Software Engineering und Projektmanagement 2.0 VO

EINHEIT 7

Integration & Testen (Teil 2)
Inbetriebnahme, Wartung

WS2016/17 – 01.Dezember 2016

Dr. Alexander Zapletal
www.inso.tuwien.ac.at

INSO - Industrial Software
Institut für Rechnergestützte Automation | Fakultät für Informatik | Technische Universität Wien

Inhalt

Ziele und Methodik des Softwaretestens

- Ziele:
  - Verifikation: Prüfen des Systems gegen Spezifikation
    - „Erstellen wir das Produkt richtig“
  - Validierung: Prüfen des Systems gegen (Kunden-)Anforderungen
    - „Erstellen wir das richtige Produkt“

- Methodik:
  - Testfälle identifizieren, mit denen die höchste Wahrscheinlichkeit gegeben ist, festzustellen, ob das Softwaresystem korrekt funktioniert

  - Möglichst hohe funktionale oder nichtfunktionale Abdeckung

Test- und Integrationsstufen

- Strategie (*Divide and Conquer*):
  - Zerlegung der Aufgabe in beherrschbare Teile
  - Festlegung überprüfbarer Übergabepunkte zwischen den Teilen

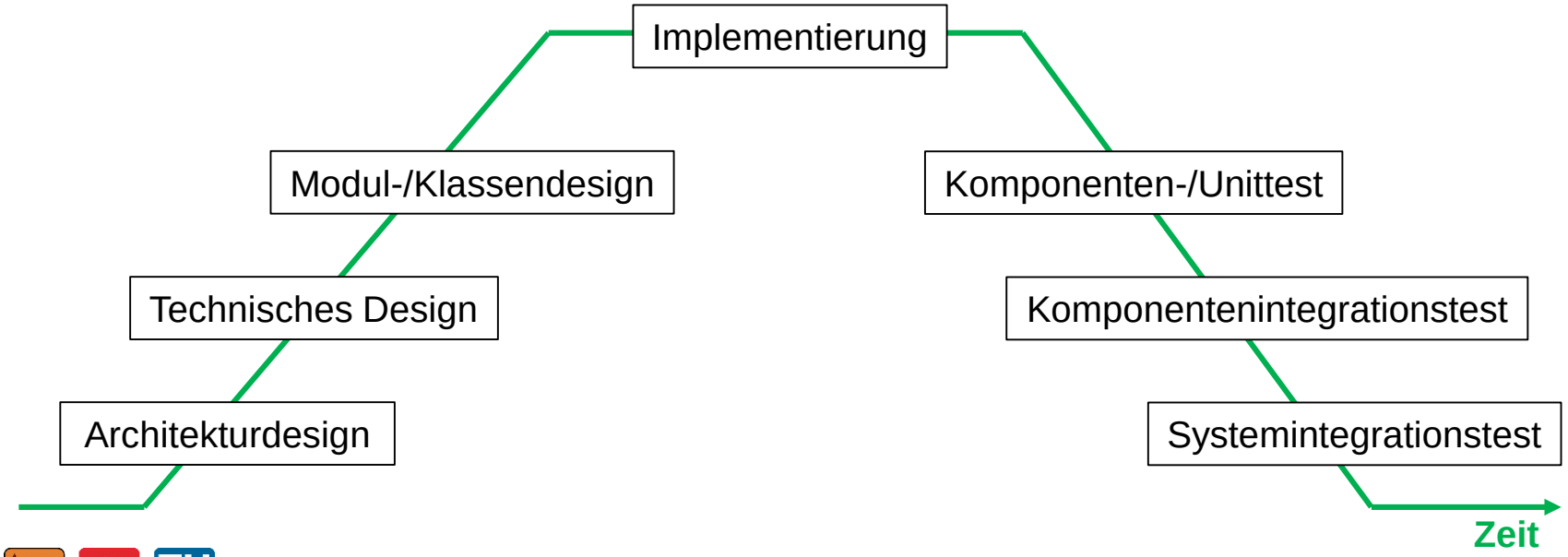
- Im Testkontext:
  - Projekt ist in Phasen eingeteilt.
  - Jeder Projektphase ist eine Teststufe zugeordnet.
  - Teststufe wird nach der Projektphase konzipiert.

