GTB
German Testing Board
Software. Testing. Excellence.

Kapitel 2

Basiswissen Softwaretest Certified Tester

Testen im Softwareentwicklungslebenszyklus

HS@GTB
2019
Version 3.1

Nach dieser Vorlesung sollten Sie ...

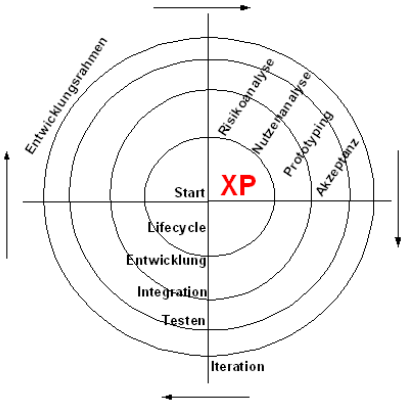
- wissen, dass Softwareentwicklungsmodelle an Projekt- und Produkteigenschaften angepasst werden müssen
- wissen, wann das Testen im Softwareentwicklungslebenszyklus beginnt
- wissen, wie Entwicklungs- und Testaktivitäten zusammenhängen
- die Teststufen Komponententest, Integrationstest, Systemtest und Abnahmetest vergleichen können
- Eigenschaften "guter" Tests nennen können, die in beliebigen Entwicklungszyklen anwendbar sind
- wissen, wie das Testen von neuen Produktversionen aussehen soll
- typische Anlässe für Wartungstests kennen
- die Rolle von Regressionstests und Auswirkungsanalysen in der Softwarewartung beschreiben können
- die Testarten funktionaler Test, nicht-funktionaler Test, White-Box-Test und änderungsbasierter Test vergleichen können

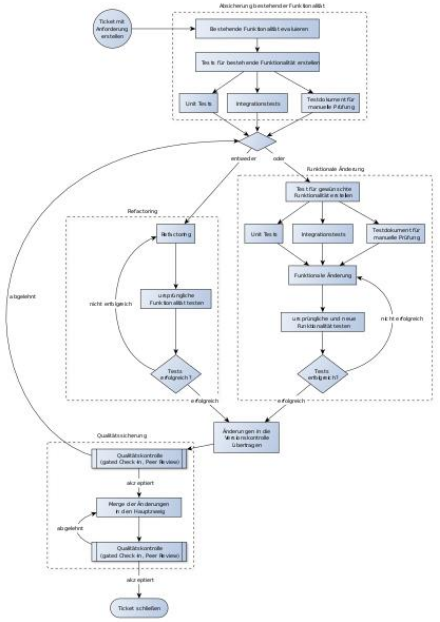
Testen im Softwareentwicklungslebenszyklus

- **Softwareentwicklungslebenszyklus-Modelle**
- Teststufen
- Testarten
- Wartungstest

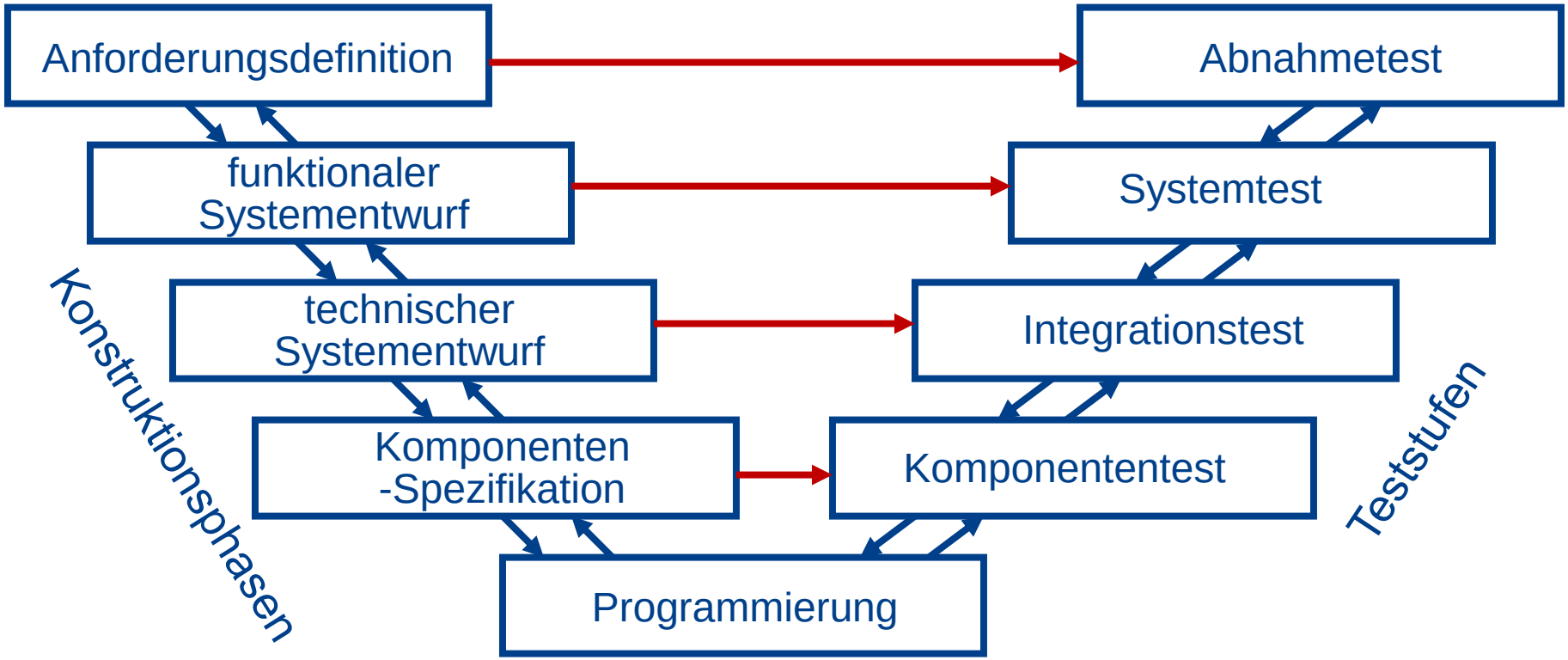
Softwareentwicklung und Softwaretesten

- Softwareentwicklungslebenszyklus-Modell
  - beschreibt Aktivitäten, für jede Phase eines Softwareentwicklungsprojekts
  - beschreibt die logische und zeitliche Beziehung der Aktivitäten
- Es gibt verschiedene Softwareentwicklungslebenszyklus-Modelle
  - jedes erfordert andere Ansätze für das Testen
- Hier zwei Kategorien vorgestellt:
  - Sequenzielle Entwicklungsmodelle
  - Iterative und inkrementelle Entwicklungsmodelle



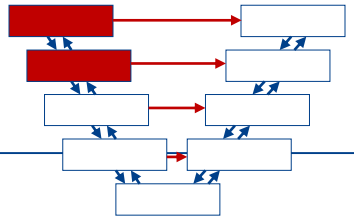


Allgemeines V-Modell (sequentielles Entwicklungsmodell)



Legende
Testfälle basieren auf den entsprechenden Dokumenten

Allgemeines V-Modell
Konstruktionsphasen (1 von 2)



Konstruktionsphasen

- konstruktive Aktivitäten im linken Ast
- Softwaresystem ist zunehmend detaillierter zu beschreiben

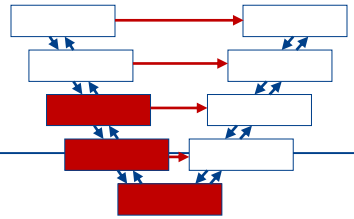
Anforderungsdefinition

- Anforderungen des Auftraggebers oder Systemanwenders sammeln
- damit Zweck und Leistungsmerkmale des zu erstellenden Softwaresystems definiert

Funktionaler Systementwurf

- Anforderungen abbilden auf Funktionen und Dialogabläufe des Systems

Allgemeines V-Modell
Konstruktionsphasen (2 von 2)



Technischer Systementwurf

- Entwurf der technisches Realisierung des Systems
- Definition der Schnittstellen zur Systemumwelt
- Zerlegung des Systems in Teilsysteme (Systemarchitektur)
- Teilsysteme möglichst unabhängig voneinander entwickeln

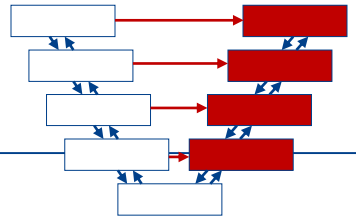
Komponentenspezifikation

- pro Teilsystem Aufgabe, Verhalten, innerer Aufbau und Schnittstellen definieren

Programmierung

- Implementierung jedes spezifizierten Bausteins (Modul, Unit, Klasse o.Ä.)

Allgemeines V-Modell
Teststufen (1 von 2)



Motivation

- Fehler am einfachsten auf derselben Abstraktionsstufe gefunden
- rechter Ast ordnet jedem Konstruktionsschritt eine Teststufe zu

Komponententest

- Erfüllt jeder Software- oder Hardwarebaustein seine Spezifikation?*

Integrationstest

- Spielen Komponenten wie im technischen Systementwurf zusammen?

Systemtest

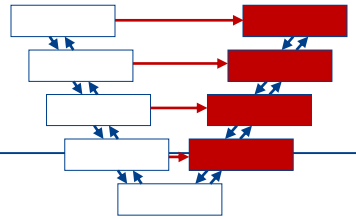
- Erfüllt das System als Ganzes die spezifizierten Anforderungen?

Abnahmetest

- Weist das System aus Kundensicht vereinbarte Leistungsmerkmale auf?

* Dieser Kurs fokussiert auf den Test von Softwarekomponenten

Allgemeines V-Modell
Teststufen (2 von 2)



Teststufe: je eine Instanz des Testprozesses

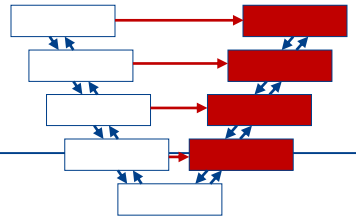
Unterschiede der Teststufen:

- Ziele und Arbeitsergebnisse
- Testbasis und Testobjekte
- Typische Fehlerzustände und Fehlerwirkungen
- Teststrategie und Testverfahren
- Testumgebung und Testwerkzeuge
- Verantwortlichkeiten und spezialisiertes Testpersonal

Beispiele:

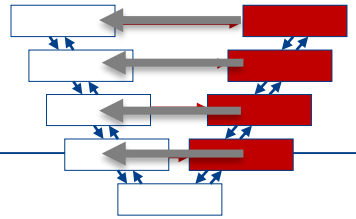
- Testumgebung:
  im Abnahmetest produktionsähnlich, im Komponententest die Entwicklungsumgebung.
- Testverfahren:
  Im Systemtest meist Black-Box-Testverfahren, im Komponententest auch White-Box-Testverfahren
- Sind Konfigurationsdaten Teil des Systems?
  Dann auch Test dieser Daten im Systemtest berücksichtigen

Allgemeines V-Modell
Anmerkungen



- Teststufen: Komponententest, Integrationstest, Systemtest, Abnahmetest an V-Modell erläutert
- auch in anderen Entwicklungsmodellen vorhanden (evtl. andere Namen)
- Allg.: Teststufen beginnen mit dem Test von kleinen Einheiten, integrieren schrittweise das System bis zu Abnahme durch den Kunden
- Weitere im Lehrplan definierte Teststufen
  - **Komponentenintegrationstest**
    nach Komponententest, testet Zusammenspiel der Softwarekomponenten
  - **Systemintegrationstest**
    nach Systemtest, testet das Zusammenspiel verschiedener Systeme oder zwischen Hardware und Software

Allgemeines V-Modell
Validierung



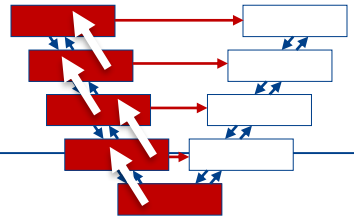
Validierung pro Teststufe: erfüllen die Entwicklungsergebnisse die Anforderungen auf derselben Stufe?

Fragen für Validierung:

- Löst ein (Teil-)Produkt eine festgelegte Aufgabe?
- Ist es für seinen Einsatzzweck tauglich?
- Produkt im Kontext der beabsichtigten Produktnutzung sinnvoll?

Validierung [ISO 9000] (Glossar V.3.2):
Bestätigung durch Bereitstellung eines objektiven Nachweises, dass die Anforderungen für einen spezifischen beabsichtigten Gebrauch oder eine spezifische beabsichtigte Anwendung erfüllt worden sind.

Allgemeines V-Modell
Verifizierung



Verifizierung:

- auf eine Entwicklungsphase bezogen
- Nachweis der Korrektheit und Vollständigkeit bzgl. direkter Spezifikation

Ziel der Verifizierung:

- Spezifikationen korrekt umgesetzt?
- unabhängig von einem beabsichtigten Zweck oder Nutzen des Produkts!

Verifizierung [ISO 9000] (Glossar V.3.2):
Bestätigung durch Bereitstellung eines objektiven Nachweises, dass festgelegte Anforderungen erfüllt worden sind.

Anmerkung:
Test hat beide Aspekte, Validierungsanteil nimmt mit steigender Teststufe zu

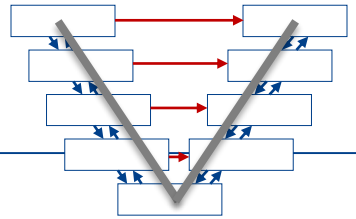
Exkurs
Unterschied zwischen Validierung und Verifizierung?

- Produkt: Rettungsring aus Blei
- Anforderung: Material soll Blei sein

- Ergebnis der Verifizierung?
  (Anforderung erfüllt?)
- Ergebnis der Validierung?
  (für beabsichtige Produktnutzung sinnvoll?)

(Quelle: Wikipedia, Uwe H. Friese)

Allgemeines V-Modell

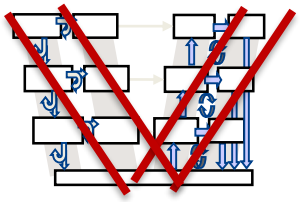


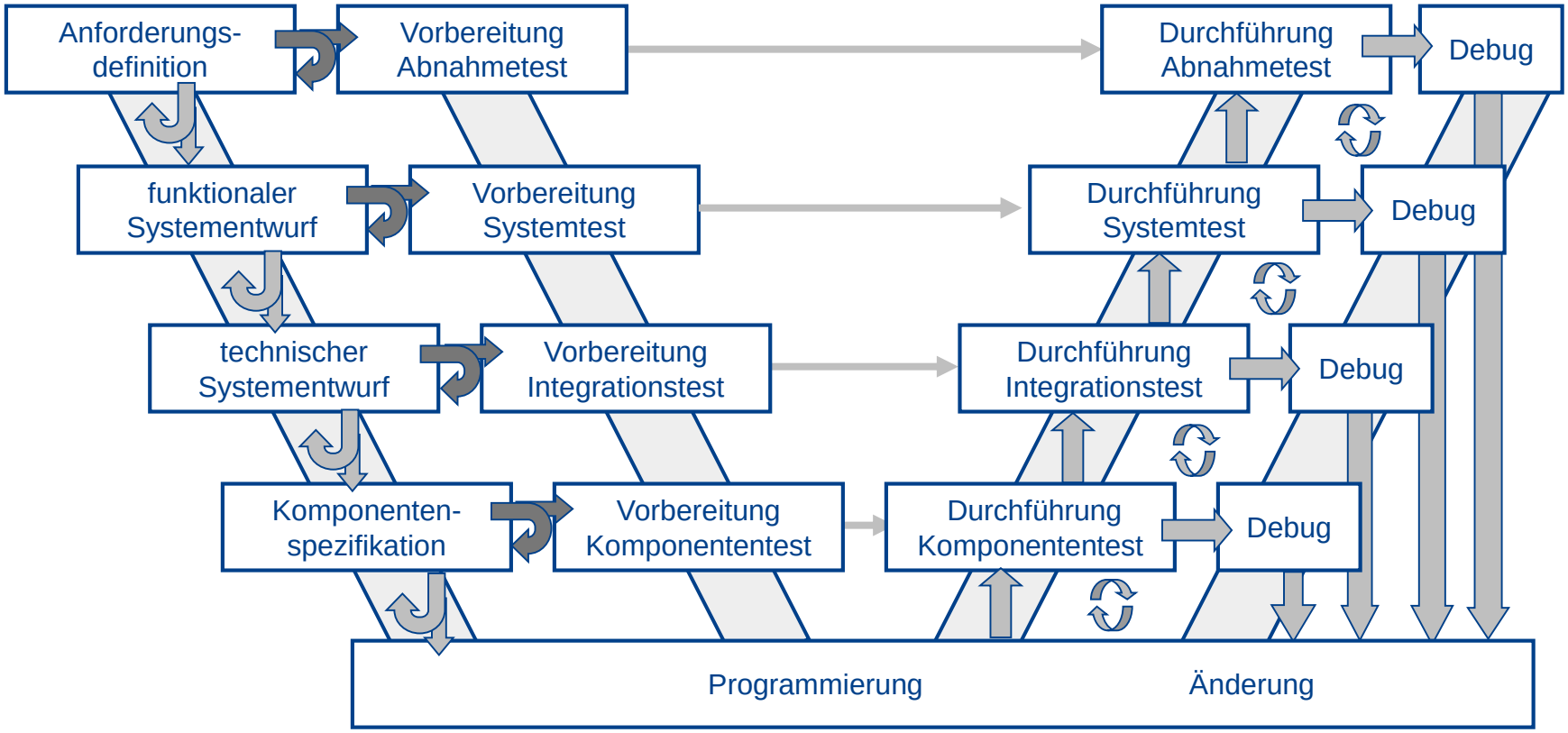
- Konstruktions- und Testaktivitäten sind getrennt, aber gleichwertig
- »V« veranschaulicht **V**erifizierung und **V**alidierung
- Unterscheidung von Teststufen, bezogen auf jeweilige Entwicklungsstufe

Falscher Eindruck: Testen beginnt erst relativ spät? Dies ist falsch.

