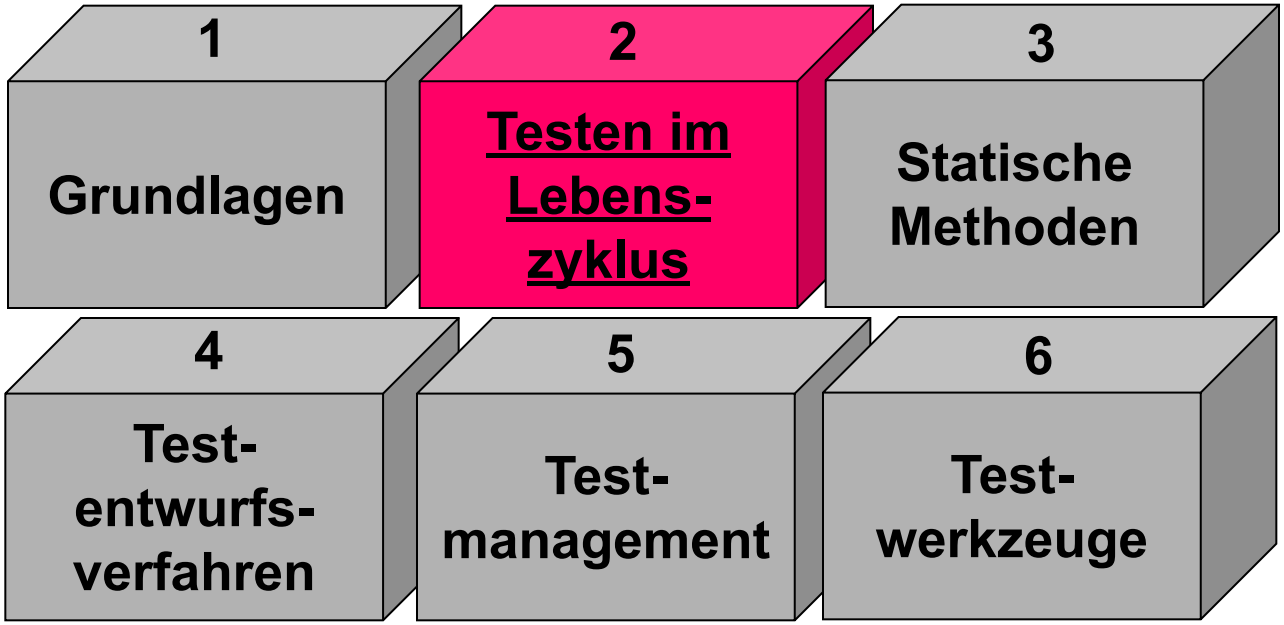
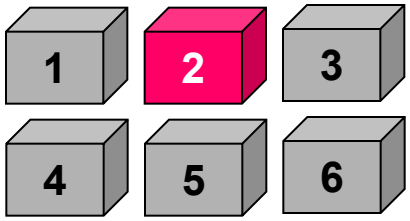




Testen im Lebenszyklus

Lernzielstufe

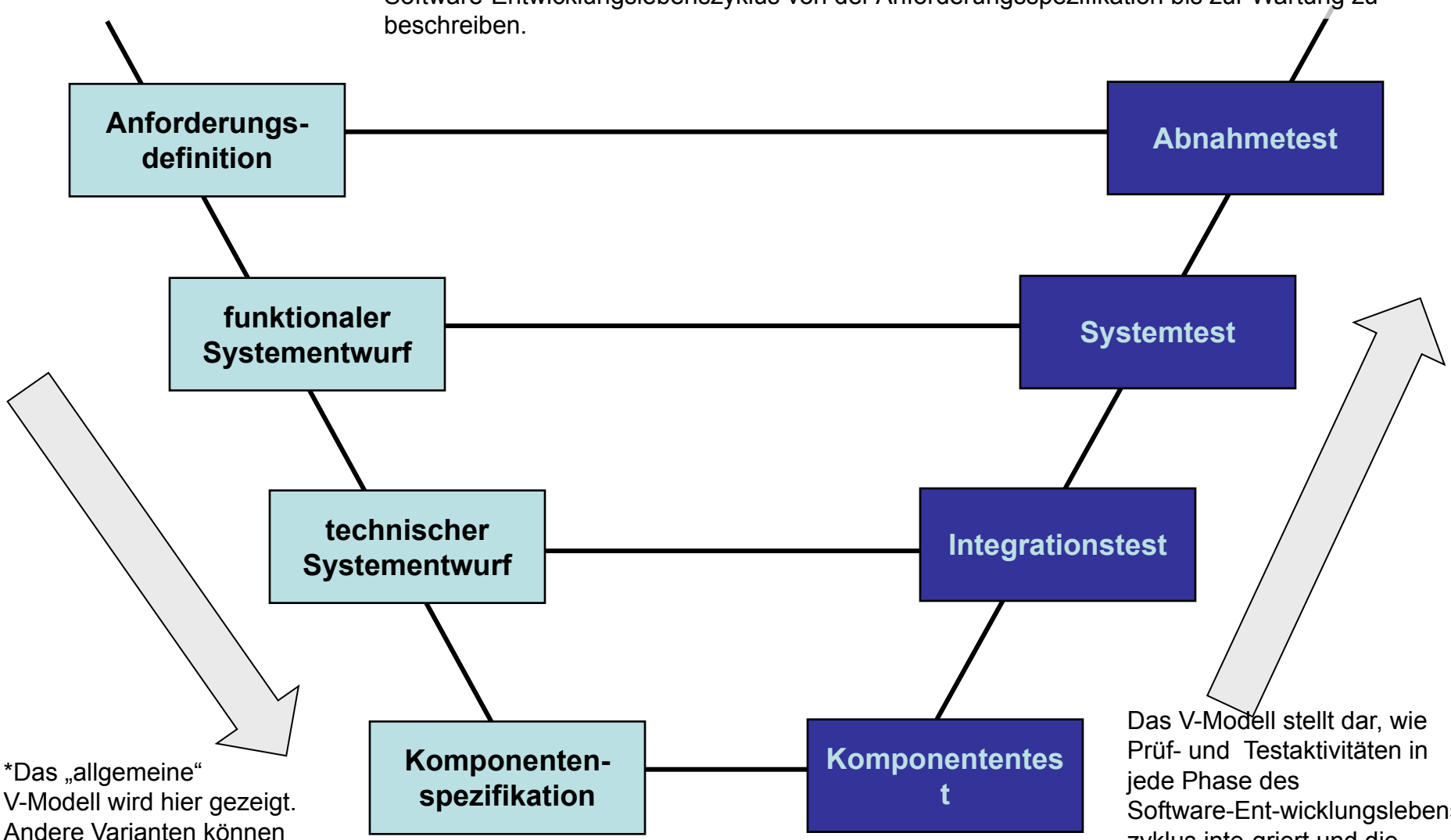
- K2 — Softwareentwicklungsmodelle
- K2 — Teststufen
- K2 — Testarten
- K2 — Wartungstest

Sehe Lehrplan, Kapitel 2 für die Lernzielbeschreibung dieses Moduls
Siehe Modul 1, Anhang A, Beschreibung der Lernzielstufen

Das V-Modell*

G

Vorgehensmodell für die Softwareentwicklung, um die Aktivitäten des Software-Entwicklungslebenszyklus von der Anforderungsspezifikation bis zur Wartung zu beschreiben.