  - Definition von konkreten, testbaren Kriterien! (funktional und nichtfunktional)

Test- und Integrationsstufen (2)

- Beispiel:

Projektphase	Teststufe
Architekturdesign	Systemintegrationstest
Technisches Design	Komponentenintegrationstest
Modul-/Klassendesign	Komponenten-/Unittest

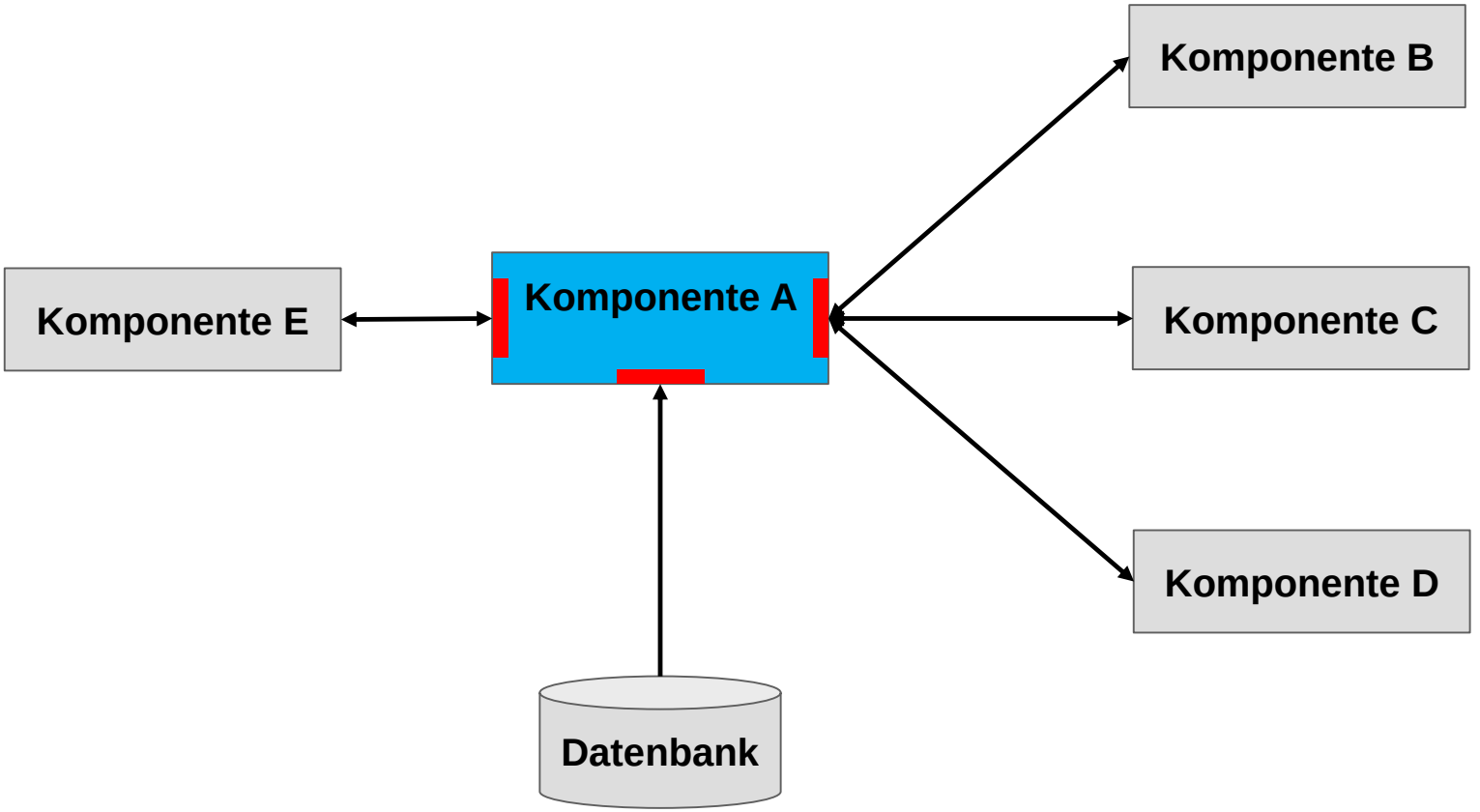


Komponententest

- Testen von funktionalen und nicht funktionalen Aspekten einer Komponente (Modul, Klasse, …), z.B.:
  - Umgang mit Ressourcen
  - Robustheit
  - Strukturelle Prüfung (z.B. Zweigüberdeckung)

