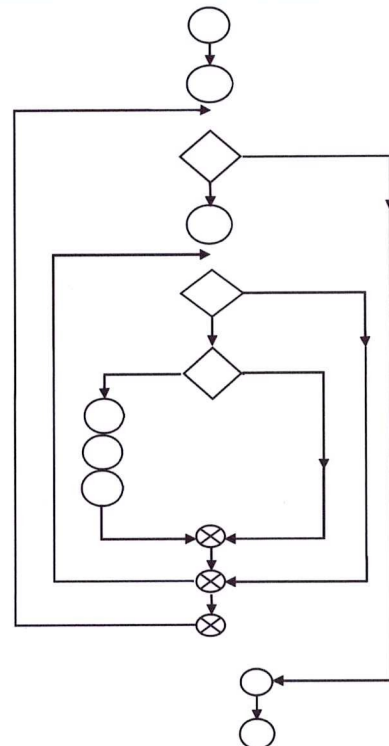
3. White-Box-Testverfahren

Übung IV.7: Abdeckungsmessung, Beispiel 4

Aufgabe	Erstelle für das in Pseudocode dargestellte Programm einen Kontrollflussgraphen. Bilde die minimal notwendige Anzahl von Testfällen, die 100% Überdeckungsgrad erzeugen für: - Anweisungsüberdeckung - Entscheidungsüberdeckung - Pfadüberdeckung
Zeitraumen	10 Min. Einzelarbeit, 15 Min. Ergebnisdiskussion

```

1  Sortier-Programm (array DATEI, int EOF)
2  boolean Sort-Ende
3  int i, Hilf
4  Sort-Ende = false
5  WHILE (Sort-Ende = false AND EOF > 1) do
6    Sort-Ende = true
7    FOR (i=1; i<=(EOF-1); i=i+1) do
8      IF DATEI[i] > DATEI[i+1] then
9        Sort-Ende=false
10       Hilf = DATEI[i]
11       DATEI[i+1] = DATEI[i]
12       DATEI[i] = Hilf
13     END-IF
14   END-FOR
15 END-WHILE
16 return DATEI
    
```



Lösung:

Anweisungsüberdeckung	1
Entscheidungsüberdeckung	1
Pfadüberdeckung	≥ 1

IV. Testverfahren

3. White-Box-Testverfahren

Übung 27 aus TrainingQuestionsGerman-V02_003a

Aufgabe

Welches ist die minimale Anzahl der benötigten Testfälle, um 100% Entscheidungsüberdeckung zu erreichen.

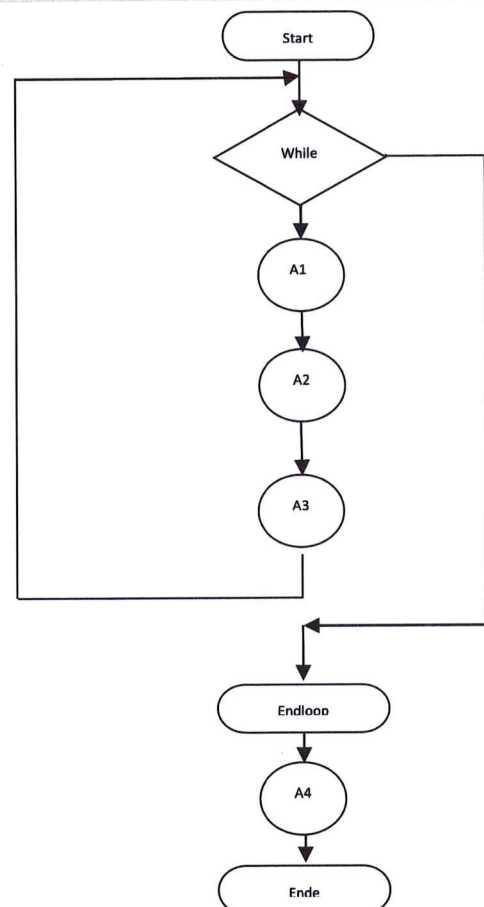
Neue Übung mit Schleife

Zeitraumen

10 Min. Einzelarbeit, 15 Min. Ergebnisdiskussion

```

1 Start
2 Input Number_of_Coins
3 Total = 0
4 While Number_of_Coins > 0
5   Input Value_of_Coin
6   Total = Total + Value_of_Coin
7   Number_of_Coins = Number_of_Coins + 1
8 endloop
9 Print "Ihre Münzen haben der Wert: & Total"
10 End
    
```



Lösung:

Anweisungsüberdeckung	1
Entscheidungsüberdeckung	1
Pfadüberdeckung	>= 1

V. Testmanagement**6. Fehlermanagement****Übung V.1: Fehlermanagement**

Aufgabe	Du möchtest am Geldautomaten mit Deiner EC-Karte von Deinem Girokonto Geld abheben. Der Geldautomat hat Deine PIN-Eingabe als richtig quittiert und Dich nach dem gewünschten Betrag gefragt. Du hast den Betrag ausgewählt. Nach einer längeren Pause geht der Geldautomat außer Betrieb und Du hast Deine EC-Karte zurückerhalten, aber es erfolgte keine Auszahlung. Bei der Umsatzabfrage Deines Kontos stellst Du fest, dass der Betrag trotzdem Deinem Konto belastet wurde. Schreibe einen Fehlerbericht über den gesamten Fehlerverlauf.
Zeitraumen	10 Min. Einzelarbeit, 15 Min. Ergebnisdiskussion

Lösung:

- *ISO 9000:2015*
Qualitätsmanagement – Begriffe
- *ISO 9000-3:2018*
Quality management and quality assurance standards -- Part 3: Leitfaden für die Anwendung von ISO 9001 auf die Entwicklung, Lieferung und Wartung von Software Qualitätsmanagement- und Qualitätssicherungsnormen
- *ISO 20246:2017 (vorheriger Standard: IEEE 1028)*
Software and systems engineering -- Work product reviews
- *ISO 24765-2017 (vorheriger Standard: IEEE 610.12)*
Systems and software engineering--Vocabulary
- *ISO 25010:2011 (vorheriger Standard ISO 9126)*
Systems and software engineering -- Systems and software Quality Requirements and Evaluation (SQuaRE) -- System and software quality models
- *ISO 25051-2014*
Software engineering -- Systems and software Quality Requirements and Evaluation (SQuaRE) -- Requirements for quality of Ready to Use Software Product (RUSP) and instructions for testing
- *ISO 29119:2013 (vorheriger Standard: IEEE 829)*
- *ISO 29119-1:2013 Software and systems engineering -- Software testing -- Part 1: Concepts and definitions*
- *ISO 29119-2:2013 Software and systems engineering -- Software testing -- Part 2: Test processes*
- *ISO 29119-3:2013 Software and systems engineering -- Software testing -- Part 3: Test documentation*
- *ISO 29119-4:2015 Software and systems engineering -- Software testing -- Part 4: Test techniques*
- *ISO 29119-5:2016 Software and systems engineering -- Software testing -- Part 5: Keyword-Driven Testing*
- *ISO 29148-2018*
Systems and software engineering - Life cycle processes - Requirements engineering
- *ISO 30130:2016 Software engineering -- Capabilities of software testing tools*
- *IEEE 730-1989*
Standard for Configuration Management in Systems and Software Engineering
- *IEEE 828-2012*
Standard for Software Configuration Management Plans
- *IEEE 982.1-2005*
Standard Standard Dictionary of Measures of the Software Aspects of Dependability
- *IEEE 1012-2016*
Standard for System, Software, and Hardware Verification and Validation
- *IEEE 1044-2009*
Standard Classification for Software Anomalies
- *IEEE 1061-1998*
Standard for a Software Quality Metrics Methodology

- *Testen in der Finanzwelt*
Bochynek, N.; Díaz, J.
- *Basiswissen modellbasierter Test, Aus- und Weiterbildung zum ISTQB® Foundation Level – Certified Model-Based Tester, 2. Auflage*
Winter, M.; Roßner, Th.; Brandes, Ch.; Götz, H. – dpunkt.verlag
- *Basiswissen Softwaretest (3. Auflage)*
Spillner, A. ; Linz, T. – dpunkt.verlag
- *Basiswissen Testautomatisierung; Konzepte, Methoden und Techniken (2. Auflage)*
Bucsics, Th.; Baumgartner, M.; Seidl, R.; Gwihs, St. – dpunkt.verlag
- *Der Integrationstest – Von Entwurf und Architektur zur Komponenten- und Systemintegration*
Winter, M.; Ekssir-Monfared, M.; Sneed, H. M.; Seidl, R.; Borner, L. – Carl Hanser Verlag
- *Der Systemtest – Von den Anforderungen zum Qualitätsnachweis, 3. Auflage*
Sneed, H. M.; Baumgartner, M.; Seidl, R. – Carl Hanser Verlag
- *Explore It! Wie Softwareentwickler und Tester mit explorativem Testen Risiken reduzieren und Fehler aufdecken*
Hendrickson, E. – dpunkt.verlag
- *Lean Testing für C++-Programmierer – angemessen statt aufwendig testen*
Spillner, A.; Breymann, U. – dpunkt.verlag, Heidelberg
- *Lehrbuch der Software-Technik*
Balzert, H. – Spektrum Akademischer Verlag
- *Methoden und Verfahren des QM für Software*
Deutsche Gesellschaft; DGQ für Qualität e.V.
- *Methodisches Testen von Programmen*
Myers, G.J. – Oldenbourg
- *Reviews in der System- und Softwareentwicklung: Grundlagen, Praxis, kontinuierliche Verbesserung, 1. Auflage*
Rössler, P.; Schlich, M.; Kneuper, R. – dpunkt.verlag, Heidelberg
- *Software automatisch testen*
Dustin, E. und Rashka, J. und Paul, J.; – Springer Verlag
- *Software-Metriken in der Praxis*
Ebert, Ch.; Dumke, R. – Springer Verlag
- *Software-Prüfung*
Frühauf, K.; Ludewig, J.; Sandmayr, H. – vdf
- *Software Qualität*
Liggesmeyer, P. – Spektrum-Verlag, Heidelberg, Berlin
- *Software QM – Aufgaben, Möglichkeiten, für Qualität e.V.*
Deutsche Gesellschaft; Lösungen; DGQ