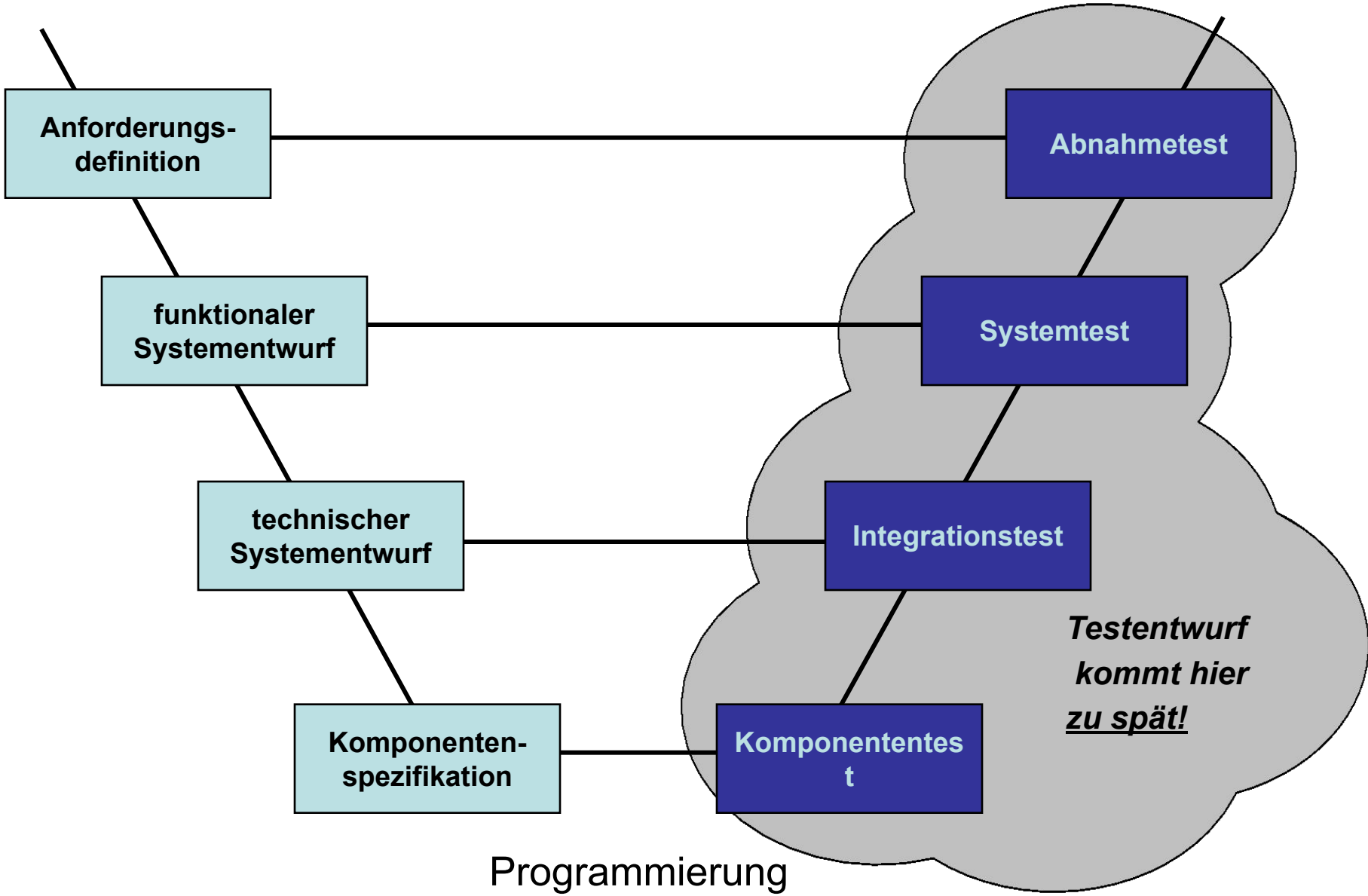


*Das „allgemeine“ V-Modell wird hier gezeigt. Andere Varianten können weniger, mehr oder andere Stufen haben.