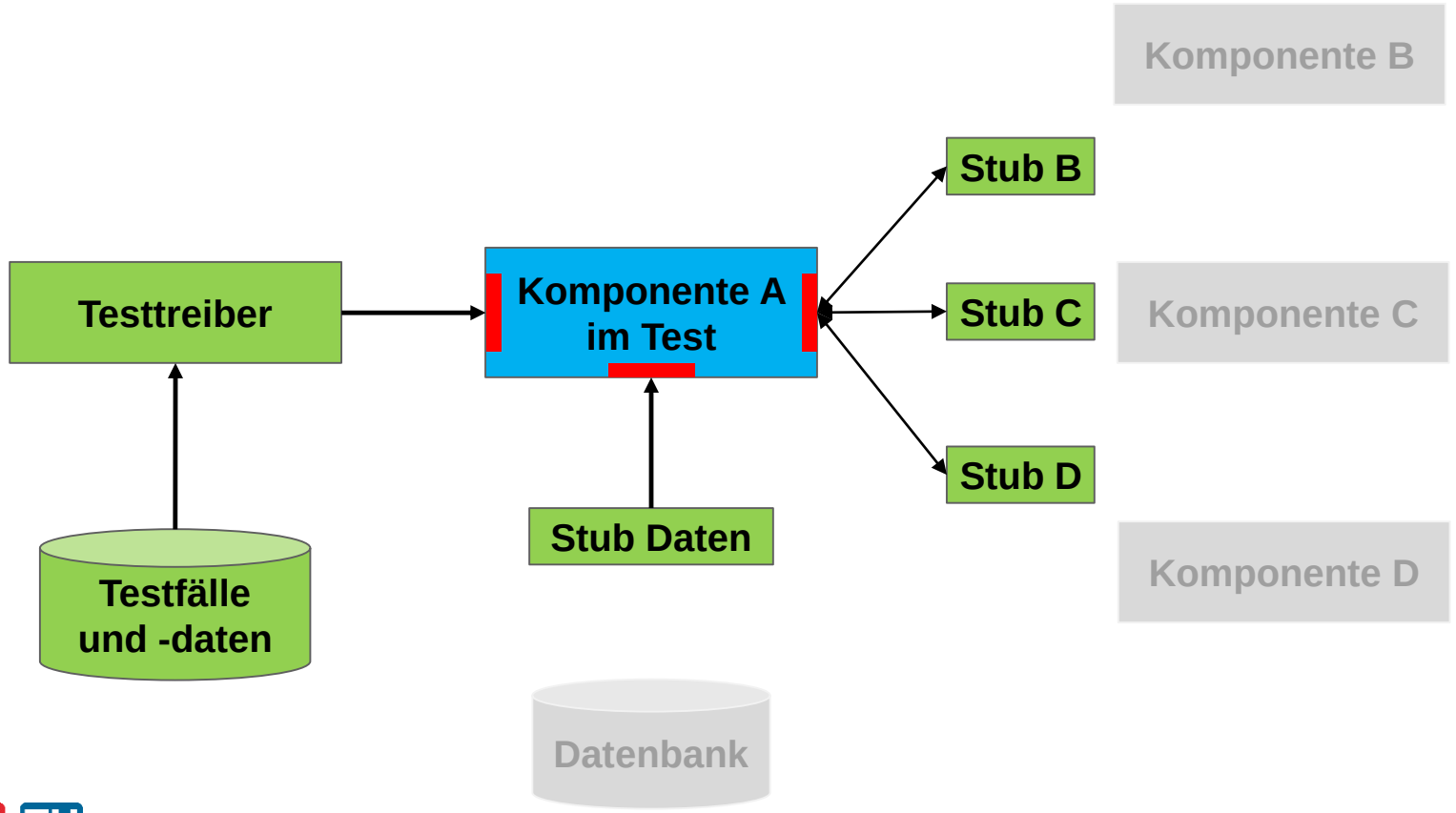
- Grundlage für Komponententests:
  - Dokumentation
  - Datenmodell
  - Quellcode
  - Kooperation mit Entwicklern

Komponententest: Ausgangssituation



Komponententest: Test-Setup

- Isolation der Komponenten vom Rest des Systems
- Erst saubere Schnittstellen ermöglichen (isoliertes) Testen!