- Teststufen: Testdurchführung und –auswertung
- Testvorbereitung (Testplanung, Testspezifikation) parallel zu den Entwicklungsschritten im linken Ast

Exkurs: W-Modell
Weiterentwicklung des V-Modells





Spillner/Roßner/Winter/Linz
Praxiswissen Softwaretest -Testmanagement,
dpunkt, 2014, Kap. 3.3.2

Review, Previews, Dokumente

Testfälle, Testrahmen

test, debug, ändern, re-test

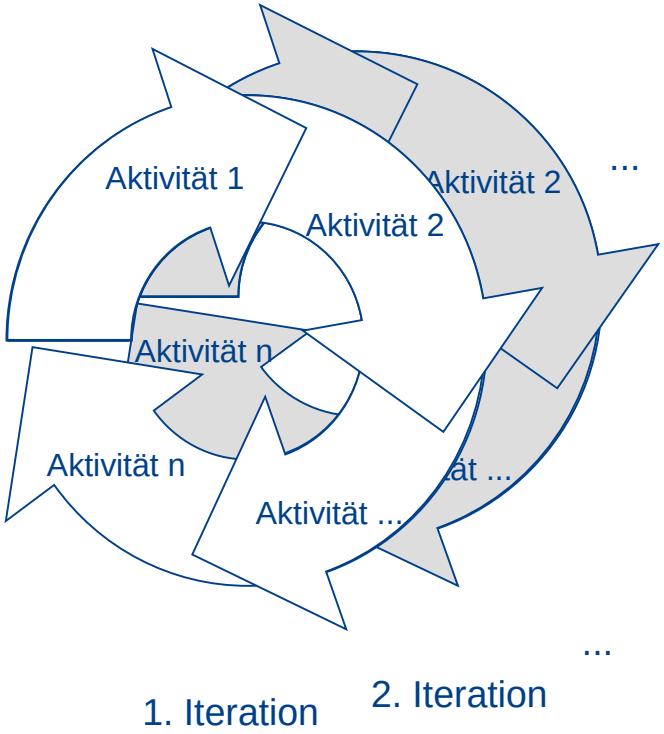
Iterativ-inkrementelle Entwicklungsmodelle (1 von 2)

Funktionen in Zyklen spezifiziert, entworfen, implementiert und getestet

System nicht »am Stück« erstellt, sondern in Inkrementen (Versionsstände, Zwischenlieferungen)

Pro Iteration:

- neue Features bzw. Funktionen
- Änderungen vorhandener Funktionen
- Verbesserung der Qualität des Systems



Iterativ-inkrementelle Entwicklungsmodelle (2 von 2)

Iterative und inkrementelle Entwicklung: unterschiedlich, kombiniert

rein inkrementell („adding“)

- Ziele des Gesamtsystems von Anfang an klar
- System iterativ überarbeitet, detailliert und ergänzt
- Inkrement: keine Untergrenze bzgl. sichtbaren Änderungsumfangs
- Annahme: Teilergebnisse nicht mehr ändern

rein iterativ („reworking“)

- System (inkl. Ziele) entsteht nach und nach mit den Inkrementen
- Features in Zyklen (mit festgelegter Dauer) spezifiziert, entworfen, implementiert und getestet
- pro Iteration lauffähige, potenziell auslieferbare Software
- Annahme: Leistungsmerkmale oder Projektumfang jederzeit änderbar

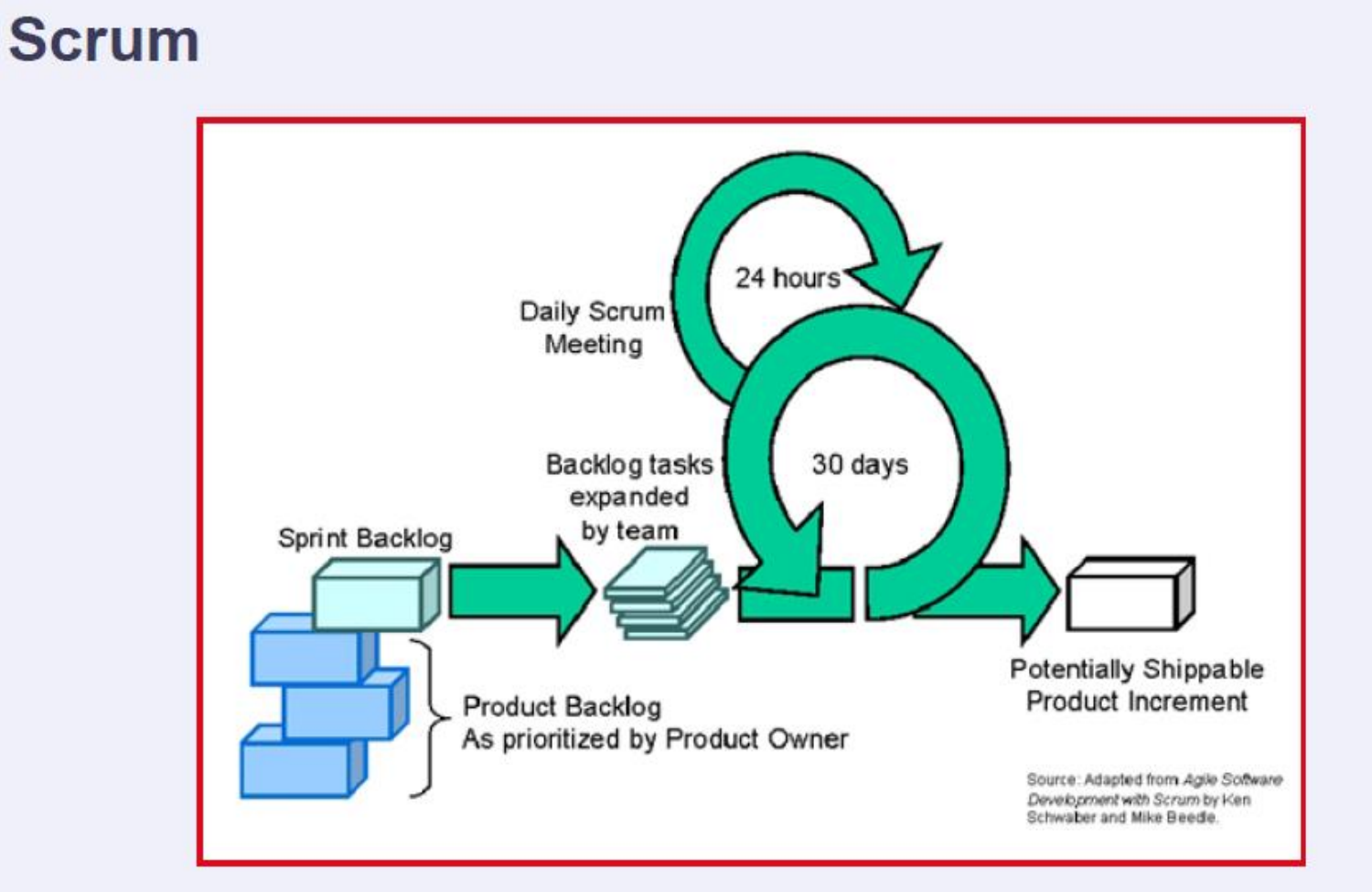
Beispiele iterativ-inkrementeller Entwicklungsmodelle

- **Rational Unified Process**
  - Eher relativ lange Iterationen (z.B. zwei bis drei Monate)
  - Inkremente der Features sind entsprechend groß
- **Scrum**
  - Eher kurze Iterationen (z.B. Stunden, Tage oder einige Wochen)
  - Inkremente der Features sind entsprechend klein (z.B. einige Verbesserungen und/oder zwei oder drei neue Features)
- **Kanban**
  - Iterationen mit oder ohne festgelegte Länge
  - ein einziges Feature bis zum Abschluss liefern oder
  - Gruppen von Features in einem Release zusammenfassen
- **Spiralmodell (oder Prototyping)**
  - Erstellt „experimentelle“ Inkremente
  - einige werden später stark überarbeitet oder sogar weggeworfen

Testen in iterativ-inkrementellen Entwicklungsmodellen

- iterativ-inkrementelle Entwicklungsmodelle:
  - Testphasen und Teststufen oft überlappend
  - Möglichst jedes Feature auf mehreren Teststufen testen
  - Pro Inkrement und Iteration wiederverwendbare Tests nutzen
  - neue Funktionalität: zusätzliche Tests
  - Pro Inkrement in einer Iteration verschiedenen Teststufen durchlaufen
  - Kontinuierlich: Integrationstests und Regressionstests durchführen
  - Verifizierung und Validierung für jedes Inkrement möglich
- Software in kurzen Zeitabständen kontinuierlich …
  - integrieren: continuous integration
  - ausliefern: continuous delivery
  - bereitstellen: continuous deployment
- Wichtig: Testautomatisierung auf mehreren Teststufen

Scrum



Continuous Integration

Continuous Integration Martin Fowler: Continuous Integration.

Idee:

Continuous Integration is a software development practice where members of a team integrate their work frequently, usually each person integrates at least daily - leading to multiple integrations per day. Each integration is verified by an automated build (including test) to detect integration errors as quickly as possible. Many teams find that this approach leads to significantly reduced integration problems and allows a team to develop cohesive software more rapidly. This article is a quick overview of Continuous Integration summarizing the technique and its current usage.

nach http://www.martinfowler.com/articles/continuousIntegration.html

vereinfachte Variante (oft auch Vorstufe): Nightly Build

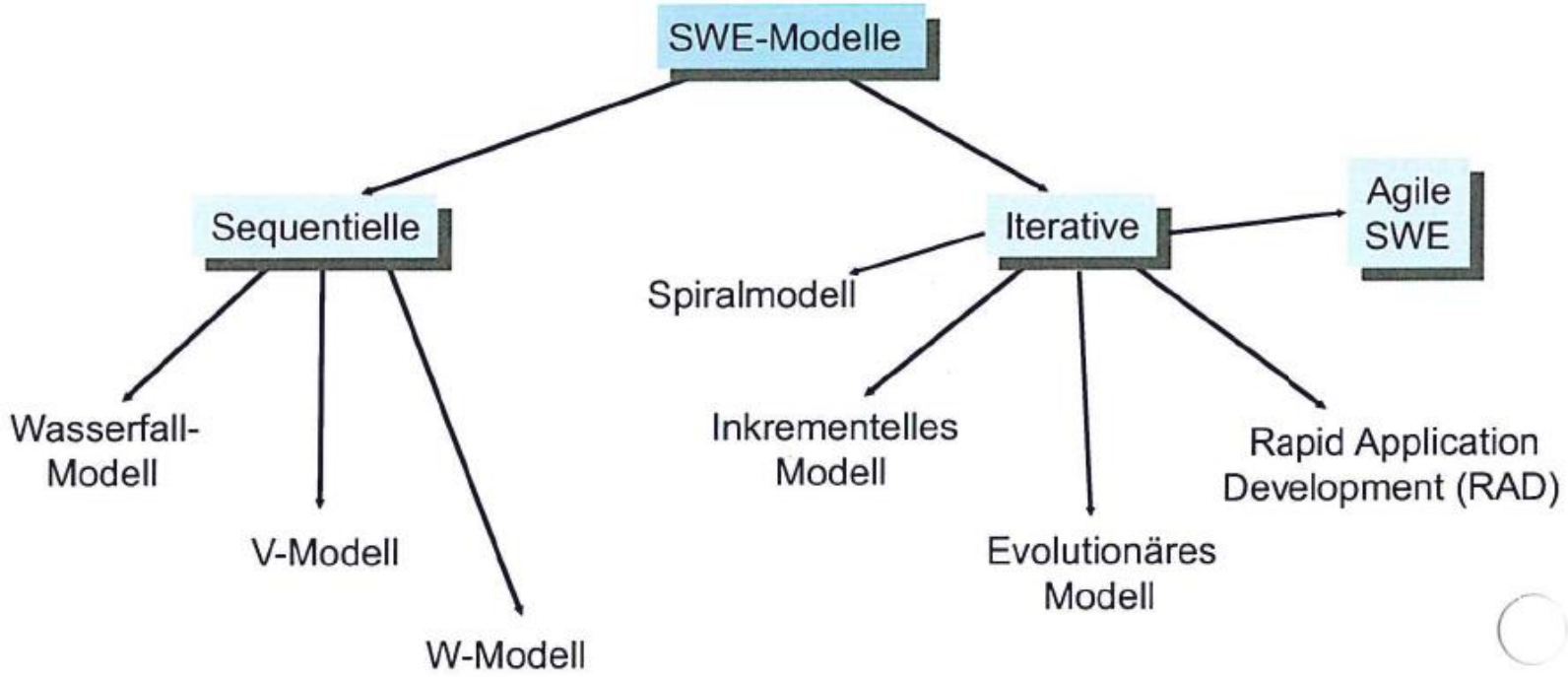
Testen in agilen Entwicklungsmodellen

- agile Entwicklung: siehe iterativ-inkrementelle Entwicklung
- agile Entwicklung oft in selbstorganisierenden Teams
  - Einfluss auf Organisation von Tests
  - Einfluss auf Beziehung zwischen Testern und Entwicklern
- wachsendes System
  - Freigabe pro Feature, Iteration oder Hauptrelease
  - Freigabe-unabhängig: Regressionstests mit der Zeit immer wichtiger
- oft nach Wochen oder Tagen bereits nutzbare Software
  - Alle Anforderungen aber auch erst nach Monaten oder Jahren erfüllt
- Agiler Softwaretest: siehe ISTQB-Lehrplan Foundation Level Agile Tester.

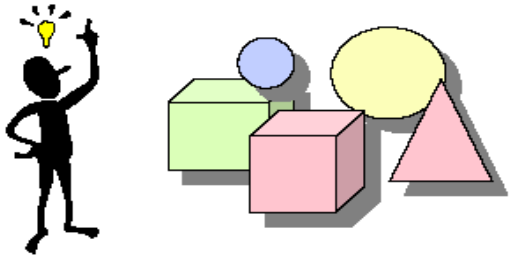
Tipps für gutes Testen

- pro Entwicklungsaktivität eine entsprechende Testaktivität
- Testaktivitäten so früh im Entwicklungszyklus wie möglich
  - Testanalyse und -entwurf parallel zur Entwicklungsstufe beginnen
- Tester früh einbinden:
  - Definition der Anforderungen
  - Softwareentwurf
  - Review-Prozess von Anforderungen, Entwurfsdokumenten, User Stories, etc.
- Softwareentwicklungsmodelle
  - nicht „Out of the Box“ anwendbar
  - an Projekt- und Produktcharakteristika anpassen (Anzahl der Teststufen, Anzahl und Länge der Iterationen, etc.)

Überblick: Softwareentwicklungsmodelle



Überblick: Testen im Softwareentwicklungslebenszyklus



- Abnahmetest
  - Systemanforderungen
  - Einsatztauglichkeit (Kundensicht)
- Systemtest
  - Funktionalität
  - Nichtfunktionale Anforderungen (Performanz, ...)
- Integrationstest
  - Testen des Zusammenspiels integrierter Komponenten/Module
  - Schnittstellen
- Komponenten-/Modultest
  - Komponenten-/Modulspezifikation
  - Implementierung

Softwareentwicklungslebenszyklus-Modelle im Kontext

- Faktoren, die die Auswahl des Softwareentwicklungslebenszyklus-Modells und das Testen beeinflussen können:
  - Projektziele
  - Art des zu entwickelnden Produkts
  - Geschäftsprioritäten (z.B. Time-to-Market)
  - Produkt- und Projektrisiken
  - Kulturelle Aspekte

Softwareentwicklungslebenszyklus-Modelle im Kontext – Beispiele (1 von 2)

- Entwicklung und Test für internes Verwaltungssystems anders als Entwicklung und Test von sicherheitskritischem System (z.B. Bremssteuerungssystem für Autos)
- Für Integration von Standardsoftware (commercial off-the-shelf, COTS) in größeres System: (nicht-)funktionale Interoperabilitätstests in Systemintegrationsteststufe oder Abnahmeteststufe durchführen
- Organisatorische und kulturelle Probleme behindern iterative Entwicklung, wenn Kommunikation zwischen Teammitgliedern erschwert wird

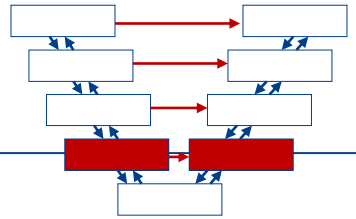
Softwareentwicklungslebenszyklus-Modelle im Kontext – Beispiele (2 von 2)

- Modelle sind kombinierbar!
  - Das V-Modell für Entwicklung und Integration eines Backend-Systems
  - agiles Entwicklungsmodell zur Entwicklung der Benutzerschnittstelle (UI)
  - Prototyping in früher Projektphase
  - Nach experimenteller Phase: inkrementelles Entwicklungsmodell
- Systeme im „Internet der Dinge“ (Internet of Things, IoT)
  - vielen verschiedene Objekte wie Geräte, Produkte und Dienste
  - eigenständige Softwareentwicklungs-lebenszyklus-Modelle pro Objekt
  - Softwareentwicklungslebenszyklus betont die späten Phasen nach Übergang in betriebliche Nutzung (z.B. Betrieb, Aktualisierung, Außerbetriebnahme)

Testen im Software-entwicklungs-lebenszyklus

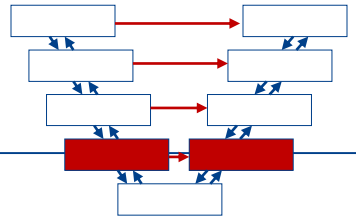
- Softwareentwicklungslebenszyklus-Modelle
- **Teststufen → Komponententest**
- Testarten
- Wartungstest

Komponententest
Begriffsklärung



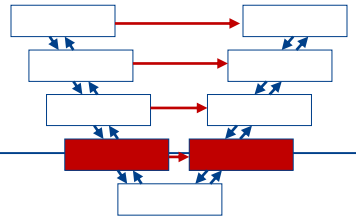
- Komponententest (erste Teststufe): Softwarebausteine erstmalig getestet
- inkrementelle/iterative (z.B. agile) Entwicklungsmodelle:
  - Wenn stetig Codeänderungen, dann automatisierte Regressionstests wichtig
  - Diese schaffen Vertrauen, dass Änderungen Bestehendes nicht beschädigen
- Name für kleinste Softwareeinheiten abhängig von Programmiersprache
  - Module, Units oder Klassen (objektorientierte Programmierung)
  - entsprechende Tests: Modul-, Unit- bzw. Klassentest
- Von Programmiersprache abstrahiert: Komponente oder Softwarebaustein
  - Test eines Softwarebausteins: Komponententest

Komponententest
Testziele (1 von 4)



- **Aufgabe von Komponententests**:
  - Realisiert Testobjekt die geforderte Funktionalität korrekt und vollständig?
  - Funktionalität gleichbedeutend mit Ein-/Ausgabe-Verhalten von Testobjekt
  -
- **Weitere Testziele:**
  - Risikoreduktion
  - Verifizierung der (nicht-)funktionalen Verhaltensweisen der Komponente
  - Schaffen von Vertrauen in die Qualität der Komponente
  - Finden von Fehlerzuständen in der Komponente
  - Fehlerzustände nicht an höhere Teststufen weitergeben
- Prüfung von Korrektheit und Vollständigkeit der Implementierung: Komponente wird getestet, jeder deckt Testfall bestimmte Ein-/Ausgabe-Kombination (Teilfunktionalität) ab

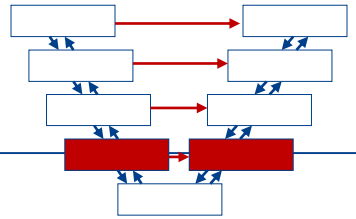
Komponententest
Testziele (2 von 4)



Test auf Robustheit

- Jede Softwarekomponente interagiert später mit Nachbarkomponenten
- Komponenten können falsch angesprochen oder verwendet werden
- Dann nicht gleich Dienst einstellen und das System zum Absturz bringen
- Stattdessen: Fehlersituation abfangen und »vernünftig« / robust reagieren

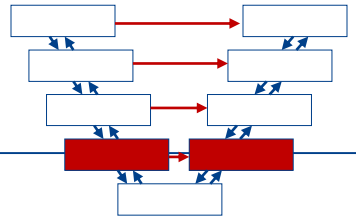
Komponententest
Testziele (3 von 4)



- **alle Komponenteneigenschaften überprüfen**
  - die die Qualität der Komponente beeinflussen
  - die in höheren Teststufen nicht mehr (einfach) geprüft werden können
  - Beispiele: Effizienz, Wartbarkeit

- **Effizienz**
  - Wie wirtschaftlich geht die Komponente mit verfügbaren Ressourcen um?
  - Teilkriterien (z.B. Speicherverbrauch, Antwortzeit) im Test exakt messen

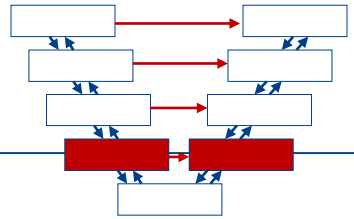
Komponententest
Testziele (4 von 4)



- **Wartbarkeit**
  - Wie leicht oder schwer ist es, das Programm zu ändern / weiterzuentwickeln?
  - Wie viel Aufwand kostet es, Programm und Kontext zu verstehen?
  - Prüfaspekte: Codestruktur, Modularität, Kommentierung des Codes, Verständlichkeit und Aktualität der Dokumentation usw.

- **Wartbarkeit nicht durch dynamische Tests überprüfbar**
  - Stattdessen: Analysen von Programmtext und Spezifikation, statischer Test und (Code-) Review
  - Analysen wenn möglich im Rahmen des Komponententests durchführen.

Komponententest
Testbasis



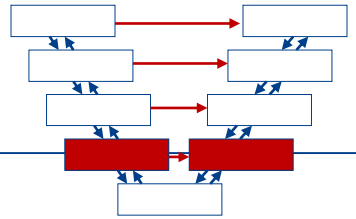
Testbasis:

- Arbeitsergebnisse des Feinentwurfs, Programmcode
- Alle Dokumente zur zu testenden Komponente
  - Datenmodelle, Klassenmodelle, Verhaltensmodelle (z.B. Sequenzdiagramme)
  - Vor- und Nachbedingungen der Operationen, Invarianten der Komponente

agile Projekte oft testgetrieben abgearbeitet (test-first, test-driven, TDD)

- automatisierte Testfälle sind Spezifikation und ausführbarer Test zugleich

Komponententest
Testobjekte

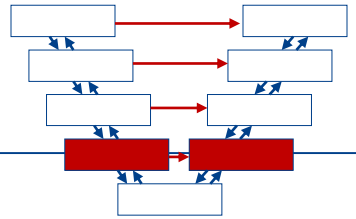


- **Testobjekte: Softwarebausteinen des Systems**
  - Komponenten, Units oder Module
  - Code, Datenstrukturen, Klassen
  - Datenbankmodule

- **Jeder Softwarebaustein isoliert von anderen getestet**
  - Keine komponentenexternen Einflüsse
  - Fehlerwirkung aufgedeckt? Ursache in der getesteten Komponente!

- **zu testende Komponente aus mehreren Bausteinen zusammengesetzt?**
  - Wichtig: komponenteninterne Aspekte prüfen
  - nicht die Wechselwirkung mit Nachbarkomponenten prüfen!

Komponententest
Fehlerzustände und Fehlerwirkungen

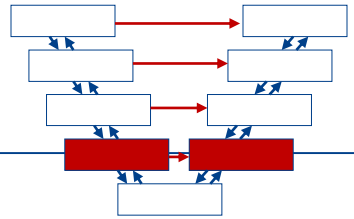


- **Typische Fehlerzustände/-wirkungen:**
  - fehlerhafter Code,
  - fehlerhafte Logik
  - Berechnungsfehler
  - Datenflussprobleme
  - fehlende und falsch gewählte Programmpfade (z.B. vergessene Sonderfälle)

- **Robustheitstests**: stürzt Komponente bei fehlerhafter Benutzung ab?

- **Effizienztests:** erfüllt Komponente unter spezifizierter Last / Überlast ihre Effizienzanforderungen?