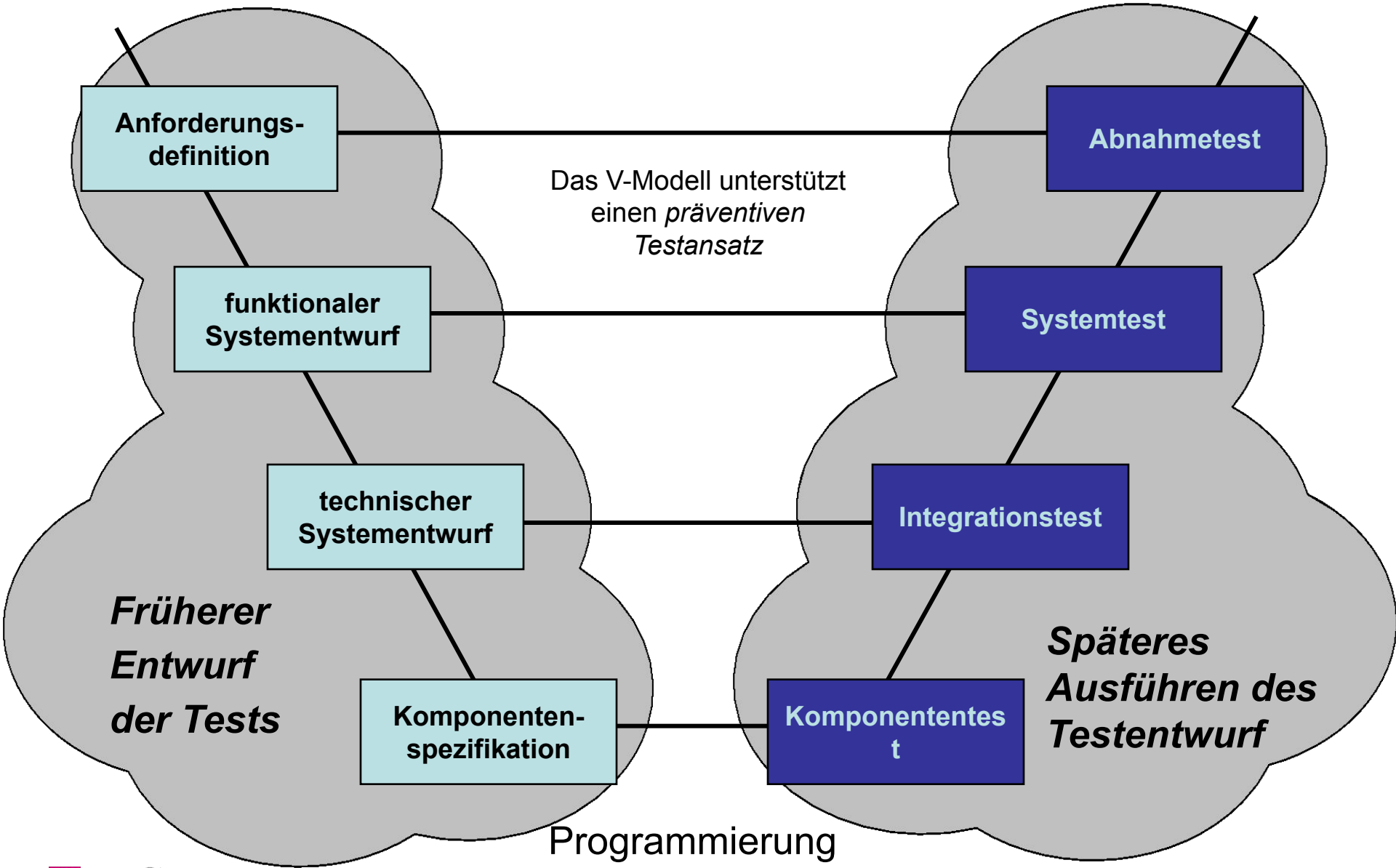
G

Das V-Modell stellt dar, wie Prüf- und Testaktivitäten in jede Phase des Software-Ent-wicklungslebens zyklus inte-griert und die Zwischen-produkte geprüft (validiert und verifiziert) werden können.

Das V-Modell: Früher Testentwurf



Das V-Modell: *Präventiver Testansatz*



Relevante Entwicklungsdokumente

- Geschäftsvorfälle (Geschäftsabläufe oder Geschäftsprozesse)
- Anforderungsdefinition
- Funktionaler und technischer Systementwurf
- Komponentenspezifikation

Testbasis

- Mögliche Quellen für generische Dokumente:
  - CMMI (Capability Maturity Model Integration)*
  - IEEE 12207 - Software life cycle processes*
  - UP (Unified Process) [www.rational.com]

* Quellen sind im Lehrplan, Kapitel 7 beschrieben

Validierung und Verifizierung

Validierung

G Bestätigung durch Bereitstellung eines objektiven Nachweises, dass die Anforderungen für einen spezifischen beabsichtigten Gebrauch oder eine spezifische beabsichtigte Anwendung erfüllt worden sind [ISO 9000].

g Validierung bestätigt, dass das Produkt, so wie es vorliegt, seinen beabsichtigten Verwendungszweck erfüllen kann. Mit anderen Worten, Validierung stellt sicher, dass „du das richtige Ding erzeugst“ [CMMI-SW, V1.1]

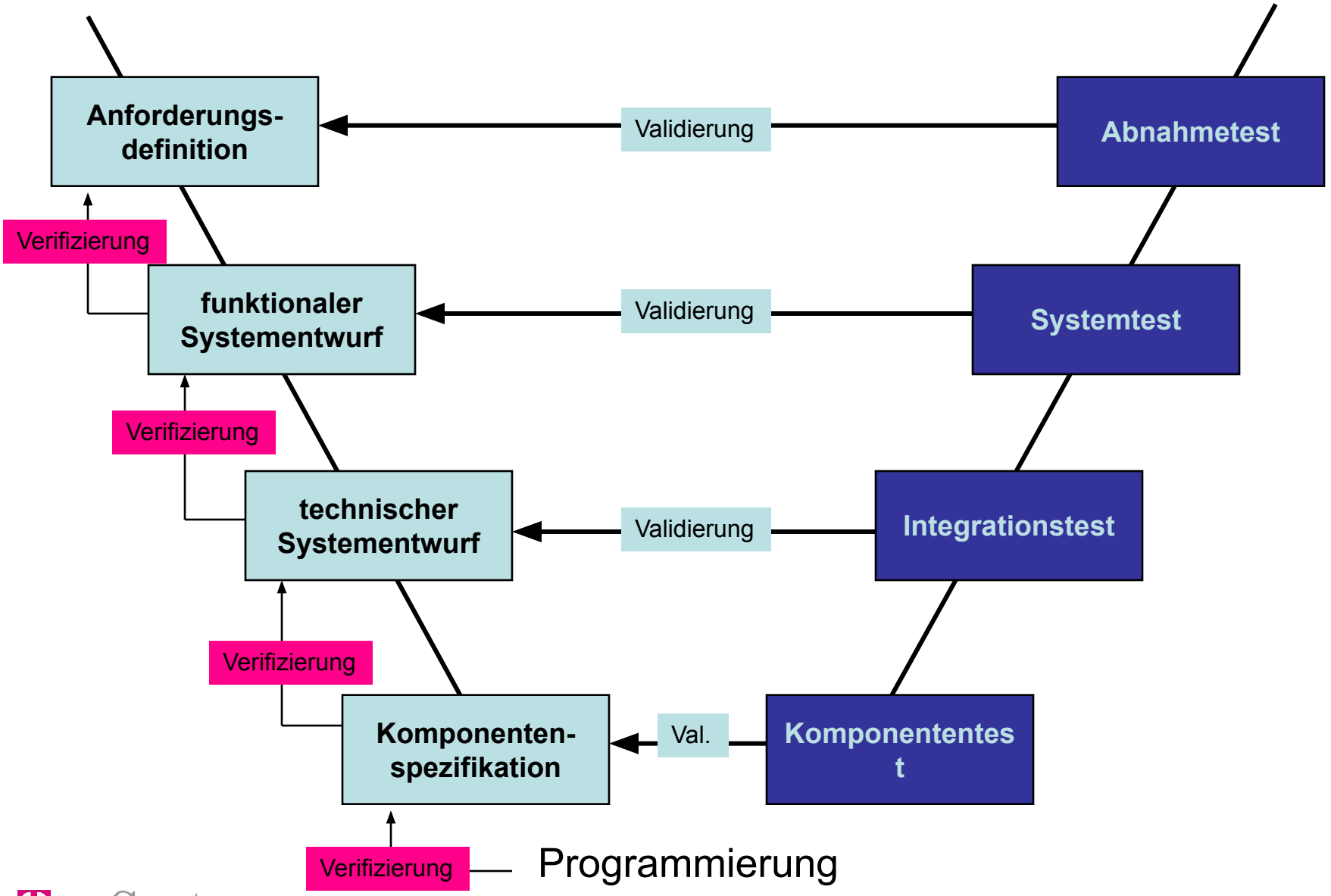
Verifizierung

G *Verifizierung:* Bestätigung durch Bereitstellung eines objektiven Nachweises, dass festgelegte Anforderungen erfüllt worden sind [ISO 9000]

g Prüfung, ob die Ergebnisse einer Entwicklungsphase die Vorgaben der Phaseneingangsdokumente erfüllen [nach IEEE 610]

g Verifikation bestätigt, dass das in Arbeit befindliche Produkt die dafür festgelegten Anforderungen erfüllt. In anderen Worten, Verifikation stellt sicher, dass „du das Ding richtig erzeugst“ [CMMI-SW, V1.1]