Integrationstest

- Prüft die Schnittstellen und Interaktion zwischen Komponenten
- Verschiedene Strategien möglich:
  - Big-Bang-Integration
  - Horizontale Integration: Integration Schicht für Schicht
    - Top-Down-Integration
    - Bottom-Up-Integration
  - Vertikale Integration: Integrationsschritt pro Funktionalität/Use-Case

Integrationstest: Big-Bang-Integration

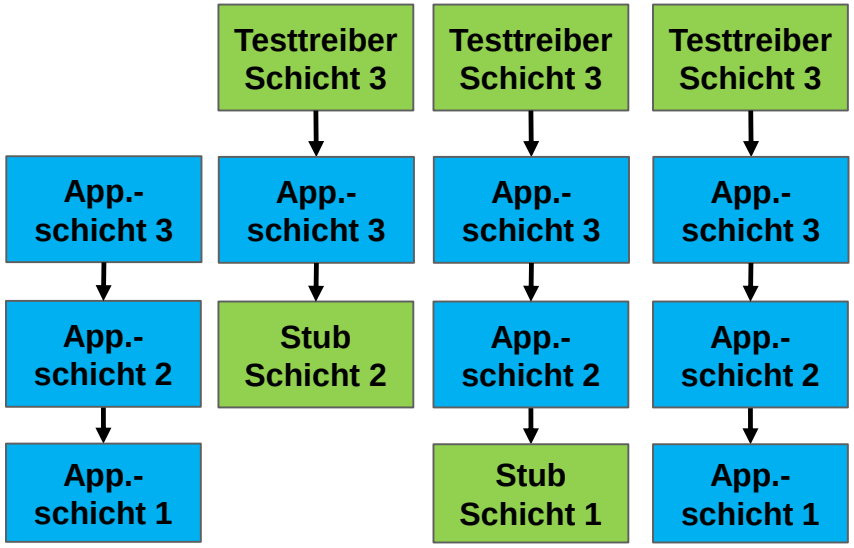
- Nicht-iterative Integrationsform
- Gleichzeitige Kombination aller Komponenten zu einem Gesamtsystem

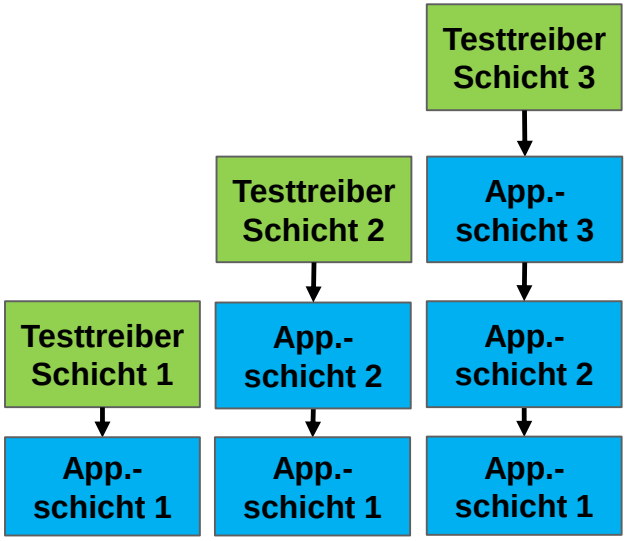
- Vorteile:
  - Keine Stubs
  - Keine Testtreiber

- Nachteile:
  - Erst spät im Projekt möglich (Risiko)
  - Fehleridentifizierung und –lokalisierung schwierig

Integrationstest: Horizontale Integration

- Iterative, schichtorientierte Integration
- Top-Down
  - Vorteil: Externe Schnittstellen früh verfügbar
  - Nachteil: Hoher Simulationsaufwand (Stubs)
- Bottom-Up
  - Vorteil: keine Stubs notwendig (dafür mehr Testtreiber, geringerer Aufwand)
  - Nachteil: Externe Schnittstellen spät verfügbar





Integrationstest: Vertikale Integration

- Iterative, funktionsorientiert Integrationsform
- Integration von Komponenten erfolgt
  - anhand von Szenarien/Use-Cases
  - mit den Komponenten, die für die zu testende Funktionalität notwendig sind
  - über die Systemschichten hinweg
- Vorteile
  - Funktionalität für Szenario/Use-Case früh verfügbar/abnehmbar
  - Nach jeder Iteration: lauffähiges, testbares Teilsystem (vgl. Scrum)
- Nachteil
  - Alle Schichten müssen Funktionalität unterstützen

Systemtest, Akzeptanztest (Abnahmetest)