Komponententest
Testumgebung (1 von 2)

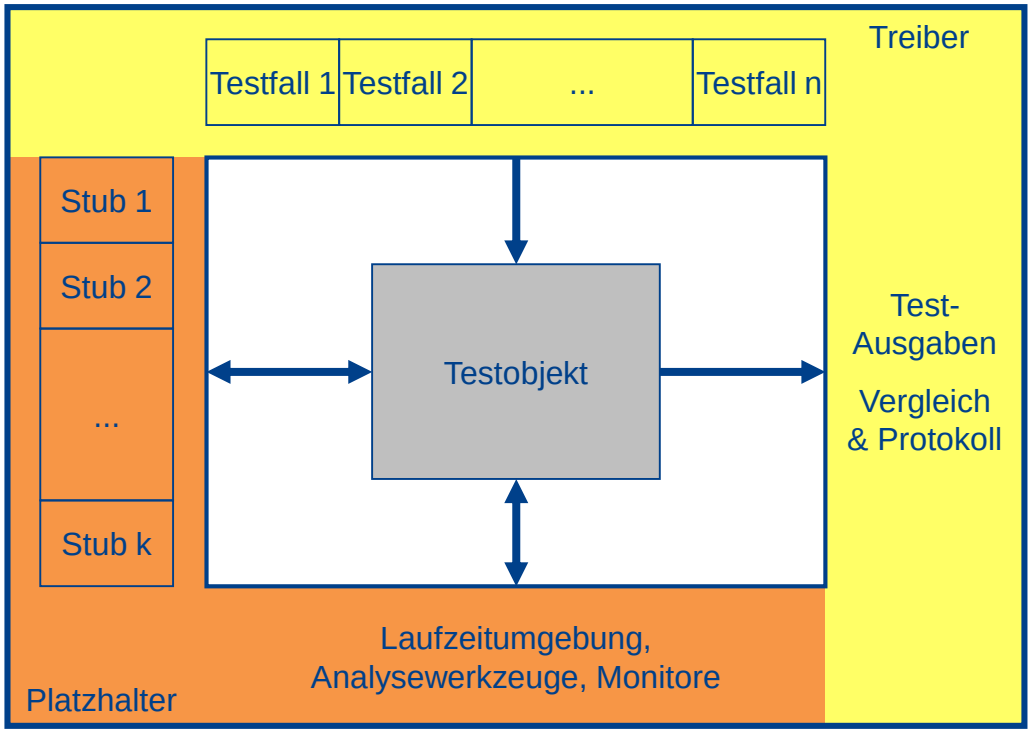


Testumgebung besteht aus

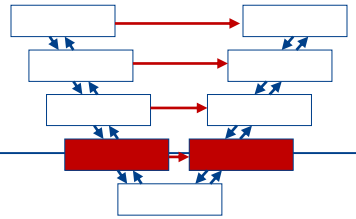
- **Treiber/Testtreiber (Driver)**
  Aufruf der Dienste des Testobjekts

  und/oder

- **Platzhalter (Stub, Dummy)**
  Simulation der Dienste,
  die das Testobjekt importiert

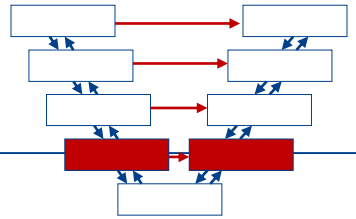


Komponententest
Testumgebung (2 von 2)



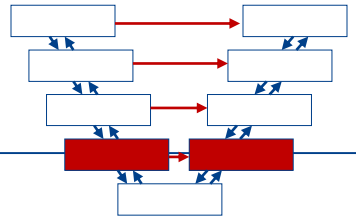
- Diese Teststufe: entwicklungsnahes Arbeiten
- Für Testumgebung Entwickler-Know-how notwendig
- Code des Testobjekts / der Schnittstelle muss verfügbar sein
  - Nur so kann Aufruf des Testobjekts programmiert werden
- Komponententests oft von Entwicklern durchgeführt: »Entwicklertest«
- Entwickler
  - Entwickeln Code für Komponente
  - schreiben Tests
  - führen Tests aus
- agile Entwicklung: Schreiben von automatisierten Komponententestfällen auch VOR dem Schreiben von Anwendungscode möglich

Komponententest
Teststrategie (1 von 2)



- Zugriff auf den Programmcode: Komponententest = White-box Test
- Entwurf Testfälle unter Ausnutzung des Wissens über Interna
- Vorliegen des Programmcodes bei Testdurchführung
  - Programmvariablen während Durchführung beobachten
  - Somit korrektes oder fehlerhaftes Verhalten der Komponente identifizierbar

Komponententest
Teststrategie (2 von 2)



Praxis: Komponententest als Kombination aus Black-box- und White-box-Test

- Testfälle aus Anforderungen der Komponenten ableiten
- Überdeckung der Strukturelemente als Endekriterium nutzen
- reale Softwaresysteme haben oft tausenden Komponenten
  - Einstieg in Code nur bei ausgewählten Komponenten praktikabel
- Während Integration Komponenten zu größeren Einheiten zusammengeführt

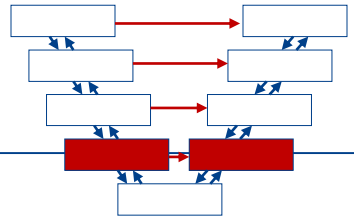
bei Komponententest oft nur zusammengesetzte Komponenten sichtbar

- Diese Testobjekte schon zu groß für Analyse auf Codeebene

Test mit Fokus auf elementare / zusammengesetzte Komponenten?

- In Integrations- und Testplanung festlegen

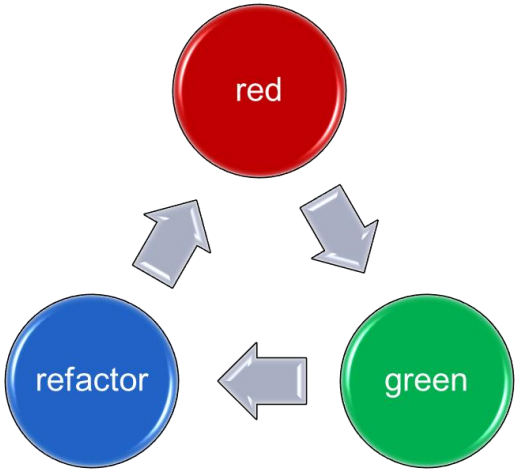
Komponententest
»Test-first«-Ansatz



agile Projekte: testgetriebenes Arbeiten

- erst Testfälle erstellen & automatisieren, dann Komponenten programmieren
- Iterativer Ansatz: Code verbessern, bis Tests keine Fehlerwirkungen zeigen
- Testgetriebene Entwicklung (test driven development, TDD):
  Ein Entwicklungsvorgehen bei dem die Entwicklung der Testfälle und oft auch ihre Automatisierung vor der Entwicklung der Software erfolgen.
  (ISTQB/GTB Glossar V.3.2)

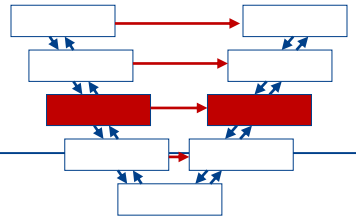
- Anmerkungen:
  - Testfälle: Komponentenspezifikation (Testbasis) und „ausführbare“ Testspezifikation
  - Erst Test „programmieren“ (make it red), dann Code schreiben (make it green), dann Code refactoring (make it blue)



Testen im Software-entwicklungs-lebenszyklus

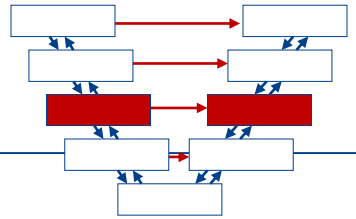
- Softwareentwicklungslebenszyklus-Modelle
- **Teststufen → Integrationstest**
- Testarten
- Wartungstest

Integrationstest
Begriffsklärung



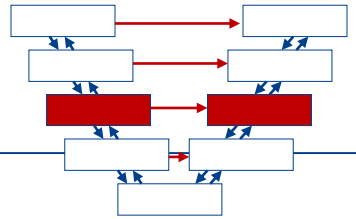
- **Integrationstest: zweite Teststufe nach Komponententest**
- **Voraussetzung**
  - Komponententest hat bereits stattgefunden
  - aufgezeigte Fehlerzustände möglichst korrigiert
- **Integration**
  - Mehrere Komponenten zu größeren Teilsystemen verbinden
  - Verantwortlich: Entwickler, Tester oder spezielle Integrationsteams
- **Integrationstest**
  - Test, ob Zusammenspiel der Einzelteile funktioniert
- **Ziele**
  - Fehlerwirkungen in Schnittstellen finden
  - Fehlerwirkungen im Zusammenspiel verschiedener Komponenten finden

Integrationstest
Testziele



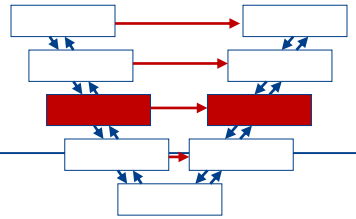
- **Testziele für Integrationstest**
  - Verifizierung des (nicht-)funktionalen Verhaltens der Schnittstellen
  - Fehlerwirkungen in Schnittstellen und Zusammenspiel der Komponenten aufdecken
  - Vertrauen schaffen in Qualität der Schnittstellen
  - Keine Fehlerzustände an höhere Teststufen weitergeben
- schon Versuch der Integration kann scheitern
  - ihre Schnittstellenformate passen nicht ,
  - Dateien fehlen oder
  - Entwickler haben System anders in Komponenten aufgeteilt als spezifiziert
- Fehler in verbundenen Programmteilen schwierig zu entdecken
  - Fehlerzustände im Datenaustausch bzw. in Kommunikation zwischen Komponenten nur durch dynamischen Test aufdeckbar

Integrationstest
Testbasis



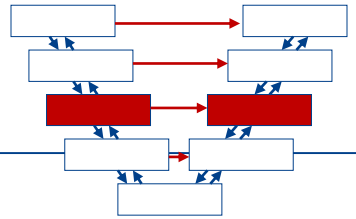
- **Testbasis für Integrationstest**
  - Software- und Systementwurf
  - Sequenzdiagramme
  - Spezifikationen von Schnittstellen und Kommunikationsprotokollen
  - Anwendungsfälle
  - Architektur auf Komponenten- oder Systemebene
  - Workflows
  - Externe Schnittstellendefinitionen (API)

Integrationstest
Testobjekte



- **Einzelbausteine schrittweise zu größeren Einheiten integrieren & testen**
  - Subsysteme
  - Datenbanken
  - Infrastruktur
  - Schnittstellen
  - APIs
  - Microservices
- **Jedes Teilsystem kann Basis für Integration größerer Einheiten sein**
- **Testobjekte für Integrationstest auch mehrfach zusammengesetzte Einheiten**
- Praxis
  - vorhandenes System verändern, ausbauen oder mit anderen Systemen koppeln
  - viele Systemkomponenten sind Standardprodukte (COTS)
    - Nicht für Komponententest betrachten
    - Wichtig für Integrationstest: Zusammenspiel betrachten

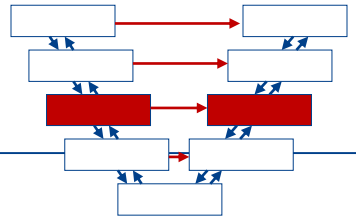
Komponentenintegrationstest vs. Systemintegrationstest



Mehrere Integrationsstufen möglich:

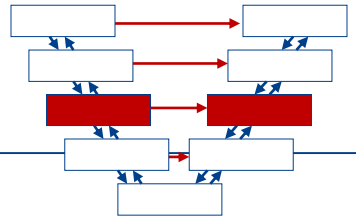
- **Komponentenintegrationstest**
  - nach Komponententest
  - Fokus: Zusammenspiel von Komponenten
  - meist Teil der kontinuierlichen Integration (continuous integration)
  - häufig in Verantwortung der Entwickler
- **Systemintegrationstest**
  - nach Systemtest (auch parallel zu Systemtest möglich)
  - Fokus: Zusammenspiel der Systeme (inkl. Hardware), Pakete, Microservices
  - auch Interaktionen und Schnittstellen von Dritten abdeckbar
    - Herausforderung: keine blockierenden Fehlerzustände im Code von Dritten, Testumgebung
  - häufig in Verantwortung der Tester.

Integrationstest
Fehlerzustände und Fehlerwirkungen (1 von 2)



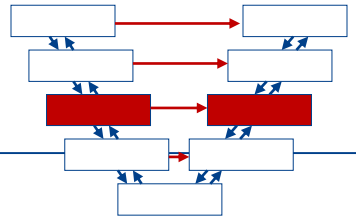
- **typische Fehlerwirkungen bzw. Fehlerzustände:**
  - Komponente übermittelt falsche Daten, empfangende Komponente stürzt ab (funktionaler Fehler einer Komponente, inkompatible Schnittstellenformate, Protokollfehler).
  - Empfangende Komponenten interpretieren Daten falsch (funktionaler Fehler, widersprüchliche oder fehlinterpretierte Spezifikationen).
  - Daten richtig übergeben, aber zum falschen Zeitpunkt (Timing-Problem) oder in zu kurzen Zeitintervallen (Durchsatz- oder Lastproblem).
- **Keine dieser Fehlerwirkungen im Komponententest auffindbar**
- **Fehlerwirkung erst durch Wechselwirkung von Komponenten**

Integrationstest
Fehlerzustände und Fehlerwirkungen (2 von 2)



- **typische Fehlerzustände und -wirkungen für Komponentenintegrationstests**
  - Falsche / fehlende Daten, falsche Datenverschlüsselung
  - Schnittstellenfehlanpassung
  - Fehlerwirkungen in Kommunikation zwischen Komponenten
  - Fehler bzgl. Bedeutung: Einheiten oder Grenzen der kommunizierten Daten
  - Falsche (zeitliche) Abfolge von Schnittstellenaufrufen
- **typischer Fehlerzustände und -wirkungen für Systemintegrationstests**
  - Falsche / fehlende Daten, falsche Datenverschlüsselung
  - Schnittstellenfehlanpassung
  - Fehlerwirkungen in Kommunikation zwischen Systemen
  - Fehler bzgl. Bedeutung: Einheiten oder Grenzen der kommunizierten Daten
  - Fehlende Konformität mit erforderlichen Richtlinien zur Informationssicherheit
  - Inkonsistente Nachrichtenstrukturen zwischen den Systemen

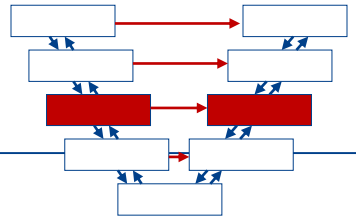
Integrationstest …ohne Komponententest?



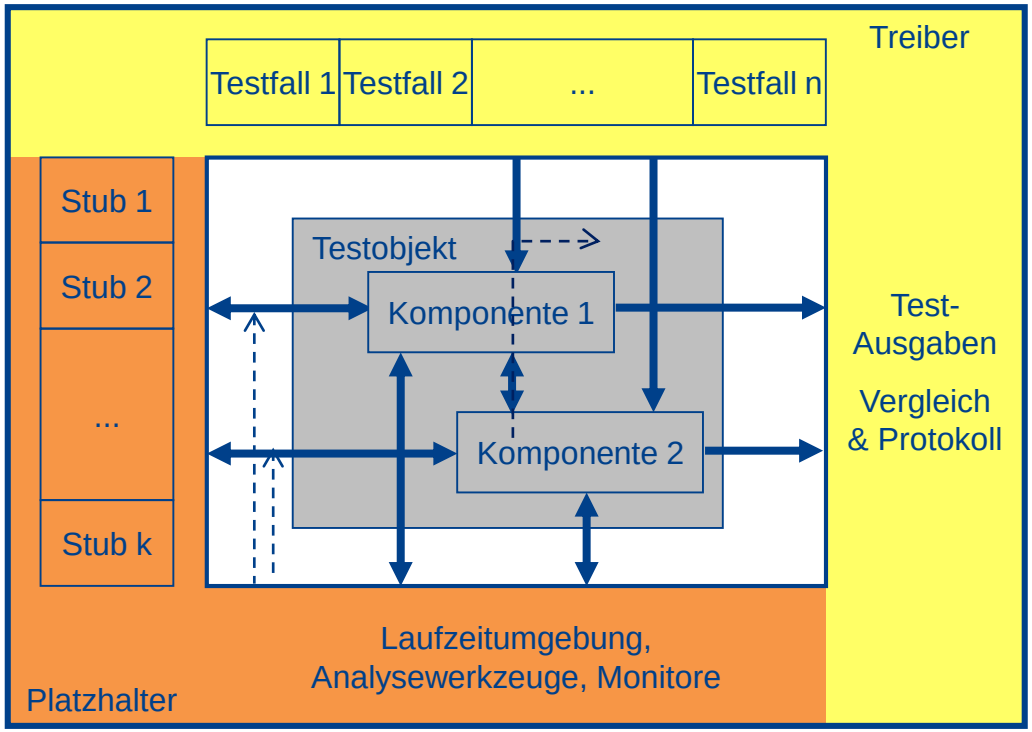
→ Auf Komponententest verzichten? Alles im Integrationstest?

- möglich und leider oft in Praxis anzutreffen
- Folgende Nachteile:
  - Viele auftretende Fehlerwirkungen durch funktionale Fehlerzustände einzelner Komponenten verursacht
  - also impliziter Komponententest in ungeeigneter Testumgebung
  - Einige Fehlerwirkungen nicht provozierbar wegen fehlendem Zugriff
  - Ursachenanalyse zu entdeckten Fehlerwirkungen sehr schwierig

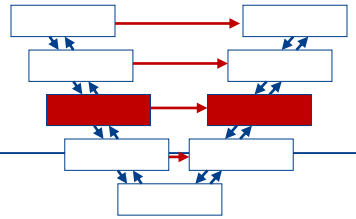
Integrationstest
Testumgebung



- Integrationstest mit Treibern: Testdaten einspielen, Ergebnisse protokollieren
- Wiederverwendung vorhandener Treiber aus Komponententest
- zusätzliches Diagnoseinstrument (Monitore) für Schnittstellenüberwachung

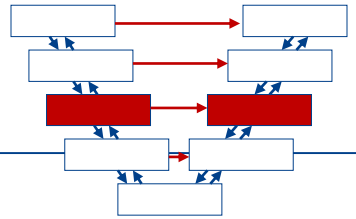


Integrationstest
Integrationsstrategie (1 von 6)



- Reihenfolge für Integration der Komponenten?
  - Ziel: Tests möglichst einfach und schnell durchführbar
- Bereitstellung der Komponenten zu unterschiedlichen Zeiten (… Monate?)
- Projekt kann mit Integration nicht auf Lieferung aller Komponenten warten

Integrationstest
Integrationsstrategie (2 von 6)



Top-down-Integration

- Test beginnt mit Komponente, die andere aufruft, aber selbst nicht aufgerufen wird
- sukzessive Integration der Komponenten niedrigerer Systemschichten
- untergeordnete Komponenten durch Platzhalter ersetzt
- getestete höhere Schicht als Treiber genutzt

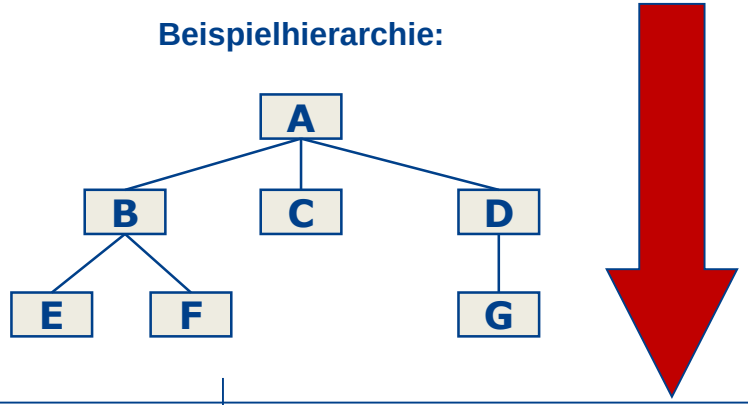
Vorteil

- keine / einfache Treiber benötigt, da aus bereits getestete Komponenten bestehend

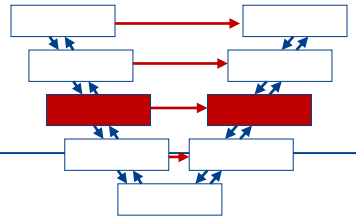
Nachteil

- untergeordnete, noch nicht integrierte Komponenten durch Platzhalter zu ersetzen

Beispielhierarchie:



Integrationstest
Integrationsstrategie (3 von 6)



Bottom-up-Integration

- Test beginnt mit elementaren Komponenten, die keine anderen aufrufen
- Sukzessive Integration von getesteten Komponenten mit anschließendem Test

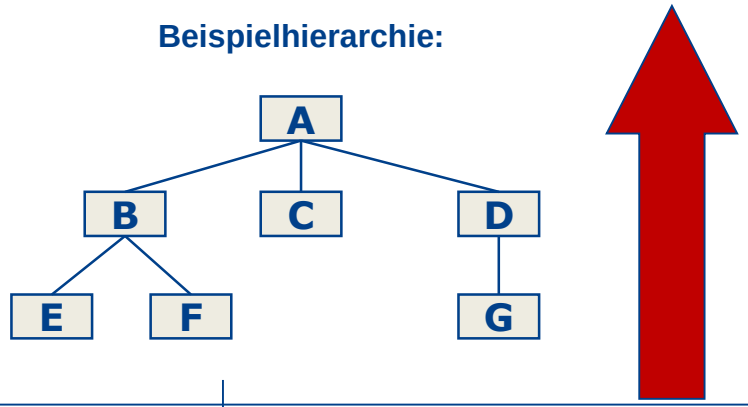
Vorteil

- keine Platzhalter benötigt.