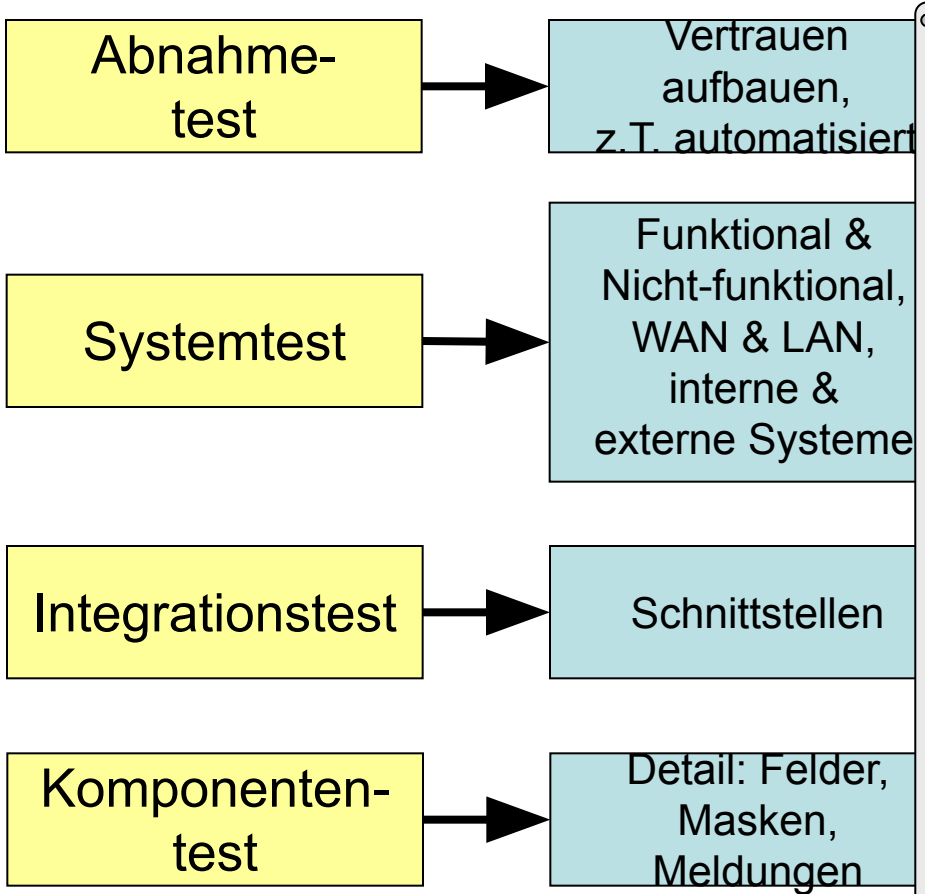
V & V im V – Modell



Gemeinsamkeiten der Softwareentwicklungsmodelle

- Das Testen „begleitet“ die SW-Entwicklung. Jede Entwicklungsaktivität wird von einer entsprechenden Testaktivität begleitet.
- Testentwurf findet während der zugehörigen SW-Entwicklungsstufe statt.
- Das frühzeitige Einbindung der Tester wird gefordert (z.B. Reviews).
- Testziele sind teststufenabhängig (siehe nächste Folie).

Teststufen während des Lebenszyklus



Eine *Teststufe* ist eine Gruppe von Testaktivitäten, die gemeinsam ausgeführt und verwaltet werden.

Inhalt des Testkonzept:

Für jede (auch neu definierte) Stufe:

- Testziele
- Testbasis
- Testobjekt
- Typische Fehlerwirkungen, die gefunden werden sollten
- Eingangs- / Ausgangs-kriterien
- Liefergegenstände
- Testtechniken
- Metriken
- Werkzeuge
- Organisatorische Verantwortlichkeit
- Einzuhaltende Standards

Iterativ-inkrementelle Entwicklungsmodelle

- Beschreibung
  - Das Produkt wird in Phasen (Inkrements) entwickelt.
  - Die Aktivitäten werden in jeder Phase wiederholt.
  - Inkrementell: die Anzahl der Phasen ist zu Beginn bekannt; alle Phasen werden vorab geplant.
  - Evolutionär: die Anzahl der Phasen ist zu Beginn noch nicht bekannt.
  - Verifizierung und Validierung für jede Erweiterung nötig.
  - Regressionstest haben nach dem ersten Inkrement große Bedeutung.
  - Werkzeugeinsatz wird betont.
- Definition eines Inkrements
  - Abgeschlossene funktionale Einheit, inklusive der Begleitdokumentation
- Beispiele
  - Prototyping
  - UP (Unified Process*) – inkrementell
  - RAD (Rapid Application Development) – evolutionär
  - SCRUM – agile Modelle
  - Extreme Programming XP

* früher „Rational Unified Process (RUP), jetzt bei IBM“

(R)UP – (Rational) Unified Process

* Abbildung und deutsche Begriffe in Anlehnung an:
Krutschen, P. „Der Rational Unified Process – eine Einleitung“, 1999