- Systemtest
  - Testet das spezifizierte Verhalten des Gesamtsystems
  - Testumgebung äquivalent zu Produktivumgebung
  - Abdeckung von funktionalen und nichtfunktionalen Anforderungen

- Akzeptanztest
  - Meist von Auftraggeber durchgeführt
  - Prüft, ob Auftragnehmer die vereinbarten Leistungen umgesetzt hat

  - Verschiedene Arten:
    - Anwender-AT: Prüft Benutzbarkeit für Anwender
    - Betrieblicher AT: Backup-, Restore-Mögl., Benutzermanagement, …
    - Regulatorischer/Vertraglicher AT: Gesetzes-, Standardkonformität
    - Feldtest

Funktionale Softwaretests

Methodik
Strukturelle Abdeckung (White-Box-Test)
Funktionale Abdeckung (Black-Box-Test)

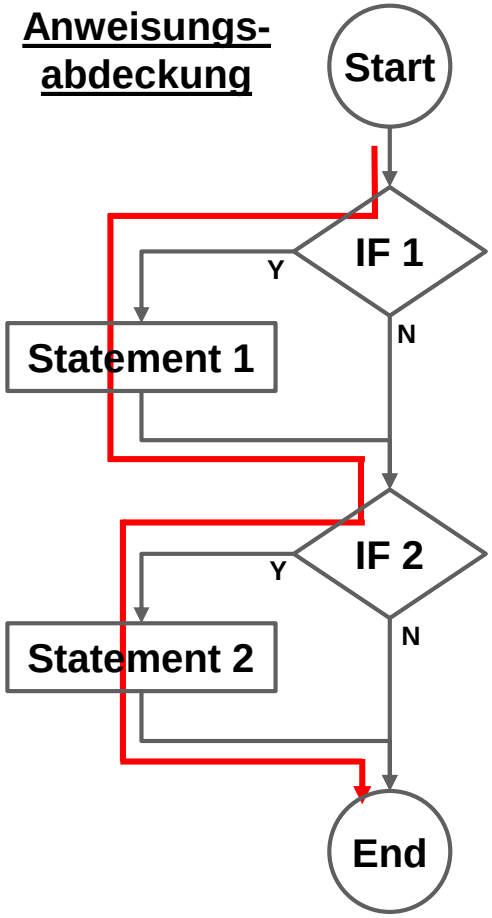
Funktionale Softwaretests: Wie testet man die Funktion einer Software?

- Ziel: Funktionale Verifikation des Systems
- Methodik:
  - hohe Abdeckung (*Coverage*) des Systems,
  - die mit hoher Wahrscheinlichkeit Fehler findet,
  - mittels ausgewählten Testfällen

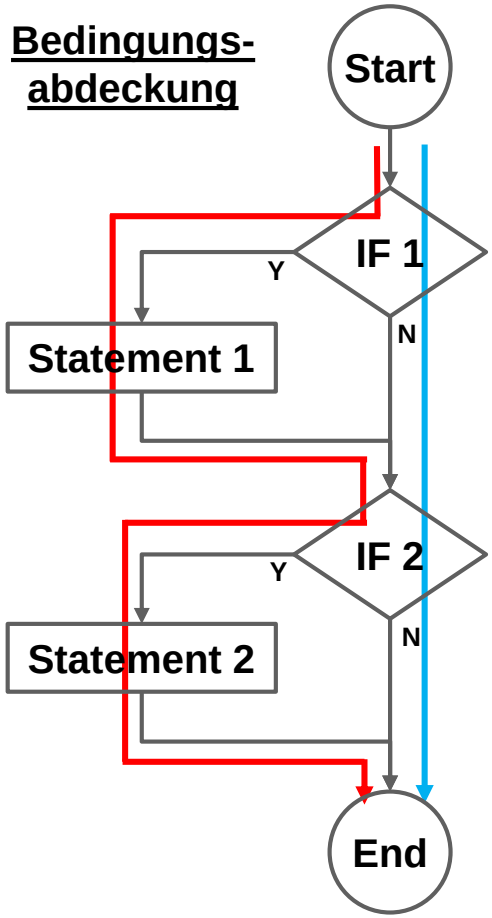
Zwei Ansätze:

- Strukturelle Abdeckung (White-Box-Tests)
  - Testfälle aus innerer Struktur des Systems ableiten
  - Analyse des Kontrollflussgraphs (Quellcode)
- Funktionale Abdeckung (Black-Box-Tests)
  - Testfälle aus Spezifikation ableiten