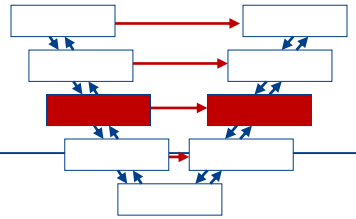
Nachteil

- Übergeordnete Komponenten durch Treiber zu simulieren

Beispielhierarchie:

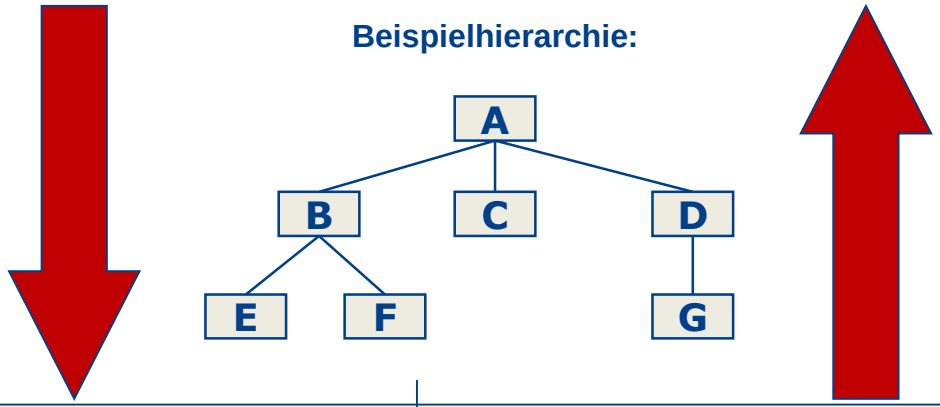


Integrationstest
Integrationsstrategie (4 von 6)

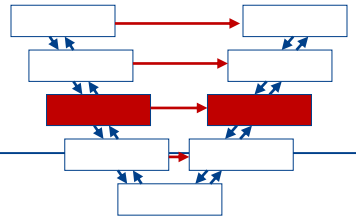


Anmerkungen

- Top-down- oder Bottom-up-Ansatz nur bei hierarchisch gegliederten Systemen einsetzbar (selten anzutreffen)
- in der Praxis meist individuelle Mischung der beiden Integrationsstrategien angewendet
- Je größer der Umfang einer Integration, desto …
  - schwieriger die Isolation von Fehlerzuständen
  - höher der Zeitbedarf zur Fehlerbehebung



Integrationstest
Integrationsstrategie (5 von 6)



Ad-hoc-Integration

- Komponenten in (zufälliger) Reihenfolge ihrer Fertigstellung integrieren
- Direkt nach Komponententest Durchführbarkeit der Integration prüfen

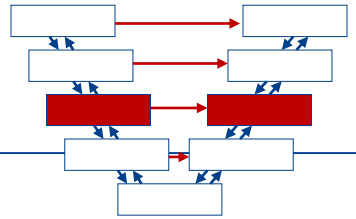
Vorteil

- Zeitgewinn, da frühestmögliche Integration

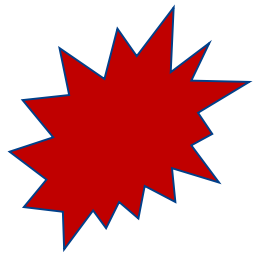
Nachteil

- Platzhalter und Treiber benötigt

Integrationstest
Integrationsstrategie (6 von 6)

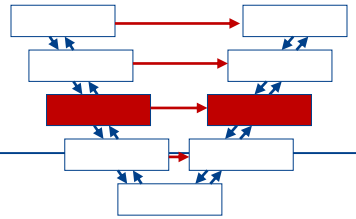


- **Nicht inkrementelle Integration – *big-bang*-Integration**
  - Warten bis alle Softwarebauteile entwickelt sind
  - alles auf einmal integrieren. Schlimmstenfalls ohne Komponententests
  - Steigerung: auch Software- und Hardwareelemente auf einmal integrieren

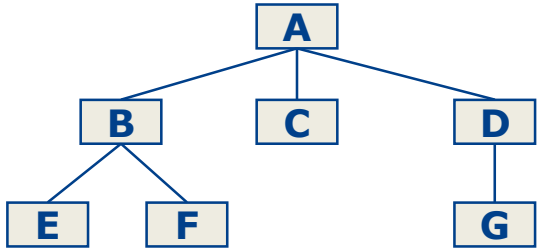


- **Nachteile**
  - Wartezeit bis zum *big-bang* ist verlorene Zeit
    - Test sowieso mit Zeitmangel – keinen einzigen Testtag verschenken
  - Fehlerwirkungen alle auf einmal
  - Sehr schwierig, dass das System überhaupt funktioniert
  - Lokalisierung und Behebung von Fehlerzuständen äußerst schwierig

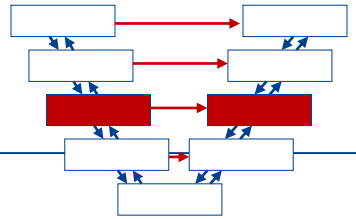
Integrationstest
Integrationsstrategien am Beispiel (1 von 4)



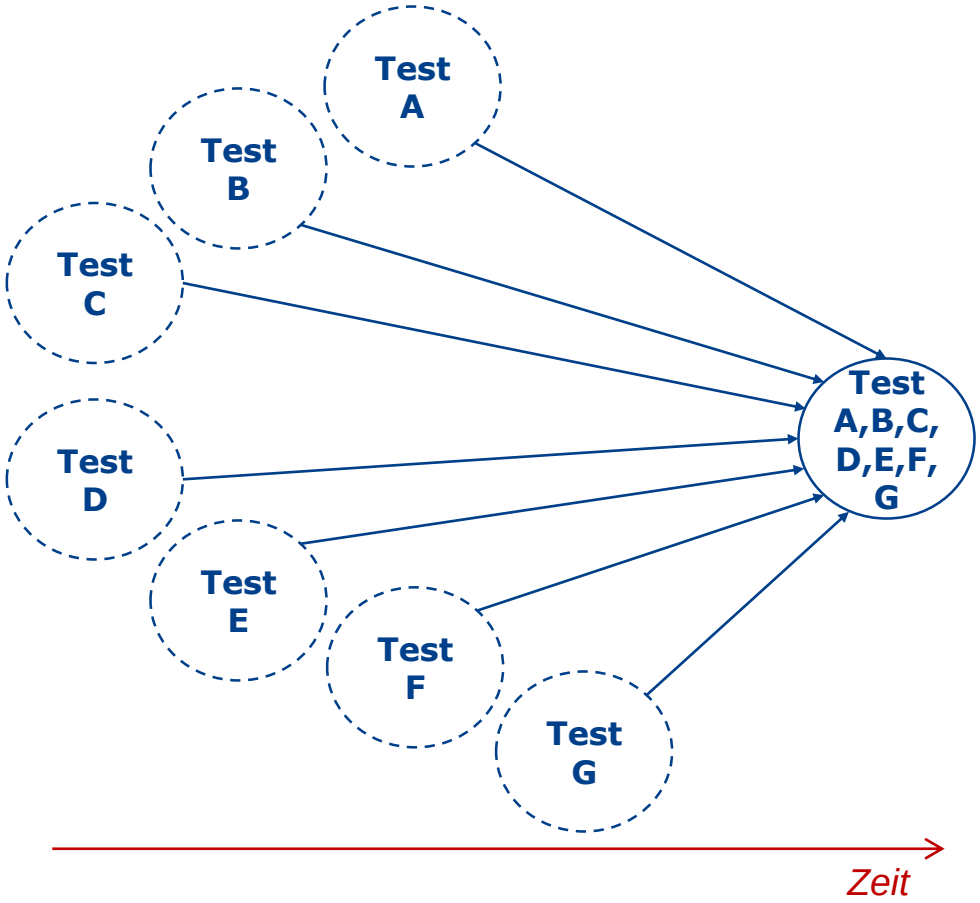
Beispielhierarchie

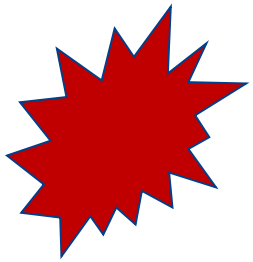


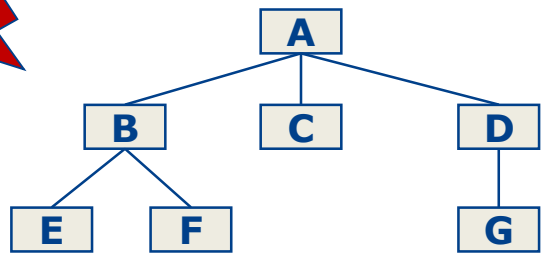
Integrationstest
Integrationsstrategien am Beispiel (2 von 4)



Big-Bang Integration



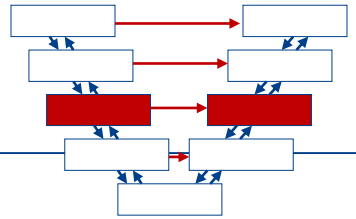




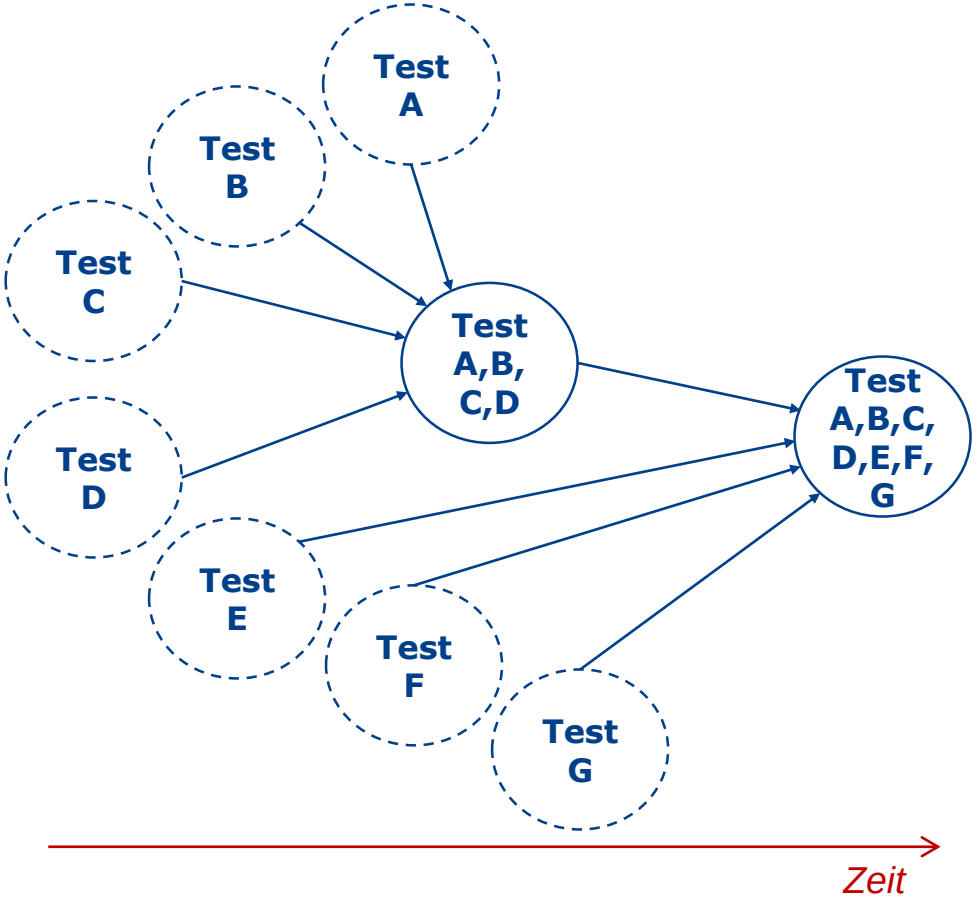
Komponententest

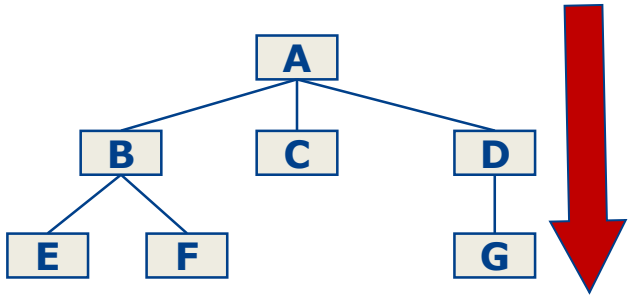
Integrationstest

Integrationstest
Integrationsstrategien am Beispiel (3 von 4)



Top-Down Integration

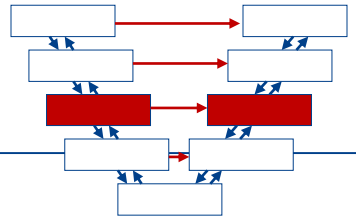




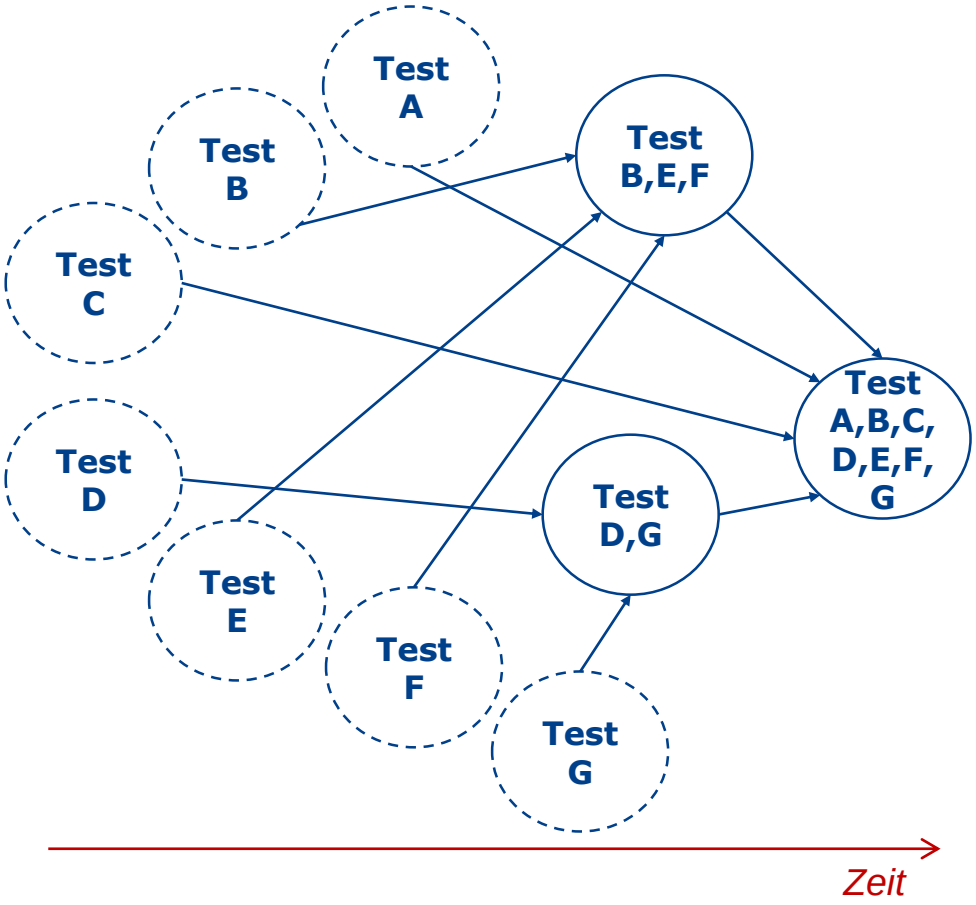
Komponententest

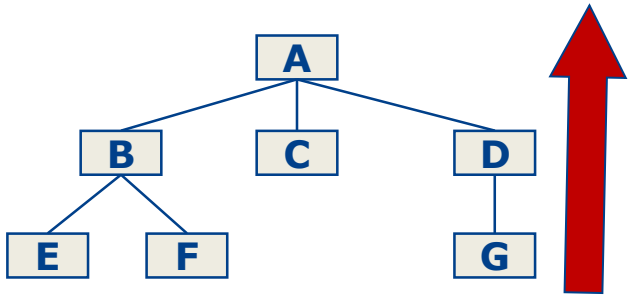
Integrationstest

Integrationstest
Integrationsstrategien am Beispiel (4 von 4)



Bottom-Up Integration

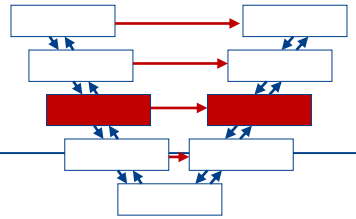




Komponententest

Integrationstest

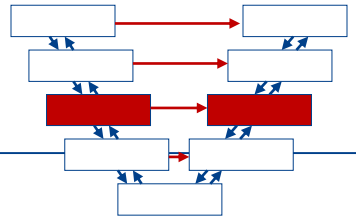
Integrationstest
Auswahl Integrationsstrategie



Integrationsstrategie von projektspezifischen Randbedingungen abhängig:

- Systemarchitektur
  - welche Komponenten mit welchen Abhängigkeiten bilden das System?
- Projektplan
  - Wann welche Komponenten entwickelt und testbereit?
- Testkonzept/Mastertestkonzept
  - Wann welche Systemaspekte auf welcher Teststufe wie intensiv getestet?
- Testmanager
  - Basierend auf Randbedingungen passende Integrationsstrategie aufstellen

Integrationstest
Auswahl Integrationsstrategie



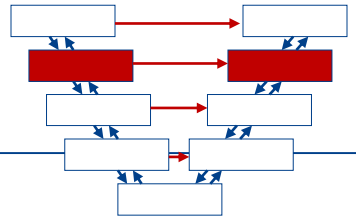
- Je größer die Integration, desto schwieriger die Fehlerfindung
- Somit höheres Risiko und größerer Zeitaufwand für Debugging
- Daher kontinuierliche Integration als gängige Vorgehensweise
  - Software auf Komponentenbasis integriert
  - Oftmals mit automatisierten Regressionstests (auf mehreren Teststufen)

Kapitel 2

Testen im Software-entwicklungs-lebenszyklus

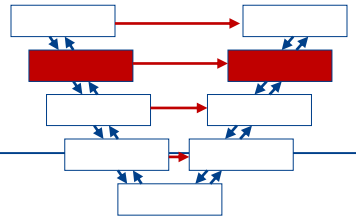
- Softwareentwicklungslebenszyklus-Modelle
- **Teststufen → Systemtest**
- Testarten
- Wartungstest

Systemtest
Begriffsklärung



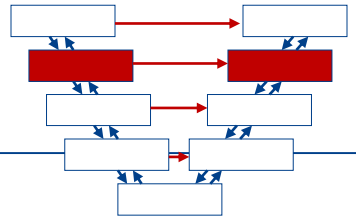
- **Systemtest: nach abgeschlossenem Integrationstest**
- **Fokus: spezifizierte Anforderungen vom Produkt erfüllt?**
- Motivation:
  - Vorherige Teststufen: Prüfung aus der Perspektive des Softwareherstellers
  - Systemtest: Prüfung aus Sicht von Kunden und Anwendern
  - Anforderungen vollständig und angemessen umgesetzt?
  - Systemfunktionen und -eigenschaften oft erst nach Integration aller (!) Systemkomponenten testbar

Systemtest
Testziele (1 von 3)



- **Betrachtung des Systems als Ganzes**
- Risikoreduktion, Finden von Fehlerwirkungen (und Fehlerzuständen)
- Fehlerzustände nicht an höhere Teststufen / Produktion weitergeben
- Teilweise: Verifizierung der Datenqualität
- automatisierte Systemregressionstests zur Bestandssicherung
- Vertrauen in die Qualität des Systems als Ganzes schaffen
- Bereitstellung von Informationen für Freigabeentscheidung

Systemtest
Testziele (2 von 3)



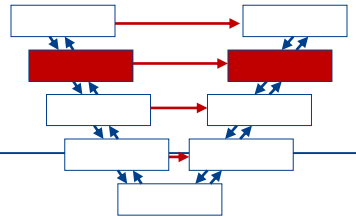
Verifizierung: Erfüllt das System den Entwurf und die Spezifikationen?

Validierung: System vollständig und funktioniert wie erwartet?