Aufgaben*

Geschäftsprozessmodellierung

Anforderungen

Analyse & Design

Implementierung

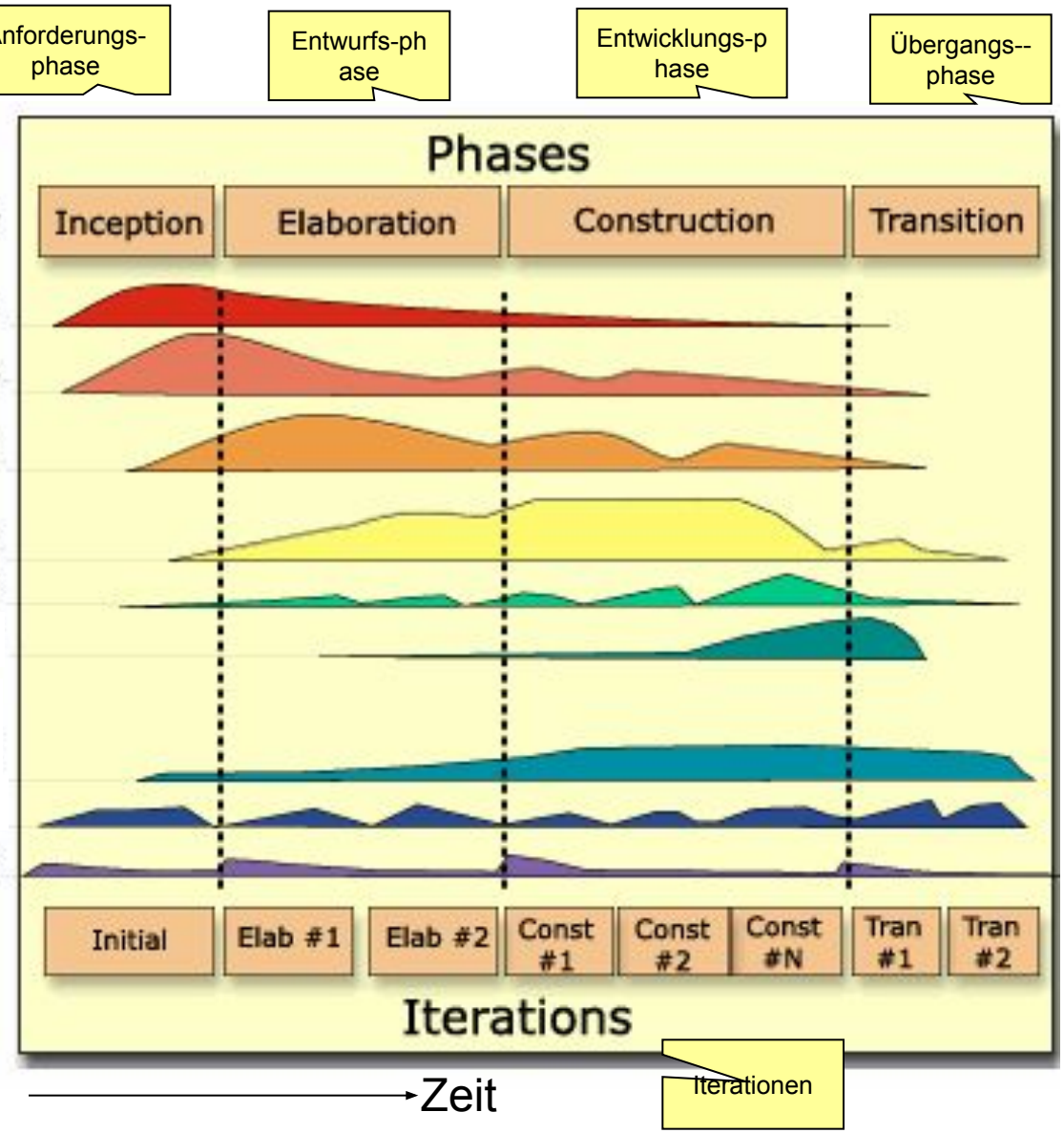
Test

Verteilung

Konfigurations- und Änderungsmanagement

Projektmanagement

Umgebung



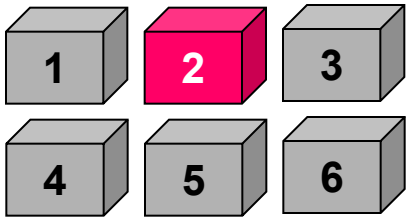
Rational Unified Process
Version
2003.06.01.06

RAD – Rapid Application Development

- Beschreibung
  - Anwender, Designer, Tester, Entwickler bilden ein kleines Team
  - Das Produkt wird in „Time-Boxes“ entwickelt, d.h. feste Liefertermine ohne Verzögerung, ggf. variabler Inhalt
  - Anforderungen, Entwurf, Entwicklung, Test, Review werden pro Iteration durchgeführt
  - Anzahl der Iterationen zu Beginn nicht bekannt
  - Die Entwicklung eines Inkrements hängt von den Informationen aus vorangegangenen Iterationen ab
  - Für kleine Projekte geeignet
- Beispiele
  - Wichtige Vertreter für RAD-Systeme sind IDEs („Integrated Development Environments“) wie Delphi und Kylix [www.borland.com]
  - DSDM (Dynamic System Development Method) [www.dsdm.org]

SCRUM

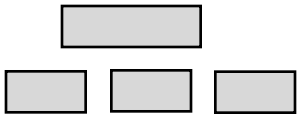
- 3 Rollen: Product Owner, SCRUM Master, Project Team
- Product Backlog: Funktionalität, die zu implementieren ist
- Sprints: Zeitblöcke von typischerweise 30 Tage
- Sprint Backlog: Funktionalität, die zum Sprint gehört
- Tägliche SCRUM-Gespräche (ca. 15 Minuten)
  während des Sprints
  - Was hast Du seit dem letzten SCRUM-Meeting geschafft?
  - Gab es Hindernisse?
  - Was wirst Du bis zum nächsten SCRUM-Meeting erledigt haben?
- Testen wird im gesamten Entwicklungsprozess berücksichtigt.
- Die Rolle des Testers ist in SCRUM nicht explizit definiert
  (in DSDM, das grosse Ähnlichkeiten zu SCRUM aufweist, schon)



Testen im
Lebenszyklus

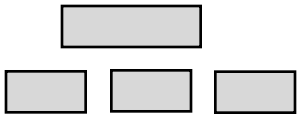
- Softwareentwicklungsmodelle
- **Teststufen**
- Testarten
- Wartungstest

Komponententest im Überblick



nach British Computer Society

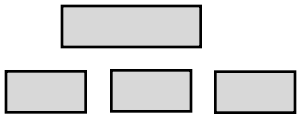
- Komponente: „Das kleinste Softwareelement, wofür eine separate Spezifikation verfügbar ist“
- *Komponententest*: „ Das Testen einer einzelnen Softwarekomponente“
- Andere Bezeichnungen:
  - Modultest - Einzeltest - Unit-Test
  - Klassentest - Entwicklertest - Programmtes
- Höchster Detaillierungsgrad
- Testbasis: Softwareentwurf, Datenmodell
- Entwicklerbeteiligung bzw. -durchführung (Regelfall)
- Gefundene Fehlerzustände werden oft sofort behoben (keine formelle Erfassung)
- Entwicklungsumgebung bzw. -werkzeuge verwendet
- *Platzhalter*, *Treiber* und Simulatoren können schon benötigt werden (siehe Integrationstest)