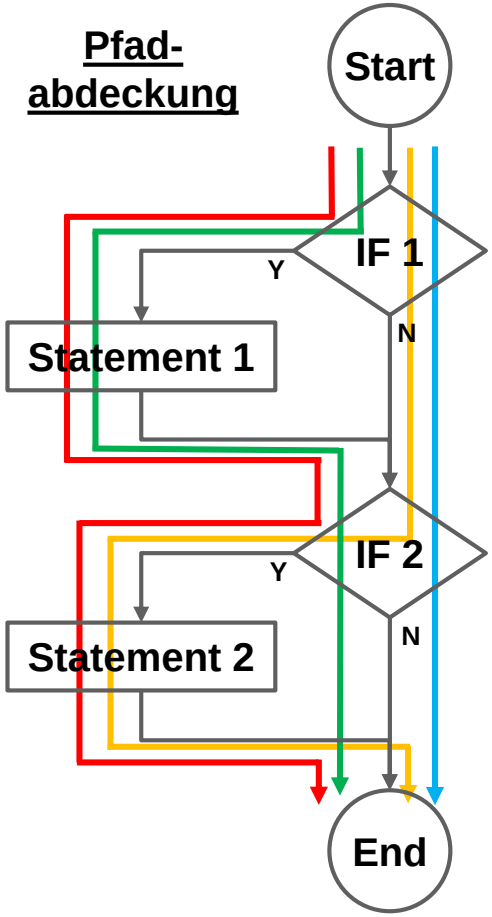
Strukturelle Abdeckung (White-Box-Test)



1 Testfall



2 Testfälle



4 Testfälle

Funktionale Abdeckung (Black-Box-Test)

- Äquivalenzklassenanalyse
  - Einteilung der möglichen Eingabewerte in Äquivalenzklassen mit gleichem erwartbaren Systemverhalten.
  - Tests durchführen für einen Repräsentanten pro Klasse
  - Grenzwertanalyse
    - Spezialfall von Äquivalenzklassenanalyse
    - Fehler treten häufig an den Grenzen der Äquivalenzklassen auf

Beispiel: Äquivalenzklassen-/Grenzwertanalyse

Beispiel: Textfeld für die Eingabe eines Centbetrags kleiner 100 Cent

Äquivalenzklassen:

- Betrag kleiner 0 Cent
- Betrag von 0 bis 99 Cent
- Betrag größer 99 Cent

Testfälle: -1, 0, 1, 99, 100

Funktionale Abdeckung (Black-Box-Test) (2)

- Äquivalenzklassenanalyse
  - Grenzwertanalyse
- Zustandsbasierte Testmethoden
  - Ableiten der Testfälle von Zustandsautomat des Systems
- Klassifikationsbaummethode
  - Einteilung der Testfälle in Klassifikationen (testrelevante Aspekte)
  - Jede Klassifikation in Äquivalenzklassen zerlegen
  - Testfall ist Kombination aus Klassen, eine Klasse pro Klassifikation.
- Informelle Testmethoden
  - Keine oder nur teilweise systematische Ableitung von Testfällen

Nichtfunktionale Softwaretests

- Testen der nichtfunktionalen Eigenschaften eines Systems
- Test abhängig von untersuchten Eigenschaft
- Mehr Expertenwissen notwendig

- Beispiele:
  - Benutzbarkeit (Usability)
  - Leistungsfähigkeit (Performance)
  - Sicherheit (Security)

Testautomatisierung

- Automatisches Ausführen von Softwaretests
- Initialaufwand wesentlich höher als bei manuellen Tests
- Testdurchführung deutlich kürzer als bei manuellen Tests
- Hoher Wartungsaufwand!

Testautomatisierung (2)

- Automatisierter Komponententest
  - Tests in Programmiersprache des Zielsystems
  - Gute Framework-Unterstützung (z.B. JUnit, TestNG)

- Automatisierter GUI-Test
  - Simulation des Benutzers
  - Capture/Replay-Verfahren
  - Nachteile: Wartung hoch, Testobjekt muss bereits verfügbar sein
  - Alternative: Scripting, Hybrider Ansatz