Zwei Klassen von Anforderungen:

- **Funktionale Anforderungen**
  - erwartetes Systemverhalten spezifiziert
  - Beschreibt »was« das (Teil)System leisten soll
- **Nicht-funktionale Anforderungen**
  - Beschreibt »wie gut« das (Teil)System funktionieren soll
  - beeinflusst Kundenzufriedenheit stark
  - Anforderungen an Datenqualität ebenfalls im Systemtest prüfen
    - Datenkonvertierungsprojekte
    - Data Warehouses

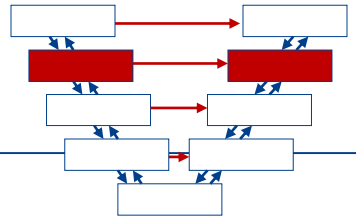
Systemtest
Testziele (3 von 3)



Produkt-Qualitätsmerkmale nach ISO 25010

Funktionalität	Performanz Effizienz	Kompabilität	Benutzbarkeit	Zuverlässigkeit	(Informations-) Sicherheit	Wartbarkeit	Portabilität
Funktionale Vollständigkeit Funktionale Korrektheit Funktionale Angemessenheit (…)	Zeitverhalten Verbrauchs-Verhalten Kapazität (…)	Co-Existenz Interoperabilität (…)	Angemessenheit Zugänglichkeit Verständlichkeit Erlernbarkeit Bedienbarkeit Attraktivität (…)	Reife Verfügbarkeit Fehlertoleranz Wiederherstellbarkeit (…)	Vertraulichkeit Integrität Nicht-Abstreitbarkeit Beweisbarkeit Authenzität (…)	Modularität Wiederverwendbarkeit Analysierbarkeit Modifizierbarkeit Testbarkeit (…)	Anpassbarkeit Installierbarkeit Austauschbarkeit (…)
Funktionaler Test	Lasttest Performanztest Volumentest Stresstest	Konfigurationstest Interoperabilitätstest	Gebrauchstauglichkeitstest	Robustheitstest	Test auf (Daten-) Sicherheit	Wartbarkeitstest Statische Analyse	Installationstest Portabilitätstest.

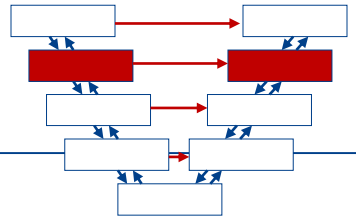
Systemtest
Testbasis



- **Beispiele für Testbasis im Systemtest**
  - (nicht-)funktionale Anforderungsspezifikationen
  - Anwendungsfälle, Epics und User-Stories
  - Risikoanalyseberichte
  - Modelle des Systemverhaltens
  - Dokumentation und Benutzeranleitungen

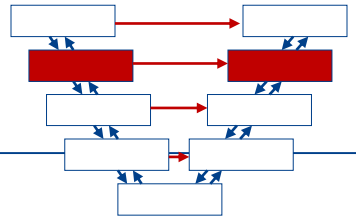
- **Systemtests oft von unabhängigen Tester*innen durchgeführt**
  - Fehlerzustände in Testbasis können zu Verständnisproblemen führen
  - Unstimmigkeiten über das erwartete Systemverhalten
  - „falsch positive“ und „falsch negative“ Testergebnisse möglich mit Auswirkungen auf Effizienz
  - Verhindern dieser Situationen: frühes Einbeziehen von Testern in Reviews

Systemtest
Testobjekte



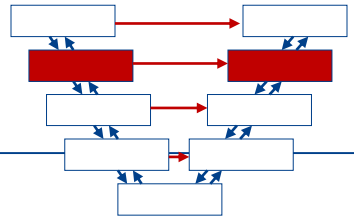
- **Typische Testobjekte für Systemtests**
  - Anwendungen
  - Hardware/Softwaresysteme
  - Betriebssysteme
  - Systeme unter Test (SUT)
  - Systemkonfiguration und Konfigurationsdaten

Systemtest
Typische Fehlerzustände und Fehlerwirkungen

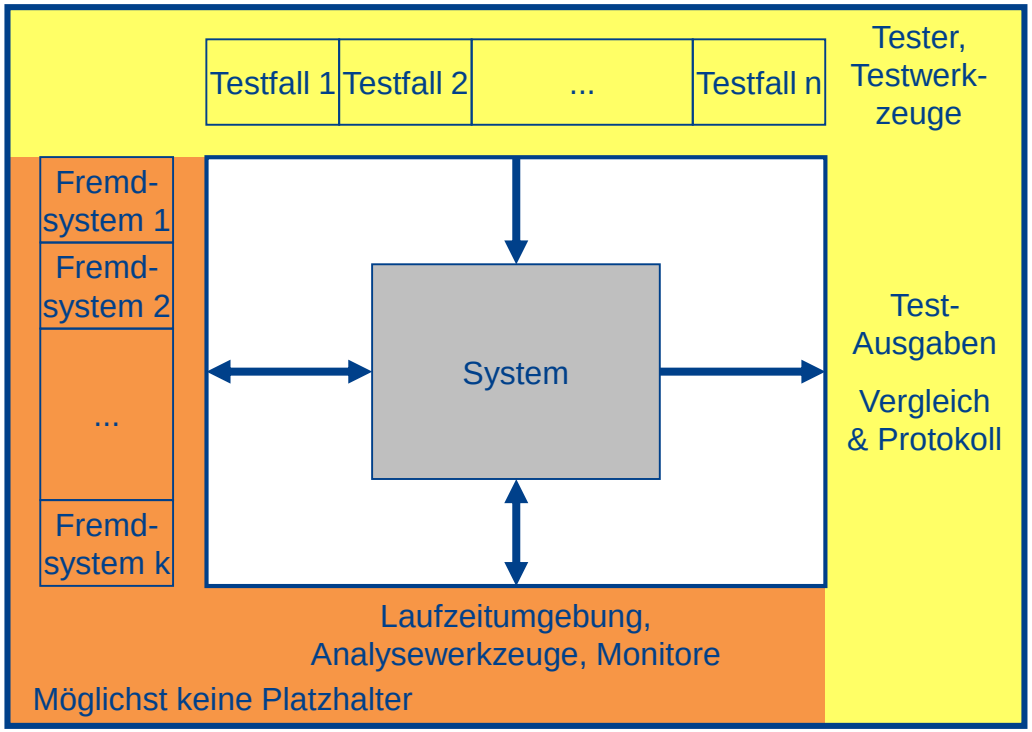


- **typische Fehlerzustände und Fehlerwirkungen im Systemtest**
  - Falsche Berechnungen
  - Falsche / unerwartete (nicht-)funktionale Systemverhaltensweisen
  - Falsche Kontroll- und/oder Datenflüsse innerhalb des Systems
  - Versagen bei der korrekten oder vollständigen Ausführung von funktionalen End-to-End-Aufgaben
  - Versagen des Systems bei der ordnungsgemäßen Arbeit in der Produktivumgebung
  - System funktioniert nicht wie in Dokumentation beschrieben

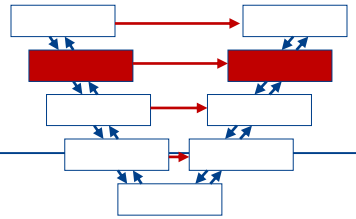
Systemtest
Testumgebung (1 von 2)



- Testumgebung möglichst nahe der späteren Produktivumgebung
- möglichst die tatsächlich zum Einsatz kommende Hard- oder Software nutzen

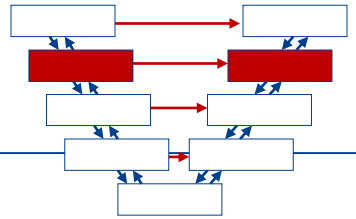


Systemtest
Testumgebung (2 von 2)



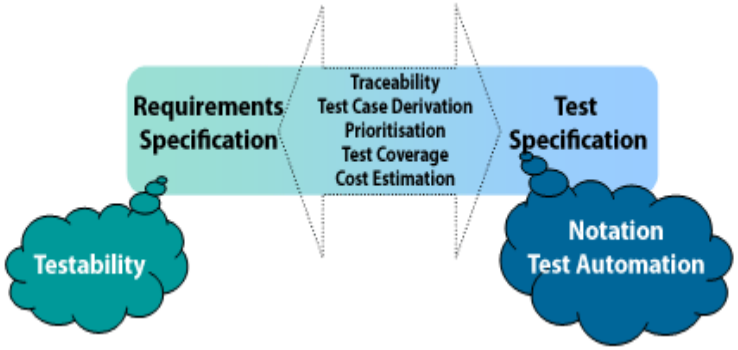
- Systemtest für datenbankgestützte Informationssysteme oft in Produktivumgebung des Kunden durchführen
  - Kosten und Aufwand sparen
- Nachteile
  - Fehlerwirkungen können Produktivumgebung des Kunden beeinträchtigen
  - Mögliche Folge: teure Systemausfälle und Datenverluste im Kundensystem
  - keine / geringe Kontrolle über Konfiguration der Produktivumgebung; dadurch Änderung der Testbedingungen durch parallel zum Test laufenden Betrieb
  - durchgeführte Systemtests sind schwer / nicht mehr reproduzierbar

Systemtest – Teststrategie
Funktionale Anforderungen (1 von 3)



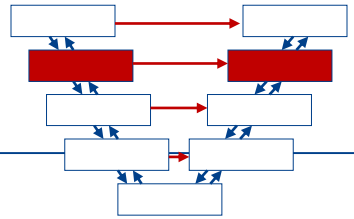
- Anforderungsdefinitionen in Anforderungsdokument dokumentiert

- **Anforderungsbasiertes Testen**
  - Testbasis: Anforderungsdokument
  - Systemtestspezifikation durch Review verifiziert
  - Pro Anforderung min. ein Systemtestfall abgeleitet und dokumentiert
  - Normalerweise mehr als ein Testfall, um eine Anforderung zu testen



Source: https://se.ifi.uni-heidelberg.de/people/timea_illes_seifert.html

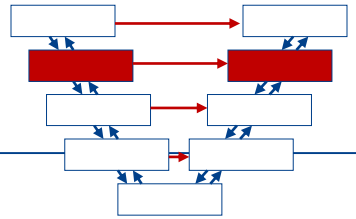
Systemtest – Teststrategie
Funktionale Anforderungen (2 von 3)



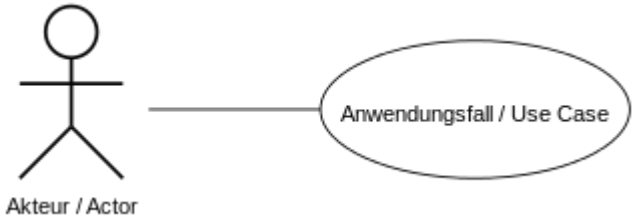
- **Geschäftsprozessbasiertes Testen**
  - Softwaresystem soll Geschäftsprozess automatisieren / unterstützen
  - Geschäftsprozessanalyse zeigt die relevanten Geschäftsprozesse (inkl. Kontext, Personen, Firmen, Fremdsysteme, …)
  - Testszenarien erstellen, die typische Geschäftsprozesse nachbilden
  - Priorität an Häufigkeit / Relevanz der Geschäftsprozesse orientiert

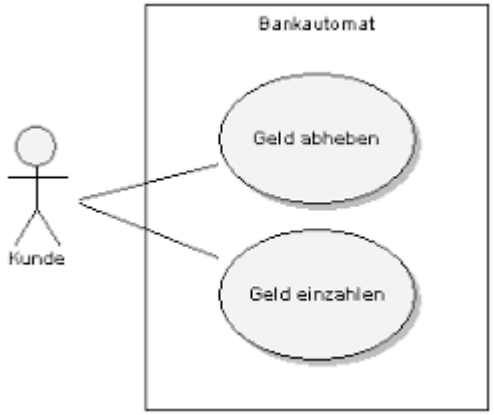
- Fokus: Abläufe, hintereinander geschaltete Tests

Systemtest – Teststrategie
Funktionale Anforderungen (3 von 3)



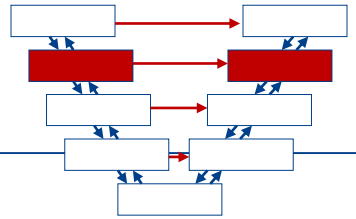
- **Anwendungsfallbasiertes Testen**
  - Systemtestfälle bilden typischen Umgang mit System ab
  - Benutzerprofil pro Anwendergruppe mit typischem Aktionsmuster
  - Testszenarien aus Aktionsmustern ableiten
  - Priorität der Testszenarien an Häufigkeit der Aktionen im späteren Betrieb orientiert





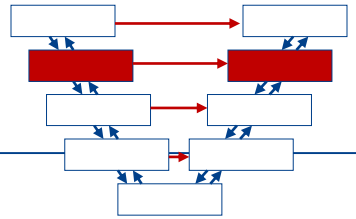
Quelle: http://www.se.uni-hannover.de/pub/File/pdfpapers/Crisp2006.pdf

Systemtest – Teststrategie
Nicht-funktionale Anforderungen (1 von 4)



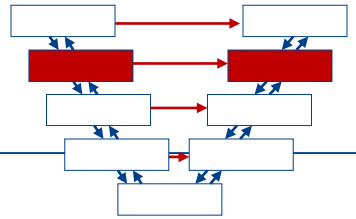
- Auch nicht-funktionale Anforderungen legen qualitative Aspekte fest
- nicht-funktionale Systemeigenschaften in Tests berücksichtigt
- Performanztest
  - Messung der Antwortzeit für Anwendungsfälle (mit steigender Last)
- Lasttest
  - Verhalten eines Systems unter wechselnder Last – üblicherweise zwischen niedriger Last, typischer Last sowie Spitzenlast

Systemtest – Teststrategie
Nicht-funktionale Anforderungen (2 von 4)



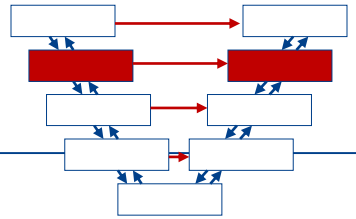
- Volumen-/Massentest
  - Systemverhalten in Abhängigkeit zu Datenmenge prüfen (z.B. für sehr großer Dateien)
- Stresstest
  - Beobachtung des Systemverhaltens bei Überlastung
- Test der (Daten-)Sicherheit
  - gegen unberechtigten Systemzugang oder Datenzugriff
- Zuverlässigkeitstest
  - Dauerbetrieb (Ausfälle pro Betriebsstunde bei Benutzungsprofil x, ...)
- Robustheitstest
  - gegenüber Fehlbedienung, Fehlprogrammierung, Hardwareausfall
  - Prüfung von Fehlerbehandlung und Wiederanlaufverhalten

Systemtest – Teststrategie
Nicht-funktionale Anforderungen (3 von 4)



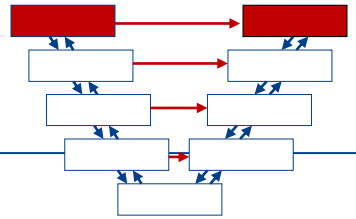
- Kompatibilitätstest / Datenkonversionstest
  - Verträglichkeit mit vorhandenen Systemen prüfen
  - Import/Export von Datenbeständen
- Konfigurationstest
  - Fokus: unterschiedliche Konfigurationen des Systems
- Gebrauchstauglichkeitstest / Benutzbarkeitstest
  - Prüfung der Bedienbarkeit, Verständlichkeit der Systemausgaben, …
- Prüfung der Dokumentation
  - Übereinstimmung mit Systemverhalten (z.B. Bedienungsanleitung)
- Prüfung auf Änderbarkeit/Wartbarkeit
  - Verständlichkeit der Entwicklungsdokumente, Systemstruktur, usw.

Systemtest – Teststrategie
Nicht-funktionale Anforderungen (4 von 4)



- nicht-funktionaler Test mit ungenauen, »schwammigen« Anforderungen
  - Nicht testbare Formulierungen
    - »Das System soll leicht bedienbar sein«
    - »Das System soll schnell reagieren«
  - nicht-funktionale Anforderungen oft zu selbstverständlich für Dokumentation
  - Auch nicht spezifizierte, aber dennoch relevanten Eigenschaften validieren

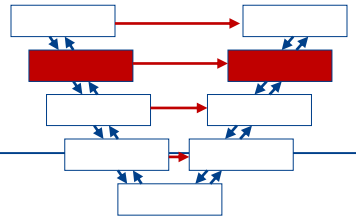
Systemtest
Testen gegen …



Produkt-Qualitätsmerkmale nach ISO 25010

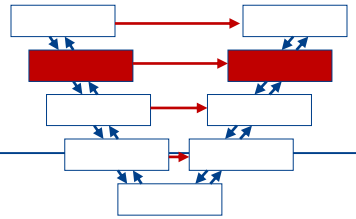
Funktionalität	Performanz Effizienz	Kompabilität	Benutzbarkeit	Zuverlässigkeit	(Informations-) Sicherheit	Wartbarkeit	Portabilität
Funktionale Vollständigkeit Funktionale Korrektheit Funktionale Angemessenheit (…)	Zeitverhalten Verbrauchs-Verhalten Kapazität (…)	Co-Existenz Interoperabilität (…)	Angemessenheit Zugänglichkeit Verständlichkeit Erlernbarkeit Bedienbarkeit Attraktivität (…)	Reife Verfügbarkeit Fehlertoleranz Wiederherstellbarkeit (…)	Vertraulichkeit Integrität Nicht-Abstreitbarkeit Beweisbarkeit Authenzität (…)	Modularität Wiederverwendbarkeit Analysierbarkeit Modifizierbarkeit Testbarkeit (…)	Anpassbarkeit Installierbarkeit Austauschbarkeit (…)
Funktionaler Test	Lasttest Performanztest Volumentest Stresstest	Konfigurationstest Interoperabilitätstest	Gebrauchstauglichkeitstest	Robustheitstest	Test auf (Daten-) Sicherheit	Wartbarkeitstest Statische Analyse	Installationstest Portabilitätstest.

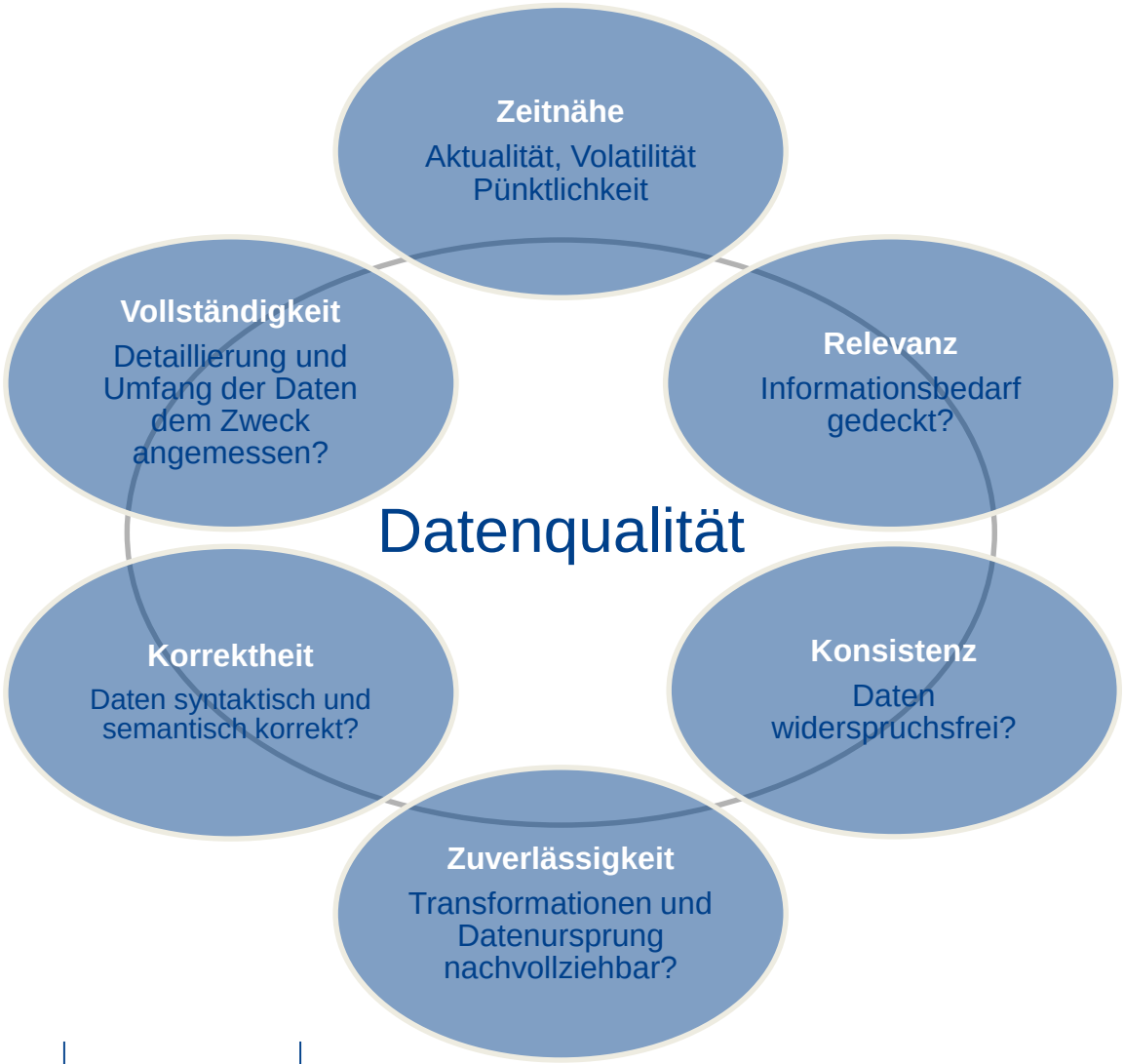
Systemtest
Anforderungen an die Datenqualität



- Datenflut steigt stetig an (Big Data), z.B. Finanz-, Material-, Adressdaten, …
- Hohe Anforderung an Datenqualität, da falsche Daten Schaden verursachen
- Mangelnde Datenqualität entsteht
  - bei Eingabe
    - Daten: falsch erfasst, nicht oder unvollständig geprüft
    - Schnittstellen: schlecht benutzbar, keine Validierung, fehlende Standards
    - Umgang mit Daten: unverantwortlich, fehlendes Problembewusstsein
  - bei Integration unabhängiger Systeme
    - Keine Datenarchitektur
    - redundante, heterogene Daten
  - bei Migration
    - Migrationsfehler (Datenverlust, semantische Inkompatibilität)
  - durch fehlende Verantwortlichkeiten.