Komponententest: Testziele und Testobjekte

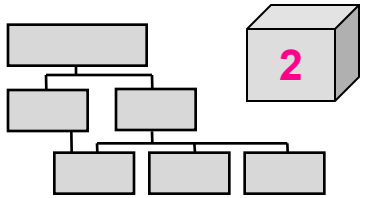
- Funktion der Module
- Struktur (Module, Klassen, Komponenten)
- Datenbankmodule, Migrationsprogramme
- Tests auf *Robustheit* prüfen, ob Komponenten bei ungültigen Eingaben und extremen Umgebungsbedingungen korrekt funktionieren.
- Performance („Benchmarking“)
- Typische Fehlerklassen sind...
  - Fehlende und falsche Struktur
  - Berechnungsfehler
  - Unzureichende Performance
  - Zu geringe Effizienz
  - Speicherfehler

Komponententestplanung



- BS7925-2 „Software Component Test Standard“ unterstützt diese Aktivitäten:
  - Spezifikation der Testtechniken mit Begründung
  - Spezifikation der Vollständigkeitskriterien mit Begründung
  - Dokumentation der Testunabhängigkeit
  - Erstellung eines Komponententestplans, der die Abhängigkeiten zwischen den Komponenten darstellt
- „Test-First-Ansatz“ oder *Testgetriebene Entwicklung*
  - Zuerst Testfälle vorbereiten und ggf. automatisieren
  - Bei kleinen, iterativen Zyklen geeignet

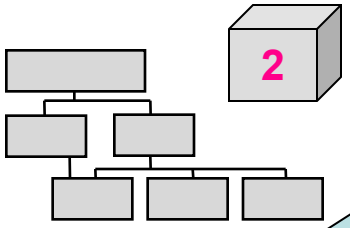
Integrationstest



G Testen mit dem Ziel, Fehlerzustände in den Schnittstellen und im Zusammenspiel zwischen integrierten Komponenten aufzudecken.

- Verbinden von Software- und/oder Hardware-Komponenten zu größeren Baugruppen oder zum Gesamtsystem.
- Schwerpunkt auf den Schnittstellen der Komponenten
- Test kollektiver Funktionen, die nicht an einzelnen Komponenten geprüft werden können
- Nicht-funktionale Prüfungen möglich (z.B. Leistungsfähigkeit)
- Testbasis: SW- und Systementwurf, Architektur, Use-Cases, Workflows

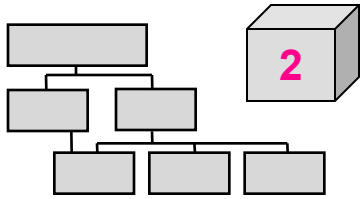
Andere Integrationstestobjekte



- Netzwerke
- Interne Systeme
  - Batch System
- *Standardsoftware (COTS: Commercial Off The Shelf)*

Systemintegrationstest:
Test des kompletten Systems gegen andere Systeme (meist nach dem Systemtest)

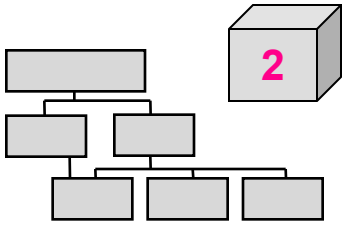
- Externe Systeme
  - Lieferanten von Eingabewerten für das getestete System
  - Abnehmer von Ausgabewerten des getesteten Systems
- Datenaustauschsysteme in Subsystemen
- Kompatibilität mit „koexistierenden“ Systemen
  - Gemeinsam genutzte Hardware (Server, Prozessoren, Laufwerke)
  - Gemeinsam genutzte Datenbanken
  - Gemeinsam genutzte Netzwerke



Integrationsstrategien

Wahl der Integrationsstrategie legt Rahmenbedingungen für die Testablaufplanung fest

- „Big Bang" Integration oder
- Inkrementelle Integration
  - Top-Down
  - Bottom-Up
  - Ad-Hoc
  - Funktional
  - Prozessgetrieben

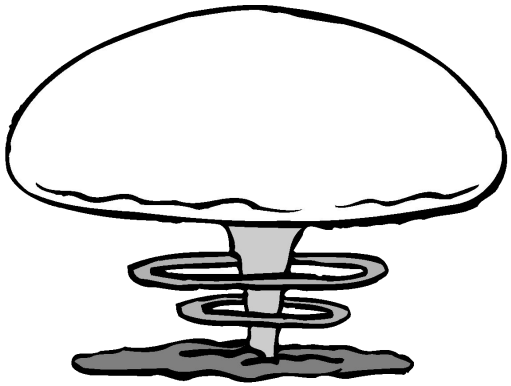


Die „*Big Bang*“ Integration

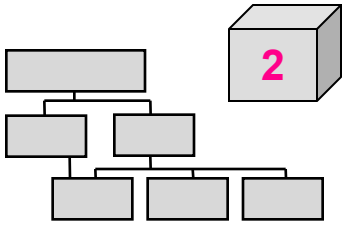
- Hypothese: Einzeln getestete Komponenten sollten ohne weitere Tests zusammenpassen
- Häufige Motivation: „Zeitersparnis“

PROBLEME ...

- Die Grundannahme: „Keine weiteren Fehlerzustände nach Test der Einzelkomponenten“ ist unrealistisch
- Die Fehlerlokalisierung wird unweigerlich schwieriger und zeitaufwändiger
- Regressionen treten häufiger auf



Inkrementelle Integrationsstrategie



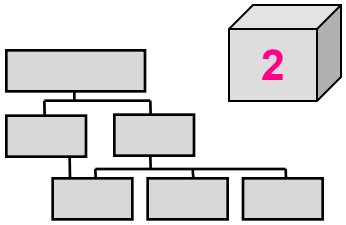
Komponenten oder Systeme werden sukzessiv (als „Baselines“) integriert und getestet, bis alle integriert und getestet sind.

Test definierter Baselines:

- Baseline 0 : Test einer Einzelkomponente
- Baseline 1 : Gemeinsamer Test zweier Komponenten
- Baseline 2 : Gemeinsamer Test von drei Komponenten
- Baseline X : etc.