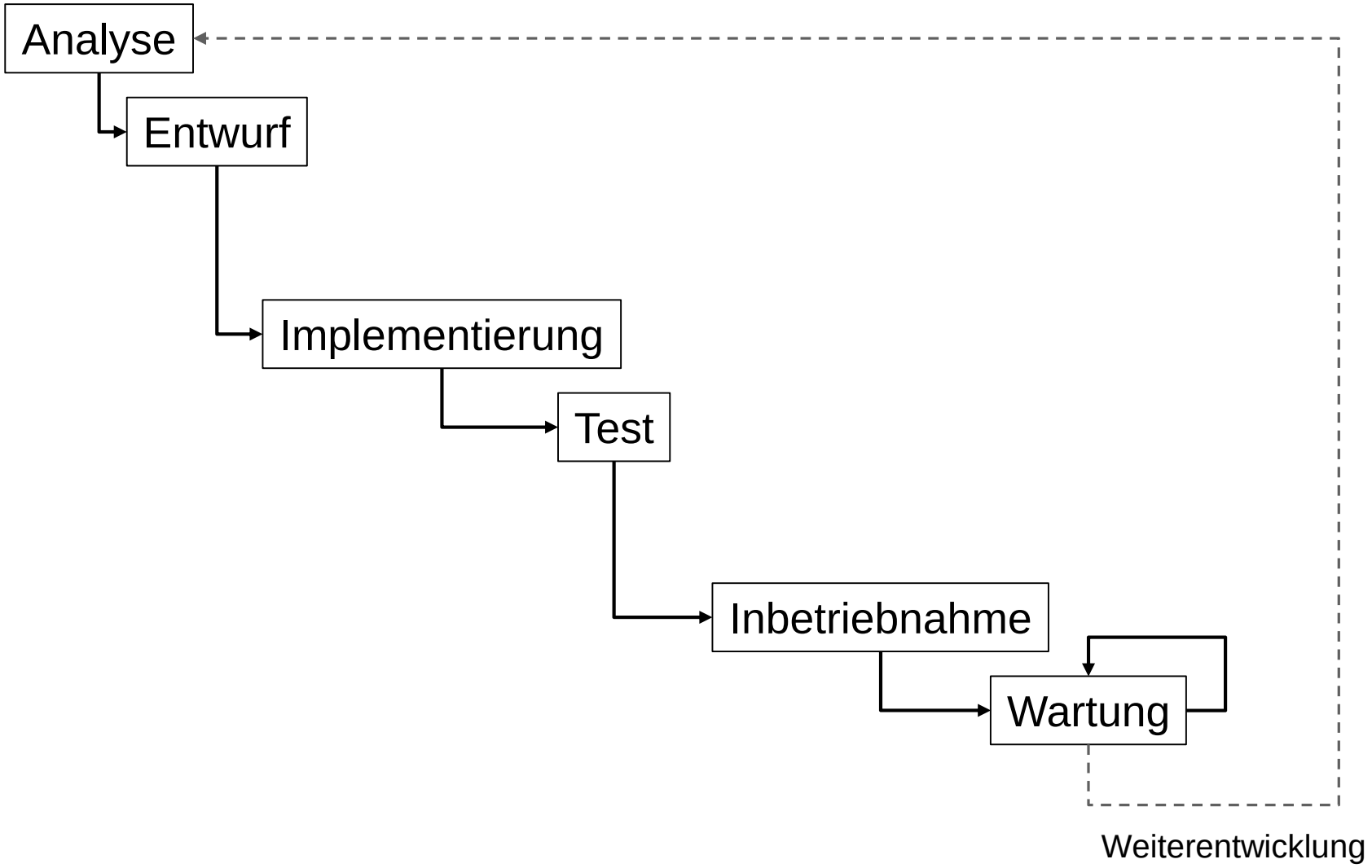
Testautomatisierung (3)

- Regressionstest
  - Ist Systemfunktionalität nach Änderungen noch gewährleistet?
  - Z.B. Änderungen an Quellcode oder Konfiguration, Tausch einer Komponente, …

  - Kontinuierliche Integration (Continuous Integration)
    - Automatisches Bauen und Testen des Systems nach jeder Änderung

Was kommt nach dem Implementieren und dem Testen?