Systemtest
Anforderungen an die Datenqualität





Anmerkungen

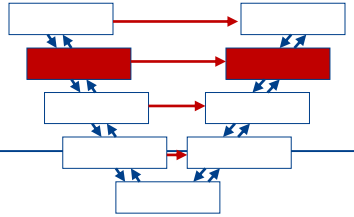
Anforderungen an Datenqualität projektspezifisch erheben und spezifizieren

Aktualität: Zeit bis Anpassung im Informationssystem an Änderung in Realität

Volatilität: Gültigkeitszeitraum eines Zustands

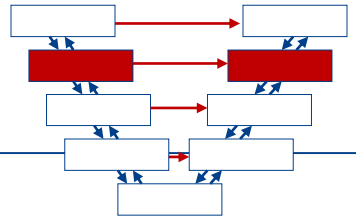
Pünktlichkeit: Lieferzeitpunkt der Daten aus Quellsystem

Datenqualitätsanalyse



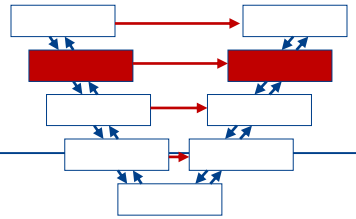
- Automatisiert
  - Ermittlung statistischer Kennzahlen, die auf ein Problem hindeuten
  - Identifikation fehlerhafter Daten
  - Nutzung von Profiling-Werkzeugen
- Manuell/semi-automatisiert durch interdisziplinäres Team
  - Auswertung der Analyse
  - Erstellen der Liste von Geschäftsregeln, für Datenbereinigung

Dauerhafte Daten-Qualitätssicherung



- Organisatorisch
  - Data Owner: Verantwortlicher für Datenqualität
  - Konstruktive Qualitätssicherung zur Vorbeugung
  - Analytische Qualitätssicherung zur Identifikation

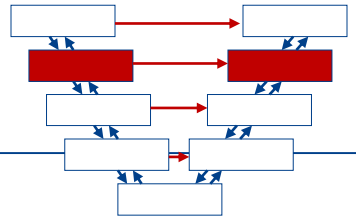
Systemtest
Probleme (1 von 3)



Unklare Kundenanforderungen

- Systemverhalten ist ohne Anforderungen nicht bewertbar
- Anwender hat eine Vorstellung davon, was er erwartet (=Anforderungen)
- diese aber nirgends nachlesbar, nur »in den Köpfen« einiger Personen
- Tester müssen dann diese Informationen nachträglich zusammenzutragen

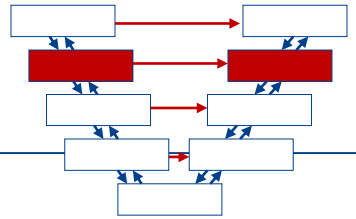
Systemtest
Probleme (2 von 3)



Versäumte Entscheidungen

- Nachträgliches Sammeln von Anforderungen offenbart unterschiedliche Ansichten und Vorstellungen
- Ohne vorherige schriftliche Abstimmung ist dies nicht verwunderlich
- Also: langwierige Entscheidungsprozesse zu spätem Zeitpunkt erzwingen
- sehr zeit- und kostenintensives Vorgehen
- Folge: Fertigstellung des System(test)s stark verzögert

Systemtest
Probleme (3 von 3)



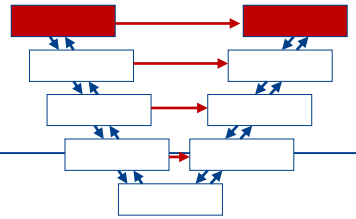
Projekte scheitern

- Ohne Anforderungen fehlen auch Entwicklern klare Ziele
- Wahrscheinlichkeit für korrektes System außerordentlich gering
- Niemand sollte unter solchen Bedingungen auf Projekterfolg hoffen
- Systemtest attestiert hier nur das Scheitern des Projekts »offiziell«

Testen im Software-entwicklungs-lebenszyklus

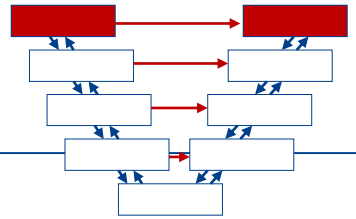
- Softwareentwicklungslebenszyklus-Modelle
- **Teststufen → Abnahmetest**
- Testarten
- Wartungstest

Abnahmetest
Begriffsklärung



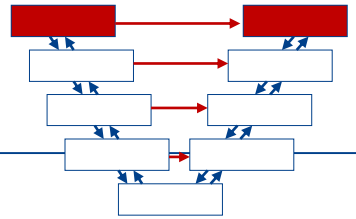
- Abnahmetest: abschließender Test vor Inbetriebnahme der Software
- spezielle Form des Systemtests
- häufig in Verantwortung von Kunde, Anwender, Product Owner, …
- u.U. der einzige Test, an dem der Kunde direkt beteiligt ist
  - bisher beschriebene Teststufen verantwortet der Hersteller
- Fokus:
  - Sicht und Urteil des Kunden bzw. Anwenders
  - Erfüllung rechtlicher oder regulatorischer Anforderungen oder Standards
- Ziele:
  - Vertrauen in Qualität des Systems schaffen
  - Validieren des Systems: System funktioniert wie erwartet?
  - Verifizieren des Systems: (nicht-)funktionale Anforderungen erfüllt?

Abnahmetest – Testziele
Regulatorischer und vertraglicher Abnahmetest



- Kunde entscheidet beim Abnahmetest:
  - bestelltes Softwaresystem mangelfrei?
  - Entwicklungsvertrag / vertraglich geschuldete Leistung erfüllt?
- auch Vertrag oder Projektauftrag zwischen beauftragender Fachabteilung und realisierender IT-Abteilung einer Firma oder eines Konzerns möglich
- Software gegen vertragliche Abnahmekriterien prüfen
  - Testkriterien: vertraglich festgeschriebene Abnahmekriterien
- Regulatorische Abnahmetests:
  - System zu relevanten Gesetzen, Richtlinien und Vorschriften konform?
- Hauptziel von vertraglichen oder regulatorischen Abnahmetests:
  - Vertrauen in vertragliche oder regulatorische Konformität aufbauen

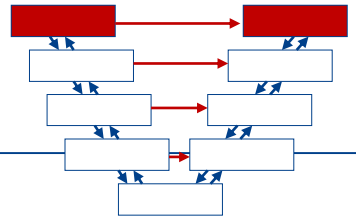
Abnahmetest
Vertraglicher Abnahmetest (1 von 2)



Praxis:

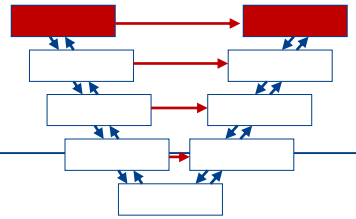
- Softwarehersteller prüft Abnahmekriterien bereits im Systemtest
- Abnahmetest: abnahmerelevante Testfälle für Kundendemo wiederholen
- Abnahmetests in Abnahmeumgebung von Kunden durchführen
- Andere Testumgebung: Testfall kann fehlschlagen, der vorher funktionierte

Abnahmetest
Vertraglicher Abnahmetest (2 von 2)



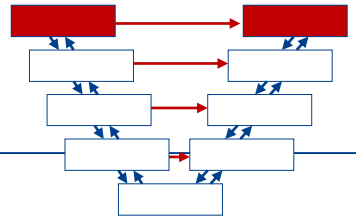
- Daher: Abnahmeumgebung so nah wie möglich an Produktivumgebung
- Test in Produktivumgebung wegen Risiko für laufenden Betrieb vermeiden
- Testfallermittlung/-entwurf wie bisher im Systemtest
- auch Geschäftsvorfälle einer typischen Abrechnungsperiode aufnehmen

Abnahmetest – Betrieblicher Abnahmetest (Operational Acceptance Testing)



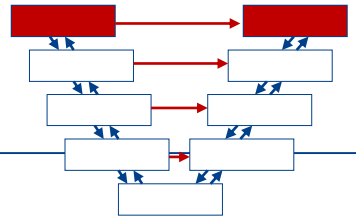
- Abnahmetest üblicherweise in simulierter Produktivumgebung durchführen
- Fokus des Tests auf betrieblichen Aspekten
  - Backups und Wiederherstellungen
  - Installieren, Deinstallieren, Aktualisieren, Wartung
  - Notfallwiederherstellung (Disaster-Recovery)
  - Benutzerverwaltung
  - Datenlade- und Migrationsaufgaben
  - Prüfen von Sicherheitsschwachstellen
  - Performanz
- Hauptziel betrieblicher Abnahmetest
  - Vertrauen aufbauen, dass System in betrieblicher Umgebung auch unter schwierigen Bedingungen funktionsfähig bleibt

Abnahmetest – Benutzerabnahmetest
User Acceptance Testing (1 von 2)



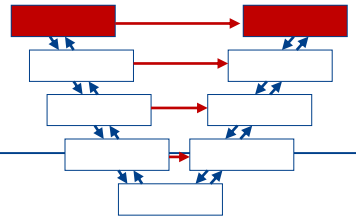
- Hauptziel: Vertrauen schaffen, dass das System…
  - die Bedürfnisse und Anforderungen der Benutzer erfüllt
  - die Geschäftsprozesse ohne Schwierigkeiten, Kosten oder Risiken ausführt
- Benutzerabnahmetest empfehlenswert wenn verschiedene Anwender
  - unterschiedliche Anwendergruppen mit verschiedenen Erwartungen
  - Ablehnung durch eine Gruppe kann Systemeinführung scheitern lassen
  - Daher Benutzerakzeptanztest für jede Anwendergruppe
  - meist vom Kunden organisiert, der auch die Testfälle auswählt

Abnahmetest – Benutzerabnahmetest
User Acceptance Testing (2 von 2)



- gravierende Akzeptanzprobleme erst im Abnahmetest?
  - meist nur noch Kosmetik möglich
- Vermeidung: Anwendern in frühen Projektphasen Prototypen vorstellen
- agile Entwicklungsmodelle: Product Owner oder Kundenvertreter im Team

Abnahmetest
Testbasis



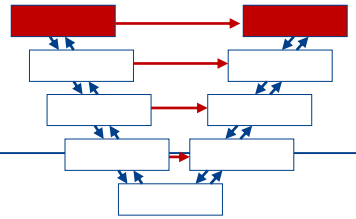
Testbasis für Abnahmetests:

- Geschäftsprozesse, Anforderungen, Anwendungsfälle, User Stories
- Vorschriften, rechtliche Verträge und Standards
- System- oder Benutzerdokumentation, Installationsverfahren
- Risikoanalyseberichte

Darüber hinaus:

- Verfahren für Sicherung, Wiederherstellung, Disaster Recovery
- nicht-funktionale Anforderungen
- Datenbankpakete

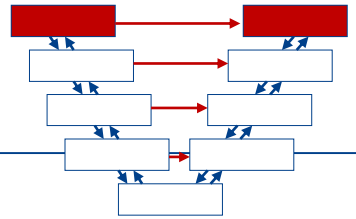
Abnahmetest
Testobjekte



Typische Testobjekte:

- System unter Test (SUT)
- Systemkonfigurationen und Konfigurationsdaten
- Geschäftsprozesse
- Wiederherstellungssysteme und Hot Sites (Betriebskontinuität und Notfallwiederherstellung)
- Betriebs- und Wartungsprozesse
- Formulare
- Berichte
- Bestehende und konvertierte Produktionsdaten

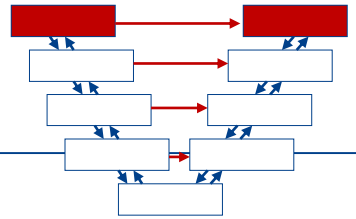
Abnahmetest
Typische Fehlerzustände und Fehlerwirkungen



typische Fehlerzustände und –wirkungen:

- Nichterfüllung der Fach- oder Benutzeranforderungen
- Nichteinhaltung der Geschäftsregeln
- Nichterfüllung der vertraglichen oder regulatorischen Anforderungen
- Nicht-funktionale Einschränkungen: Informationssicherheit, Performanz, …

Alpha- und Beta-Test (1 von 3)



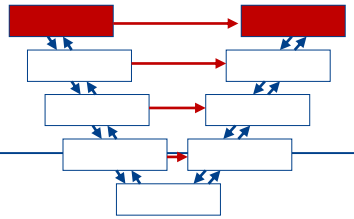
- Systemtests für sehr viele verschiedene Produktivumgebungen sehr kostenintensiv bis unmöglich

- **Dann: vor Abnahmetest noch Alpha- und Beta-Tests durchführen**

- Ziele
  - Fehler durch unbekannte / nicht spezifizierte Produktivumgebungen erkennen und ggf. beheben
  - Vertrauen bei Kunden aufbauen, dass sie System nutzen können

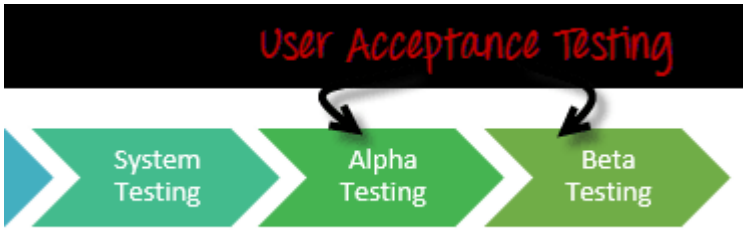
Kuhtransport mal anders: Zwei Rinder auf Autorückbank

Alpha- und Beta-Test (2 von 3)

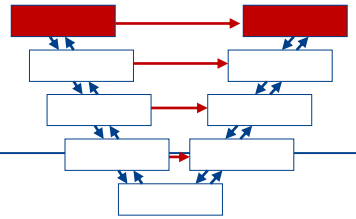


- stabile Vorabversionen an repräsentative Kunden geben
- Anwendung durch Kunden
  - Durchführung vorgegebener Tests
  - probehalber Einsatz des Produkts unter realistischen Bedingungen
  - Meldung von Kommentaren, Eindrücken und Fehlermeldungen

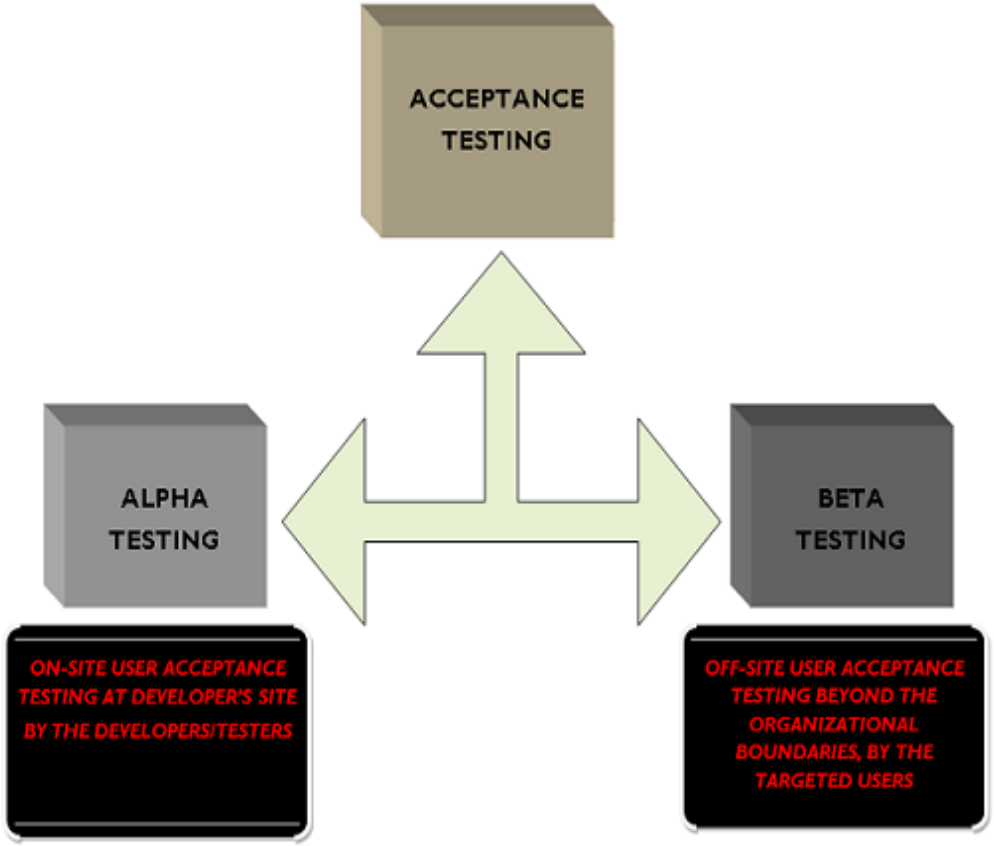
- **Unterschied von Alpha- und Beta-Tests**
  - Alpha-Tests beim Hersteller durch Benutzer und unabhängigem Testteam
  - Beta-Tests (Feldtests) von Kunden an ihren eigenen Standorten durchgeführt



Alpha- und Beta-Test (3 von 3)

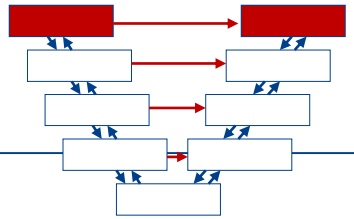


- Alpha- und Beta-Tests dürfen Systemtest nicht ersetzen
- Erst nach erfolgreichem Systemtest vorläufiges Produkt an Kunden geben
- Beta-Tests auch ohne vorhergehende Alpha-Tests durchführbar
  - wenn Vertrauen in Produkt bereits ausreichend



SOURCE: :http://www.professionalqa.com/alpha-vs-beta-testing

Abnahmetest und iterative Softwareentwicklung



- Abnahmetest oft letzte Stufe in Entwicklungslebenszyklus
- aber auch zu anderen Zeitpunkten möglich
  - Für Standardsoftwareprodukte zum Zeitpunkt der Installation oder Integration
  - Für neue funktionale Verbesserung vor dem Systemtest
- In iterativer Entwicklung verschiedene Formen der Abnahmetests möglich
  - Verifizierung, dass ein Feature Abnahmekriterien erfüllt
  - Validieren, dass ein Feature die Bedürfnisse der Benutzer erfüllt
- Zeitpunkt: am Ende oder nach jeder Iteration, nach mehreren Iterationen

Testen im Softwareentwicklungslebenszyklus

- Softwareentwicklungslebenszyklus-Modelle
- Teststufen
- **Testarten**
- Wartungstest