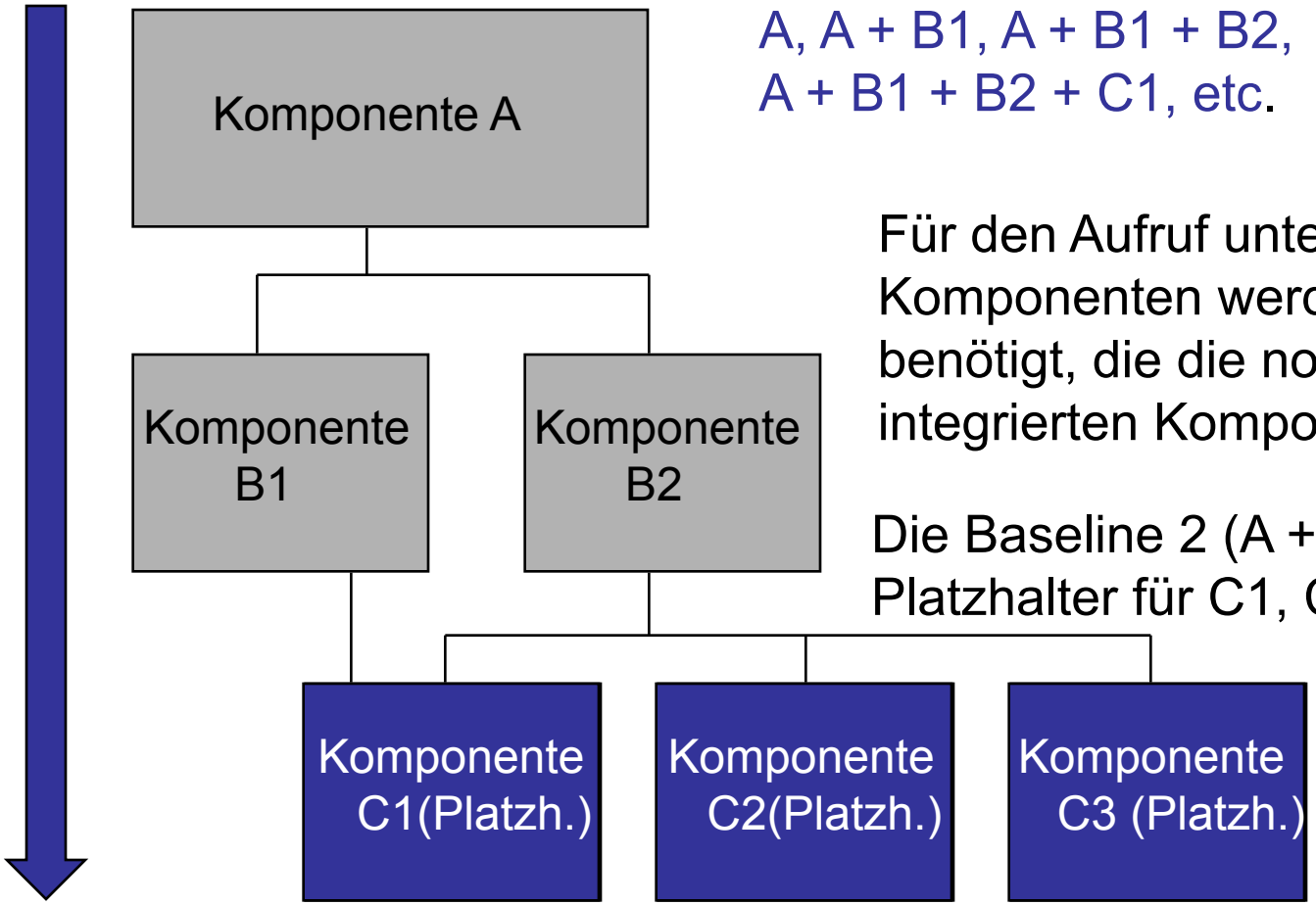
Vorteile:

- Einfachere Fehlerlokalisierung und -korrektur
- Einfachere Problembehebung
- „Fallback“ auf frühere Baseline möglich
- Fehlerzustände werden an den Schnittstellen gefunden
- Kontrollierter, steuerbarer Testfortschritt



„Top-Down“ Integration

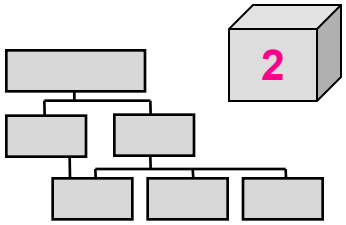
Baseline-Abfolge:
A, A + B1, A + B1 + B2,
A + B1 + B2 + C1, etc.



Für den Aufruf untergeordneter Komponenten werden *Platzhalter* benötigt, die die noch nicht integrierten Komponenten vertreten.

Die Baseline 2 (A + B1 + B2) braucht Platzhalter für C1, C2 und C3

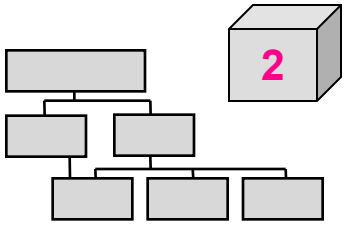
Platzhalter
= Stub



Tipps für den Gebrauch von *Platzhaltern*

- *Platzhalter* ...
  - Ersetzen im Integrationstest aufzurufende Komponenten.
  - Emulieren die aufgerufene Komponente.
- Wie detailliert soll die Emulation sein?
  - Fester Rückgabewert
  - Einfache Berechnung oder Zufallswert
  - Eingabe der Antwort durch den Tester
  - Simulation von Bearbeitungszeiten
- Mögliche Probleme?
  - Zu komplizierte *Platzhalter* (zu viel Logik oder Code)
  - Wartung der *Platzhalter* wird zu aufwändig
  - *Platzhalter* repräsentieren mangels Pflege nicht mehr die echten Komponenten

„Top-Down“ Integration

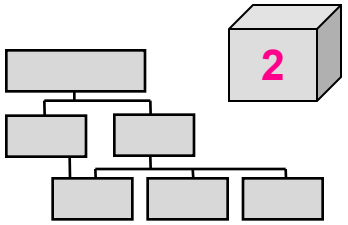


Vorteile

- Kritische Kontrollflüsse werden zuerst und am häufigsten getestet
- Ein funktionierendes Modellsystem ist früh verfügbar
- Ideal für Prototyping
- Keine *Treiber* werden benötigt (siehe S.26)