Software Life Cycle



Inbetriebnahme & Wartung

- Worauf muss man bei der Einführung der erstellten Software beim Kunden achten?
- Wie wird die Software erfolgreich beim Kunden in Betrieb genommen?
- Welche Anforderungen ergeben sich beim Betrieb einer Applikation?
- Wie kann eine Software gewartet werden, um sie möglichst lange am Leben zu erhalten

Übersicht

- Inbetriebnahme
  - Anwenderakzeptanz (Integration der Systemanwender)
  - Migration
  - Inbetriebsetzung und laufender Betrieb

- Wartung
  - Auslöser von Wartungstätigkeiten
  - Wartungstypen
  - Ausgewählte How-Tos

Anwenderakzeptanz , Integration der Systemanwender

Wird erreicht durch:

- Anwendergerechte Dokumentation
  - Benutzerhandbuch
  - Online-Hilfe
- Schulungen
  - Meist mehrere Benutzergruppen (unterschiedliche Funktionalität/Berechtigungen)
- Rollout
  - „Big Bang“-Rollout ist zu vermeiden
  - Testbetrieb mit kleinen Benutzergruppe
  - Parallelbetrieb

Migration von bestehenden Systemteilen/Daten

- Ablöse eines bestehenden Systems durch ein neues System
- Laufenden Betrieb nicht beeinflussen (unterbrechungsfrei)

- Softwaremigration
- Datenmigration
  - Transfer von Daten in ein neues System
- Hardwaremigration
- Kombination, insb. Softwaremigration + Datenmigration

Inbetriebsetzung

Planung!

- Eigene Maschine(n) ?
- Datenbank
- Integration von Fremdsystemen
- Verantwortlichkeiten klären
  - Installation der Software
  - Einrichten der Datenbank
  - Benutzerkonten, Netzwerkfreigaben
- Dokumentation
  - Release-Checkliste
  - Release-Drehbuch

Inbetriebnahme (2)

- Best Practices
  - Zusammenarbeit zwischen Entwicklern und Betrieb
  - Verantwortlichkeiten festlegen
  - Frühzeitiges Testrollout

- Anti-Patterns
  - Manuelle Installation der Software
  - Installation erst nach Abschluss der Entwicklung
  - Manuelle Konfiguration
  - Änderungen direkt in Betriebsumgebung („Hot Fixes“)

Definition Softwarewartung

- Sämtliche Änderungen eines Software zur
  - Fehlerbehebung,
  - Verbesserung gewisser Eigenschaften der Software (z.B. Performance),
  - Anpassung an eine geänderte Umgebung

- Änderungen erfolgen nach der Lieferung
- Änderungen verändern Funktionalität nicht

Auslöser von Wartungstätigkeiten

- Gefundener Fehler
- Performance-Verbesserungen
- Updaten von verwendeten Frameworks und Bibliotheken
- Behebung von Sicherheitslücken
- Optimierung der Konfiguration

Wartungstypen

- Korrektive Wartung
  - Ausbessern von aufgetretenem Fehler
- Präventive Wartung
  - Ausbessern von Fehler, bevor dieser auftritt
- Adaptive Wartung
  - Anpassung an eine geänderte Umgebung
  - Investitionsschutz
- Perfektionierende Wartung
  - Verbesserung des Systems, z.B. Performance, Usability, Speicherverbrauch, …

Softwarewartung: How-To

Einige typische Maßnahmen während der Wartung:

- Reengineering
  - Neuentwicklung (eines Systemteils) bei gleichbleibender Funktionalität
  - Ziel: Qualitätssteigerung, Vorbereitung für Weiterentwicklung
- Refaktorisierung
  - Umstrukturierung bei gleichbleibender Funktionalität
  - Ziel: Erhaltung/Erhöhung der Wartbarkeit
- Eliminieren von Anti-Patterns
  - Erkennen und Entfernen von schlechten Softwareteilen

Zusammenfassung

- Implementierung
  - Know Your Tools
  - Frameworks immer wichtiger für Softwareentwicklung
  - Saubere Schnittstellen, geringe Kopplung, hohe Kohäsion

- Testen
  - Strategische Planung notwendig
  - Vollständiger Test nicht möglich, Auswahl der Testfälle
  - Frühes Testen/Integration reduziert Risiko
  - Testen zeigt nicht Fehlerfreiheit, sondern reduziert Fehlerwahrscheinlichkeit

Zusammenfassung (2)

Softwareprojekt ist nach Implementierungs- und Testphase nicht abgeschlossen.

- Inbetriebnahme
  - Ausführliche Planung erforderlich
  - Endanwender müssen vorbereitet/unterstützt werden
  - Migration

- Wartung
  - Wartung hat größten Anteil an Software-Life-Cycle
  - Meist mehrere Wartungszyklen