Testarten
Begriffsklärung

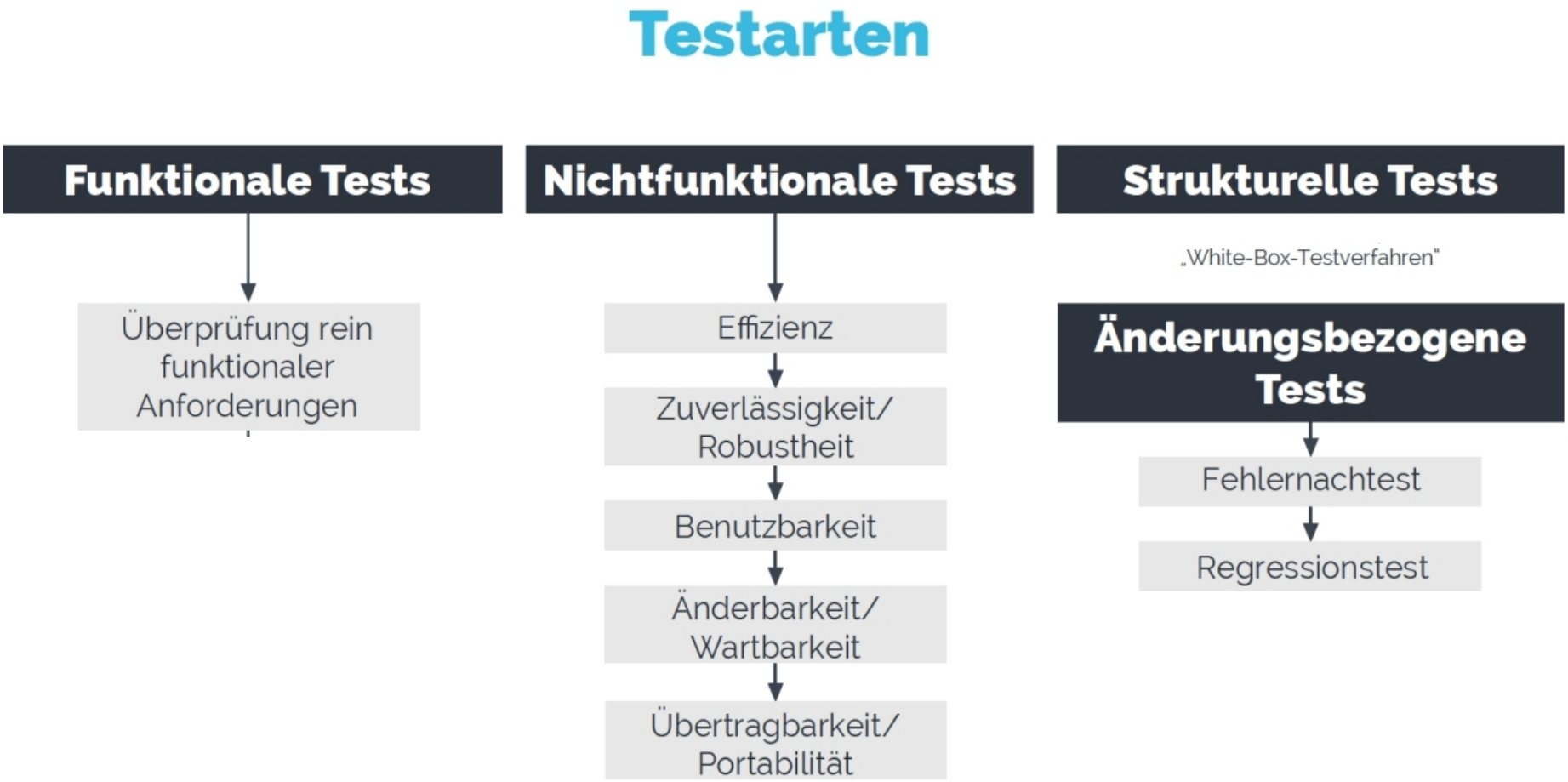
Testarten gruppieren Testaktivitäten nach Testzielen:

- funktionale Qualitätsmerkmale: vollständig, korrekt, angemessen?
- nicht-funktionale Qualitätsmerkmale: zuverlässig, performant, nutzbar?
- Korrektheit und Vollständigkeit der Architektur der Komponente
- Auswirkung von Änderungen

grundlegende Testarten:

- **Funktionale Tests**
- **Nicht-funktionale Tests**
- **White-Box-Tests (Softwarestruktur/Softwarearchitektur)**
- **Änderungsbasierte Tests (Fehlernachtests, Regressionstests)**

Testarten
Begriffsklärung



Teststufen und Testarten

- Alle Testarten auf allen Teststufen möglich
  - Komponententest
  - Integrationstest
  - Systemtest
  - Abnahmetest
- Testziel je nach Teststufe unterschiedlich
  - Komponententest: Korrektheit, Effizienz
  - Integrationstest: Funktionale Angemessenheit, Fehlertoleranz
  - System- / Abnahmetest: Vollständigkeit, Gebrauchstauglichkeit, Kompatibilität
- Pro Teststufe verschiedene Testarten in unterschiedlicher Intensität

Funktionale Tests

- Funktionalität: »was« soll das System leisten?
- Testbasis
  - Komponententest: Spezifikation für Komponente oder API
  - Integrationstest: Spezifikation für Architektur oder API
  - System- / Abnahmetest: Anforderungen, Anwendungsfälle
- Funktionalität nicht immer dokumentiert (z.B. berechtigte Erwartungen?)
- Funktionaler Test prüft von außen sichtbares Verhalten
- Anwendung spezifikationsorientierter Testverfahren
- Funktionale Tests auf allen Teststufen durchzuführen

Nicht-funktionale Tests

- Nicht-funktionale Tests: »Wie gut« arbeitet das System?
  - Performanztest
  - Lasttest
  - Stresstest
  - Benutzbarkeitstest
  - Wartbarkeitstest
  - Zuverlässigkeitstest
  - Portabilitätstest
- in allen Teststufen anwendbar
- prüfen das von außen sicht- oder messbare Verhalten der Software
- Grundlage: Qualitätsmodelle wie ISO 25010

White-Box-Tests (1 von 2)

- White-Box-Tests (strukturbasierte Tests)
  - basieren auf interner Struktur / Architektur der Software
- Mögliche Grundlagen
  - Kontroll- oder Datenfluss innerhalb von Komponenten
  - Aufrufhierarchie von Prozeduren oder Menüstrukturen
  - Modelle der Software (z.B. Zustandsautomaten)
- Ziel: Überdeckung der betrachteten Strukturelemente durch Tests
- Mittel: Entwurf angemessener Testfälle
- White-Box-Tests
  - Komponenten- und Integrationstest im Einsatz
  - auch in höheren Teststufen (für Menüstrukturen, durch Zustandsautomaten)

White-Box-Tests (2 von 2)

Qualität von White-Box-Tests durch strukturelle Überdeckung messen

- Komponententest: Codeüberdeckung misst z.B. prozentualen Anteil ausgeführter Codezeilen
- Verschiedene Aspekte messbar: z.B. Anweisungen, Entscheidungen
  - Komponentenintegrationstests: auf Architektur des Systems basierend, z.B. die Schnittstellen zwischen Komponenten

White-Box-Testentwurf und -durchführung erfordern spezielles Wissen

- Aufbau des Codes (z.B. für Codeüberdeckungsmessung)
- Datenspeicherung (z.B. für Bewertung von Datenbankanfragen)
- Nutzung von Überdeckungswerkzeugen und Interpretation der Ergebnisse

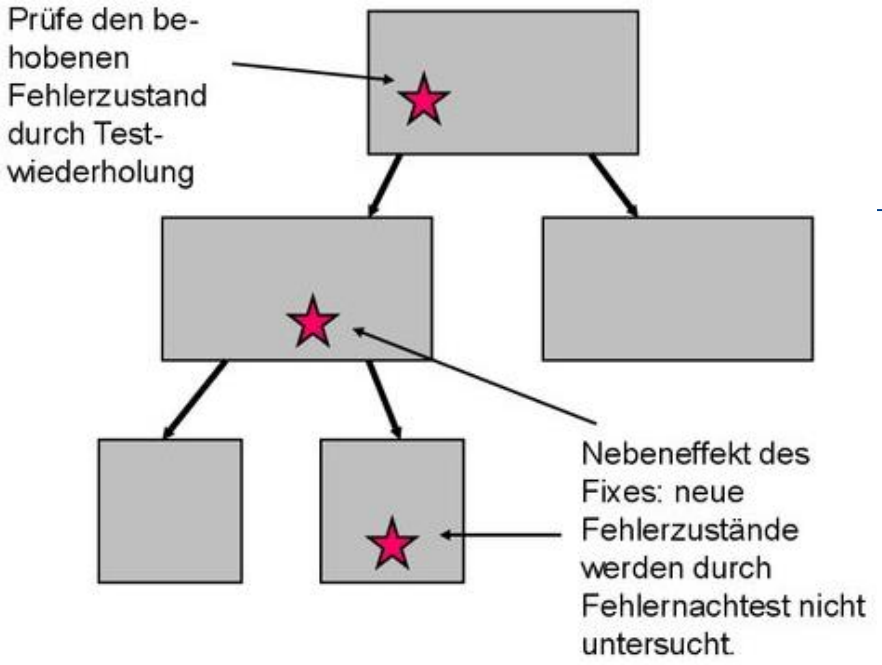
Änderungsbasierte Tests

Testen bei Änderungen

- Fehlernachtest
  - Test, um nach Fehlerwirkung Beseitigung von Fehlerzustand zu prüfen
- Regressionstest
  - Test, um Einbringen von Fehlerzuständen nach Änderungen zu verhindern / entdecken
- Fehlernachtests und Regressionstests in allen Teststufen (wiederholt) ausführbar sein
- Wichtig in iterativen / inkrementellen Entwicklungsmodellen
  - Neue Features & Änderungen erfordern änderungsbasierte Tests
- Anwendung Internet der Dinge (IoT)
  - häufiges Ersetzen oder Aktualisieren von Objekten (z.B. Endgeräte)

Fehlernachtest

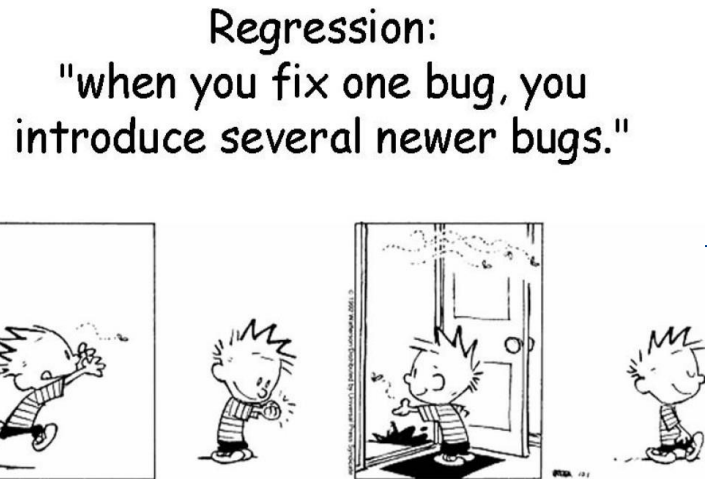
- Fehlerzustand korrigiert: nun Durchführung der Tests, die wegen des Fehlerzustands fehlschlugen
- neue Tests möglich, falls Fehlerzustand das Fehlen einer Funktion war
- Zumindest die die Fehlerwirkung provozierenden Schritte durchführen
- Ziel: Bestätigung, dass Fehlerzustand behoben wurde



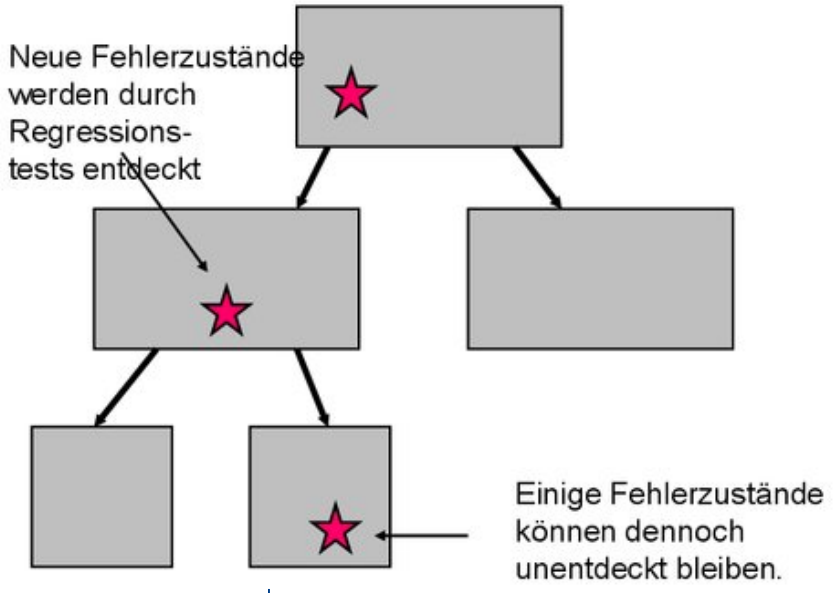
Nachtest beruht auf der exakten Wiederholbarkeit bzgl. Testumgebung, SW-Konfiguration, Eingaben und Voraussetzungen.

Regressionstest (1 von 4)

- Wartungsarbeiten und Weiterentwicklung ändern und ergänzen Software
- geänderte Software muss erneut getestet werden: Regressionstests
- Regressionstest: erneuter Test bereits getesteter Software
- Ziel: keine neuen / bisher maskierten Fehlerzustände in nicht geänderten Bereichen
- Regressionstests auch wenn Softwareumgebung geändert
- Umfang des Regressionstest? Auswirkungsanalyse durchführen!



Erneuter Test eines bereits getesteten Programms bzw. einer Teilfunktionalität nach deren Modifikation, mit dem Ziel nachzuweisen, dass durch die vorgenommenen Änderungen keine Fehlerzustände eingebaut oder (bisher maskierte Fehler) freigelegt wurden.



Regressionstest (2 von 4)

- Umfang des Regressionstests
  1. alle Tests, die durch Änderung behobene Fehlerwirkung erzeugt haben?
  2. Test aller geänderten Codezeilen?
  3. Test aller neu eingefügten Programmteile?
  4. komplettes System (vollständiger Regressionstest)?
- reine Fehlernachtest (1) und nur am »Ort« der Modifikation (2, 3) zu wenig
- scheinbar simple lokale Änderungen mit unerwarteten Auswirkungen
- Seiteneffekte auf beliebige (auch weit entfernte) Software möglich

Regressionstest (3 von 4)

- Vollständiger Regressionstest
  - Wiederholung aller vorhandenen Testfälle
  - gleiche Aussagekraft wie Test an Ausgangsversion der Software
  - ebenfalls notwendig, wenn Systemumgebung geändert
- Praxis: vollständiger Regressionstest fast immer zu teuer

Regressionstest (4 von 4)

- Auswahl von Regressionstestfällen
  - Wiederholung nur von Tests mit hoher Priorität
  - funktionalen Tests mit Verzicht auf Sonderfälle
  - Fokus auf bestimmte Konfigurationen (z.B. Sprache, OS)
  - Fokus auf von Änderungen betroffene Software
- Regressionstestsuiten oft ausgeführt mit wenig Änderungen
- hervorragende Kandidaten für Testautomatisierung
- Automatisierung dieser Tests sollte früh im Projekt beginnen

Testarten und Teststufen – Bankanwendung (1 von 4)

- Funktionaler Test
  - Komponententest: Komponente berechnet Zinseszinsen
  - Komponentenintegrationstest: Kontoinformationen auf Benutzeroberfläche erfasst und in fachliche Logik übertragen
  - Systemtest: Kontoinhaber beantragen Kreditlinie für Konto
  - Systemintegrationstest: Nutzung eines ext. Microservice für Bonitätsprüfung
  - Abnahmetest: Bankmitarbeiter entscheidet über Annahme / Ablehnung einer Kreditanfrage

Testarten und Teststufen – Bankanwendung (2 von 4)

- Nicht-funktionaler Test
  - Komponententest: Anzahl CPU-Zyklen messen für Zinsrechnung
  - Komponentenintegrationstest: Speicherüberlauf-Schwachstellen finden aufgrund von Datenübertragung von Benutzungsschnittstelle an Logikschicht
  - Systemtest: Kompatibilität der Präsentationsschicht mit Browsern und mobilen Endgeräten
  - Systemintegrationstest: Robustheit des Systems bewerten, falls der Bonitäts-Microservice nicht antwortet
  - Abnahmetest: Barrierefreiheit der Kreditbearbeitungsoberfläche bewerten

Testarten und Teststufen – Bankanwendung (3 von 4)

- White-Box-Test
  - Komponententest: vollständige Anweisungs- und Entscheidungsüberdeckung für Software zur Finanzberechnungen
  - Komponentenintegrationstest: Bildschirm der Browserbenutzungsoberfläche gibt Daten an anderen Bildschirm und Fachlogik weiter
  - Systemtest: Reihenfolgen von Webseiten während Kreditanfrage abdecken
  - Systemintegrationstest: alle möglichen Anfragearten an Microservice senden
  - Abnahmetest: Dateistrukturen für Finanzdaten und Überweisungen abdecken

Testarten und Teststufen – Bankanwendung (4 von 4)

- Änderungsbasierter Test
  - Komponententest: Automatisierte Regressionstests pro Komponente im Continuous-Integration-Framework
  - Komponentenintegrationstest: Beseitigung schnittstellenbezogener Fehlerzustände bestätigen
  - Systemtest: Tests für Workflows wiederholen, falls Änderung im Workflow
  - Systemintegrationstest: Tests der Interaktion mit dem Bonitäts-Microservice, Wiederholung als Teil der kontinuierlichen Verteilung
  - Abnahmetest: Alle zuvor fehlgeschlagenen Tests wiederholen

Kapitel 2

Testen im Software-entwicklungs-lebenszyklus

- Softwareentwicklungslebenszyklus-Modelle
- Teststufen
- Testarten
- **Wartungstest**

Test neuer Produktversionen (Wartungstest)

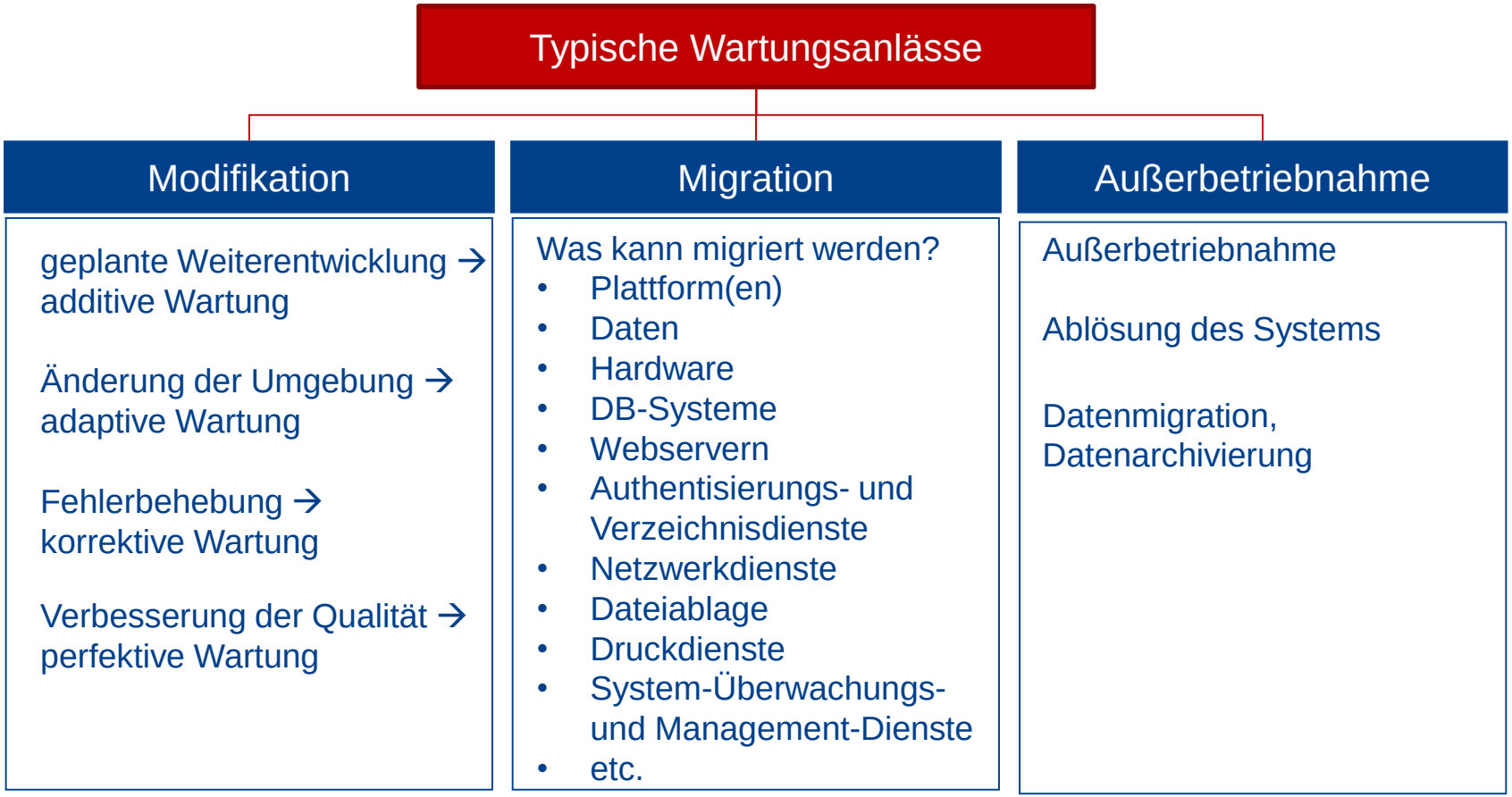
- Ende der Entwicklung nach bestandenem Abnahmetest und Auslieferung?