-
- *Software-Qualitätsmanagement in der Praxis*
Wallmüller, E. – Hanser
 - *Software Qualitätssicherung in der Praxis*
Wallmüller, E. – Hanser
 - *Testen in Scrum-Projekten Leitfaden für Softwarequalität in der agilen Welt. Aus- und Weiterbildung zum ISTQB® Certified Agile Tester – Foundation Extension, 2. Version*
Linz, T. – dpunkt.verlag, Heidelberg
 - *Agile Testing*
Crispin, L.; Gregory, J. – Pearson Education Boston
 - *Agile Testing Foundations*
Black, R. – BCS Learning & Development Ltd. Swindon UK
 - *A Practitioner's Guide to Software Test Design*
Copeland, L. – Artech House Publishers
 - *Client-Server Software Testing*
Mosley, D.J. – Prentice Hall PTR
 - *Complete Guide to Software Testing (2e)*
Hetzel, W. – QED Information Sciences
 - *Effective Methods for Software Testing*
Perry, W.E. – Wiley
 - *Experiences of Test Automation*
Graham, D.; Fewster, M. – Pearson Education
 - *Foundations of Software Testing: ISTQB Certification (4e)*
Black, R.; van Veenendaal, E.; Graham, D. – Cengage Learning
 - *How Perspective-Based Reading can Improve Requirement Inspections*
Shull, F.; Rus, I.; Basili, V. – IEEE Computer, Volume 33, Issue 7, pp 73-79
 - *Integrated Test Design and Automation*
Buwalda, H. et al. – Addison Wesley
 - *Introducing Software Testing*
Tamres, L. – Addison Wesley
 - *Lessons Learned in Software Testing*
Kaner, C.; Bach, J.; Pettichord, B. – Wiley
 - *Managing the Testing Process*
Black, R. – Microsoft Press
 - *More Agile Testing*
Gregory, J.; Crispin, L. – Pearson Education
 - *Model-Based Testing Essentials: Guide to the ISTQB Certified Model-Based Tester: Foundation Level*
Kramer, A.; Legeard, B. – John Wiley & Sons

- *Peer Reviews in Software*
Wiegers, K. – Pearson Education
- *Perfect Software and Other Illusions about Testing*
Weinberg, G. – Dorset House
- *Software Inspection*
Gilb, T.; Graham, D. – Addison Wesley
- *Software-Qualität*
Liggesmeyer, P. – Spektrum-Verlag
- *Software Test Automation*
Fewster, M.; Graham, D. – Addison-Wesley
- *Software Testing and Continuous Quality Improvement*
Lewis, W.E. – Auerbach
- *Software Testing, A Craftsman's Approach*
Jorgensen, P. – CRC Press
- *Software Testing in the real world*
Kit, E. – Addison-Wesley
- *Software Testing Techniques (2e)*
Beizer, R. – Van Nostrand Reinhold; Boston MA
- *Software-Test/Verifikation und Validation*
Thaller, G.E. – Heise
- *Systematic Software Testing*
Craig, R.D., Jaskiel, St. – Artech House Publishers
- *Testing Computer Software*
Kaner, C.; Falk, J.; Nguyen, H. – Wiley
- *The Art of Software Testing*
Myers, G. – John Wiley & Sons
- *The Domain Testing Workbook*
Kaner, C.; Padmanabhan, S.; Hoffman, D. – Context-Driven Press
- *The Effectiveness of Software Development Technical Reviews: A Behaviorally Motivated Program of Research*
Sauer, C. – IEEE Transactions on Software Engineering, Volume 26, Issue 1, pp 1-14
- *The Testing Practitioner (Chapters 8 - 10)*
van Veenendaal, E. – UTN Publishers
- *Waltzing with Bears*
De Marco, T. und Lister, T. – Dorset House Publishing
- ISO 9001
Thaller, G.E. – Heise

■ Web

- www.istqb.org
International Software Testing Qualification Board
- www.brightest.org
International certification body
- www.asqf.de
Arbeitskreis Software Qualität und Fortbildung
- www.austriantestingboard.at
Austrian Testing Board
- www.german-testing-board.info
German Testing Board
- www.trendig.com
trendig technology services GmbH
- <http://www.istqb.org/downloads.html>
Glossary of Terms (Englisch)
- <https://www.german-testing-board.info/lehrplaene/istqbr-certified-tester-schema/kurz-vorgestellt/>
Download-Links zur deutschen und englischen Version des Glossars, Download-Links zur deutschen und englischen Version der Lehrpläne Certified Tester