- Realität:
  - erstmalige Auslieferung ist erst Anfang vom Lebenszyklus
  - Nach Installation oft Jahre / Jahrzehnte im Einsatz
  - Mehrfache Änderung von Software, Umgebung oder Konfiguration in der Zeit
  - Jedes Mal neue Version, die getestet werden muss

Softwarewartung (1 von 3)

- Softwarewartung nicht regelmäßige Pflege, Software ohne Abnutzung
- Softwarewartung
  - neue Funktionalitäten hinzugefügt oder existierende gelöscht oder geändert
  - Anpassung an geänderte Einsatzbedingungen
  - Beseitigung von bereits enthaltenen Fehlerzuständen
  - Verbesserung von Qualitätsmerkmalen: Performanz, Kompatibilität, Zuverlässigkeit, Informationssicherheit und Übertragbarkeit

Softwarewartung (2 von 3)



Softwarewartung (3 von 3)

Software bedarf nach Auslieferung Korrekturen und Ergänzungen

- Wartung auf jeden Fall notwendig
- darf aber nicht als Argument für zu wenige Tests missbraucht werden
- »Wir müssen ja sowieso immer wieder neue Versionen rausbringen; also ist es nicht so schlimm, wenn wir es mit dem Testen nicht so genau nehmen und Fehler übersehen.«

Wartungsgründe (1 von 2)

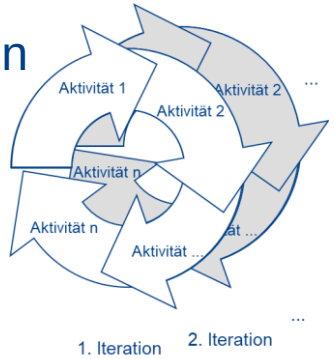
- Geplante Weiterentwicklung (additive Wartung)
  - Von Anfang an vorgesehene Änderungs- und Erweiterungsarbeiten
  - normale Produktweiterentwicklung
  - Beispiele:
    - Anpassungen durch Änderung eines Nachbarsystems
    - Implementierung einer vorgesehenen, aber bislang nicht gelieferten Funktionalität
    - Erweiterungen für eine geplante Marktausdehnung
- Softwareprojekt nicht mit Lieferung der ersten Version abgeschlossen

Wartungsgründe (2 von 2)

- Fehlerbehebung (korrektive Wartung)
  - Fehler aus dem Betrieb
  - Notfallkorrekturen („Hot Fixes“)
- Änderung der Umgebung (adaptive Wartung)
  - Aktualisierung des Betriebssystems
  - Aktualisierung des DB-Managementsystems
  - Upgrades kommerzieller Software
  - Patches externer Komponenten
  - etc.
- Verbesserung der Qualität (perfektive Wartung)
  - Verbesserung der Qualitätsfaktoren, z.B. Wartbarkeit, Performanz, Benutzbarkeit ohne Änderung des funktionalen Umfangs

Wartungstest und Testumfang

- Software-Produkte meist kontinuierlich weiterentwickelt
- Lieferungen mit Wartungsarbeiten synchronisiert für Wartungs-Updates
  - Alternative: echte funktionale Updates
- vorausschauende Release-Planung wichtig für erfolgreichen Wartungstest
  - Alternative: ungeplanten Releases / Hot Fixes
- Bei jeder Auslieferung erneuter Durchlauf aller Projektphasen
- iterativ-inkrementelle Softwareentwicklung stellt heute den Regelfall dar
- Reaktion des Testens darauf?
- Pro Release alle Tests auf allen Teststufen wiederholen?



Fehlernachtest
Auswirkungsanalyse

- Identifikation geänderter / betroffener Bereiche der Software
  - Wirkung der Änderung auf Tests zu identifizieren
  - vor der Änderung durchführen, um Auswirkung zu bewerten
- Auswirkungsanalyse evtl. schwierig, wenn:
  - Spezifikationen veraltet oder fehlen
  - Testfälle nicht dokumentiert oder veraltet
  - Bidirektionale Rückverfolgbarkeit zwischen Tests und Testbasis nicht vorhanden
  - Werkzeugunterstützung schwach oder nicht existent
  - beteiligte Personen ohne Fachkenntnis
  - während Entwicklung wenig Aufmerksamkeit auf Wartbarkeit
- Wichtig: Verfolgbarkeitsmatrix von Anforderung über Software bis zu Test
- Auswahl der Regressions-Testfälle nach Umfang der Änderung, Risiken, Kosten der Testausführung, …

Wartungstest bei Modifikation

- Wartungsrelease kann Wartungstests in mehreren Teststufen erfordern
- Umfang von Wartungstests hängt ab von:
  - Risikohöhe der Änderung (z.B. Kommunikationsintensität)
  - Größe des bestehenden Systems
  - Größe der Änderung
- Auswirkungsanalysen und Verfolgbarkeit nutzen!

Wartungstest bei Migration und Außerbetriebnahme

- Wartungstest bei Migration
  - Umfasst Tests im Betrieb der neuen Umgebung, der geänderten Software
- Wartungstest bei der Außerbetriebnahme für …
  - Datenmigration (auf neues System)
  - Archivierung (bei langer Aufbewahrungszeit)
  - Wiederherstellungsverfahren nach Archivierung bei langer Aufbewahrung
  - Regressionstests für Funktionalität, die in Betrieb bleibt
- Konvertierungstests notwendig
  - Test der Konvertierungs-Software für Daten
  - Test der konvertierten Daten

Zusammenfassung (1 von 4)

- allgemeines V-Modell
  - Teststufen: Komponententest, Integrationstest, Systemtest und Abnahmetest
  - unterscheidet zwischen verifizierender und validierender Prüfung
- Komponententest testet einzelne Softwarebausteine
- Integrationstest prüft Zusammenwirken getesteter Softwarebausteine
- Funktionaler und nicht-funktionaler Systemtest betrachten Gesamtsystem
- Abnahmetest: Auftraggeber prüft Produkt auf Benutzerakzeptanz und gegen Abnahmekriterien
- Alpha- und Beta-Tests sammeln Erfahrung mit Vorabversionen

Zusammenfassung (2 von 4)
Vergleich der Teststufen

Kriterium	Komponententest	Integrationstest	Systemtest	Abnahmetest
Testziele	Fehlerzustände in Software (-bausteinen), die separat getestet werden können, finden	Fehlerzustände in Schnittstellen und im Zusammenspiel zwischen integrierten Komponenten finden.	Prüfen, ob die spezifizierten Anforderungen (funktional, nicht-funktional) vom Produkt erfüllt werden.	Vertrauen des Auftrag-gebers bzw. der Nutzer in das System oder in bestimmte nicht-funktionale Eigenschaften gewinnen.
Testbasis	Komponentenspezifikation, detaillierter Entwurf, Datenmodell, Programmcode.	Software- und System-entwurf, Sequenzdiagramme, Spezifikation interner und externer Schnittstellen, Kommunikationsprotokolle, Anwendungsfälle, Architektur auf Komponenten oder Systemebene, Workflows.	System- und Anforderungsspez., Anwendungsfälle, funktionale Spezifikation, Geschäftsprozesse, Risikoanalyseberichte.	Benutzeranforderungen, Systemanforderungen, Anwendungsfälle, Geschäftsprozesse, Risikoanalyseberichte.
Typische Testobjekte	Isolierte Softwarebausteine (Klasse, Unit, Modul, Klasse), Komponenten, Programme, Code und Datenstrukturen, Datenumwandlungs-/ Migrationsprogramme, Datenbankmodule.	Zu integrierende Einzel-bausteine, Subsysteme und zugekaufte Standard-Komponenten, z.B. Daten-bankimplementierungen, Infrastruktur, Schnittstellen, APIs, Microservices, Systemkonfiguration und Konfigurationsdaten.	System-, Anwender- und Betriebshandbücher, Systemkonfiguration und Konfigurationsdaten.	Geschäftsprozesse des integrierten Systems, Betriebs- und Wartungs-prozesse, Anwenderverfahren, Formulare, Berichte, Konfigurationsdaten.
Test-werkzeuge	Entwicklungsumgebung, Debugging-Unterstützung Stat. Analysewerkzeuge, Komponententestumgebung.	Testmonitore zur Überwachung des Datenaustauschs zwischen Komponenten.	Testmanagement-Werkzeuge, GUI-Automatisierungs-werkzeuge.	Meist manuell durchgeführt, tw. GUI-Automatisierungs-werkzeuge.

Zusammenfassung (3 von 4)
Vergleich der Teststufen

Kriterium	Komponententest	Integrationstest	Systemtest	Abnahmetest
Test-umgebung	Platzhalter, Treiber, Simulatoren.	Wiederverwendung/Erweiterung der Platzhalter, Treiber, Simulatoren aus dem Komponententest.	Test- und Produktiv-umgebung sollten so weit wie möglich übereinstimmen.	Test- und Produktivumgebung sollten so weit wie möglich übereinstimmen.
Typische Fehler-wirkungen	Nicht korrekte Funktionalität (z.B. nicht wie in den Entwurfsspezifikationen beschrieben), Datenflussprobleme, fehlerhafter Code und fehlerhafte Logik. Alle Fehlerzustände werden in der Regel behoben, sobald sie gefunden werden. Oftmals ohne formales Fehlermanagement. Wenn Entwickler allerdings Fehlerzustände berichten, liefert dies wichtige Informationen für die Grundursachenanalyse und die Prozess-verbesserung.	Falsche Daten, fehlende Daten oder falsche Datenverschlüsselung, Schnittstellenfehlanpassung, nicht behandelte oder nicht ordnungsgemäß behandelte Fehlerwirkungen in der Kommunikation zwischen den Komponenten, falsche Annahmen über die Bedeutung, Einheiten oder Grenzen übermittelter Daten, falsche Reihenfolge oder fehlerhafte zeitliche Abfolge von Schnittstellen-aufrufen. Blockierende Fehlerzustände werden behoben, sobald sie gefunden werden. Im Systemintegrationstest mit formalem Fehlermanage-ment.	Falsche Berechnungen, falsches oder unerwartetes funktionales oder nicht-funktionales Verhalten, falsche Kontroll- und/oder Datenflüsse innerhalb des Systems, Versagen bei der korrekten oder vollstän-digen Ausführung von End-to-End-Aufgaben, Versagen des Systems bei in der Produktivumgebung, System funktioniert nicht wie in den System- oder Benutzeranleitungen beschrieben. Abnahmeverhindernde Fehlerzustände werden vor Auslieferung behoben. Mit formalem Fehler-management.	System unvollständig, System funktioniert nicht wie erwartet, funktionale und nicht-funktionale Verhaltensweisen des Systems entsprechen nicht den Spezifikationen, Vertragliche oder regulato-rische Konformität verletzt, Einzelne Anwendergruppen lehnen das System ab. Abnahmeverhindernde Fehlerzustände werden nach Absprache behoben. Mit formalem Fehler-management.

Zusammenfassung (4 von 4)

- grundlegenden Testarten
  - funktionaler und nicht-funktionaler Test
  - struktur- und änderungsorientierter Test
- Fehlerkorrekturen (Wartung) und geplante Weiterentwicklung (Pflege) erfolgt Änderung von Software während Lebenszyklus
- Test jeder geänderten Versionen notwendig!
- Umfang der Regressionstests durch Auswirkungsanalyse und Risikoabschätzung festlegen

Folgende Fragen sollten Sie jetzt beantworten können

- Erläutern Sie die einzelnen Phasen des allgemeinen V-Modells.
- Definieren Sie die Begriffe Verifizierung und Validierung.
- Begründen Sie, warum Verifizierung sinnvoll ist, auch wenn eine sorgfältige Validierung stattfindet (und umgekehrt).
- Charakterisieren Sie die typischen Testobjekte im Komponententest.
- Nennen Sie die Testziele des Integrationstests.
- Welche Integrationsstrategien lassen sich unterscheiden?
- Wie lassen sich Systemtest und Abnahmetest unterscheiden?

Folgende Fragen sollten Sie jetzt beantworten können

- Welche Gründe sprechen dafür, Tests in einer separaten Testinfrastruktur durchzuführen?
- Erläutern Sie anforderungsbasiertes Testen.
- Definieren Sie Lasttest, Performanztest, Stresstest.
  Was sind die Unterscheidungsmerkmale?
- Worin unterscheiden sich Fehlernachtest und Regressionstest?
- In welcher Projektphase nach allgemeinem V-Modell sollte das Testkonzept erstellt werden?
- Was sind die Ziele von Fehlernachtest und Regressionstest?
- Welche Testarten lassen sich unterscheiden?

Muster-Prüfungsfragen

Testen Sie Ihr Wissen…

Frage 1

9. Wie kann der White-Box-Test während des Abnahmetests angewendet werden? [K1]

a)	Um zu prüfen, ob große Datenmengen zwischen integrierten Systemen übertragen werden können.	☐
b)	Um zu prüfen, ob alle Code-Anweisungen und Code-Entscheidungspfade ausgeführt wurden.	☐
c)	Um zu prüfen, ob alle Abläufe der Arbeitsprozesse abgedeckt sind.	☐
d)	Um alle Webseiten-Navigationen abzudecken.	☐

Quelle: ISTQB Foundation Level Sample Paper; SET A; 2018; Deutschsprachige Fassung/ Lokalisierung; German Testing Board;

Frage 1 – Lösung

9. Wie kann der White-Box-Test während des Abnahmetests angewendet werden? [K1]

a)	Um zu prüfen, ob große Datenmengen zwischen integrierten Systemen übertragen werden können.	☐
b)	Um zu prüfen, ob alle Code-Anweisungen und Code-Entscheidungspfade ausgeführt wurden.	☐
c)	Um zu prüfen, ob alle Abläufe der Arbeitsprozesse abgedeckt sind.	☒
d)	Um alle Webseiten-Navigationen abzudecken.	☐

Quelle: ISTQB Foundation Level Sample Paper; SET A; 2018; Deutschsprachige Fassung/ Lokalisierung; German Testing Board;

Frage 2

10. Welche der folgenden Aussagen zum Vergleich zwischen Komponententest und Systemtest ist WAHR? [K2]

a)	Komponententests überprüfen die Funktion von Komponenten, Programmobjekten und Klassen, die separat prüfbar sind, während Systemtests die Schnittstellen zwischen den Komponenten und Wechselwirkungen mit anderen Teilen des Systems überprüfen.	☐
b)	Testfälle für den Komponententest werden in der Regel von Komponentenspezifikationen, Designspezifikationen oder Datenmodellen abgeleitet, während Testfälle für den Systemtest in der Regel von Anforderungsspezifikationen oder Anwendungsfällen abgeleitet werden.	☐
c)	Komponententests konzentrieren sich nur auf die funktionalen Eigenschaften, während Systemtests sich auf die funktionalen und nicht-funktionalen Eigenschaften konzentrieren.	☐
d)	Komponententests sind in der Verantwortung der Tester, während die Systemtests in der Regel in der Verantwortung der Benutzer des Systems liegen.	☐

Quelle: ISTQB Foundation Level Sample Paper; SET A; 2018; Deutschsprachige Fassung/ Lokalisierung; German Testing Board;

Frage 2 – Lösung

10. Welche der folgenden Aussagen zum Vergleich zwischen Komponententest und Systemtest ist WAHR? [K2]

a)	Komponententests überprüfen die Funktion von Komponenten, Programmobjekten und Klassen, die separat prüfbar sind, während Systemtests die Schnittstellen zwischen den Komponenten und Wechselwirkungen mit anderen Teilen des Systems überprüfen.	☐
b)	Testfälle für den Komponententest werden in der Regel von Komponentenspezifikationen, Designspezifikationen oder Datenmodellen abgeleitet, während Testfälle für den Systemtest in der Regel von Anforderungsspezifikationen oder Anwendungsfällen abgeleitet werden.	☒
c)	Komponententests konzentrieren sich nur auf die funktionalen Eigenschaften, während Systemtests sich auf die funktionalen und nicht-funktionalen Eigenschaften konzentrieren.	☐
d)	Komponententests sind in der Verantwortung der Tester, während die Systemtests in der Regel in der Verantwortung der Benutzer des Systems liegen.	☐

Quelle: ISTQB Foundation Level Sample Paper; SET A; 2018; Deutschsprachige Fassung/ Lokalisierung; German Testing Board;

Frage 3

11. Welche der folgenden Aussagen ist zutreffend? [K2]

a)	Ziel des Regressionstests ist es, zu überprüfen, ob die Korrektur erfolgreich implementiert wurde, während der Zweck der Fehlernachtests darin besteht, zu bestätigen, dass die Korrektur keine Seiteneffekte hat.	☐
b)	Der Zweck des Regressionstests ist es, unbeabsichtigte Seiteneffekte zu erkennen, während der Zweck des Fehlernachtests darin besteht zu prüfen, ob das System in einer neuen Umgebung noch funktioniert.	☐
c)	Der Zweck des Regressionstests ist es, unbeabsichtigte Seiteneffekte zu erkennen, während der Zweck des Fehlernachtests darin besteht zu prüfen, ob der ursprüngliche Fehlerzustand behoben wurde.	☐
d)	Der Zweck des Regressionstests ist es zu prüfen, ob die neue Funktionalität funktioniert, während der Zweck des Fehlernachtests darin besteht zu prüfen, ob der ursprüngliche Fehlerzustand behoben wurde.	☐

Quelle: ISTQB Foundation Level Sample Paper; SET A; 2018; Deutschsprachige Fassung/ Lokalisierung; German Testing Board;

Frage 3 – Lösung

11. Welche der folgenden Aussagen ist zutreffend? [K2]

a)	Ziel des Regressionstests ist es, zu überprüfen, ob die Korrektur erfolgreich implementiert wurde, während der Zweck der Fehlernachtests darin besteht, zu bestätigen, dass die Korrektur keine Seiteneffekte hat.	☐
b)	Der Zweck des Regressionstests ist es, unbeabsichtigte Seiteneffekte zu erkennen, während der Zweck des Fehlernachtests darin besteht zu prüfen, ob das System in einer neuen Umgebung noch funktioniert.	☐
c)	Der Zweck des Regressionstests ist es, unbeabsichtigte Seiteneffekte zu erkennen, während der Zweck des Fehlernachtests darin besteht zu prüfen, ob der ursprüngliche Fehlerzustand behoben wurde.	☒
d)	Der Zweck des Regressionstests ist es zu prüfen, ob die neue Funktionalität funktioniert, während der Zweck des Fehlernachtests darin besteht zu prüfen, ob der ursprüngliche Fehlerzustand behoben wurde.	☐

Quelle: ISTQB Foundation Level Sample Paper; SET A; 2018; Deutschsprachige Fassung/ Lokalisierung; German Testing Board;

Frage 4

12. Welches ist die BESTE Definition eines inkrementellen Entwicklungsmodells? [K2]

a)	Die Definition der Anforderungen, das Design der Software und das Testen erfolgen in einer Serie durch Hinzufügen von Teilen.	☐
b)	Eine Phase des Entwicklungsprozesses sollte beginnen, wenn die vorhergehende Phase abgeschlossen ist.	☐
c)	Das Testen wird als separate Phase betrachtet. Sie startet, wenn die Entwicklung abgeschlossen ist.	☐
d)	Das Testen wird der Entwicklung als Inkrement hinzugefügt.	☐

Quelle: ISTQB Foundation Level Sample Paper; SET A; 2018; Deutschsprachige Fassung/ Lokalisierung; German Testing Board;

Frage 4 – Lösung

12. Welches ist die BESTE Definition eines inkrementellen Entwicklungsmodells? [K2]

a)	Die Definition der Anforderungen, das Design der Software und das Testen erfolgen in einer Serie durch Hinzufügen von Teilen.	☒
b)	Eine Phase des Entwicklungsprozesses sollte beginnen, wenn die vorhergehende Phase abgeschlossen ist.	☐
c)	Das Testen wird als separate Phase betrachtet. Sie startet, wenn die Entwicklung abgeschlossen ist.	☐
d)	Das Testen wird der Entwicklung als Inkrement hinzugefügt.	☐

Quelle: ISTQB Foundation Level Sample Paper; SET A; 2018; Deutschsprachige Fassung/ Lokalisierung; German Testing Board;

Frage 5

13. Welcher der folgenden Aussagen sollte KEIN Auslöser für Wartungstests sein? [K2]

a)	Die Entscheidung, die Wartbarkeit der Software zu testen	☐
b)	Die Entscheidung, das System nach der Migration auf einer neuen Betriebsplattform zu testen	☐
c)	Die Entscheidung zu testen, ob archivierte Daten abgerufen werden können	☐
d)	Die Entscheidung zum Testen nach "Hotfixes"	☐

Quelle: ISTQB Foundation Level Sample Paper; SET A; 2018; Deutschsprachige Fassung/ Lokalisierung; German Testing Board;

Frage 5 – Lösung

13. Welcher der folgenden Aussagen sollte KEIN Auslöser für Wartungstests sein? [K2]

a)	Die Entscheidung, die Wartbarkeit der Software zu testen	☒
b)	Die Entscheidung, das System nach der Migration auf einer neuen Betriebsplattform zu testen	☐
c)	Die Entscheidung zu testen, ob archivierte Daten abgerufen werden können	☐
d)	Die Entscheidung zum Testen nach "Hotfixes"	☐

Quelle: ISTQB Foundation Level Sample Paper; SET A; 2018; Deutschsprachige Fassung/ Lokalisierung; German Testing Board;