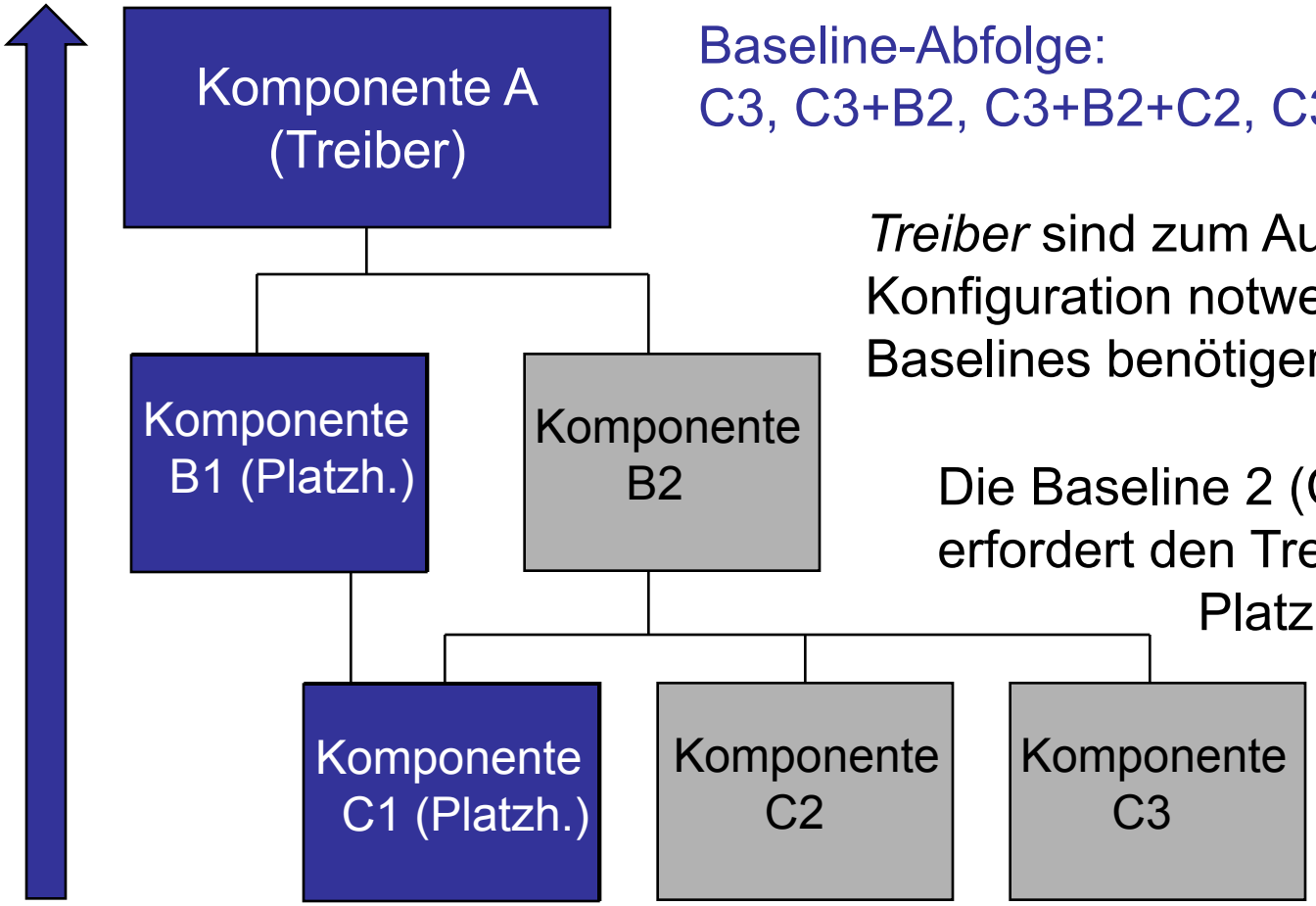
Nachteile

- Details kommen als Letztes
- Das 95% Problem: „Der Teufel steckt im Detail“
- Entwicklung und Pflege der *Platzhalter* erfordern Aufwand
- Versuchung, die *Platzhalter* zu kompliziert zu machen



„Bottom-Up“ Integration

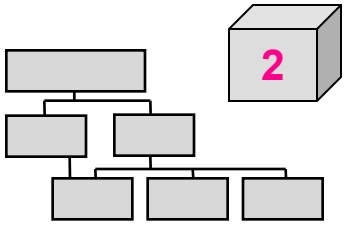
Treiber = Driver



Baseline-Abfolge:
C3, C3+B2, C3+B2+C2, C3+B2+C2+A etc.

Treiber sind zum Aufruf der Baseline Konfiguration notwendig. Manche Baselines benötigen auch Platzhalter.

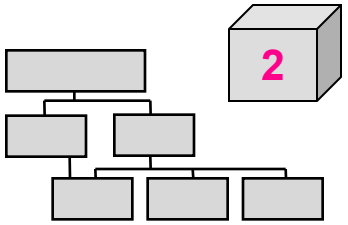
Die Baseline 2 (C3+B2+C2) erfordert den Treiber A und die Platzhalter B1 und C1



Tipps für den Gebrauch von *Treibern*

- Treiber bilden das Gerüst des Systems
- Wie komplex sollten die Treiber sein?
  - Grundlogik des Aufrufs
  - Einfacher Datenempfänger für die Baseline
  - Senden der für die Baseline wesentlichen Daten
- Mögliche Probleme?
  - Fehlende Updates der Treiber im Laufe der Baselines
  - Die Aufrufslogik wird zu komplex gemacht

„Bottom-Up“ Integration

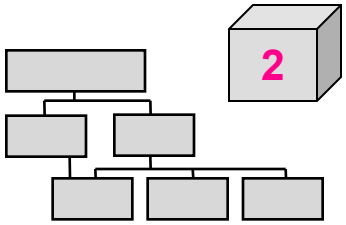


Vorteile

- Schnittstellen zu „Low-Level“-Komponenten werden zuerst und sehr gründlich getestet
- Funktionale Details können früh untersucht werden
- Ideal für die Prüfung von Schnittstellen zu System-komponenten wie Hard-ware, Middleware (RMI, CORBA etc)

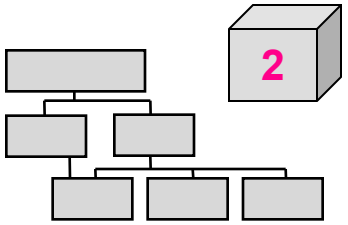
Nachteile

- Funktionierendes Modell wird erst mit der letzten Baseline verfügbar
- Kritische Kontrollprobleme fallen oft erst zum Schluss auf
- Häufige Anpassungen der Treiber bei neuen Baselines können speziell bei kritischen Systemen erheblichen Aufwand verursachen
- *Platzhalter* sind ebenfalls nötig



Allgemeine Tipps zum Integrationstest

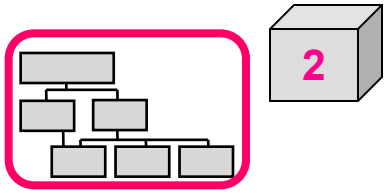
- Möglichst wenig neue Komponenten pro Baseline integrieren
- Risikoreiche Komponenten (z.B. geschäftskritische oder fehleranfällige) einzeln integrieren
- Einfache und verwandte Komponenten gemeinsam integrieren
- Jede Baseline sollte ein einfach verifizierbares Ergebnis hervorbringen
- *Treiber* und *Platzhalter* auf ein Minimum beschränken
- Ausschließlich auf die eigentliche Integration konzentrieren



Voraussetzungen für erfolgreiche Integrationstests

- Tester haben ein Verständnis für die Architektur und sind in die Integrationsplanung mit einbezogen
- Die Integrationstests werden frühzeitig im Lebenszyklus eingeplant.
- Die geplanten Baselines treiben die Reihenfolge, in der die Komponenten entwickelt werden.
  - Komponenten werden aus Zeitersparnis parallel zum Integrationstest entwickelt
  - Die Fertigstellung der Komponenten wird anhand der Baselines geplant
  - Nicht möglich bei „Ad-Hoc“-Integration

Andere Integrationsstrategien

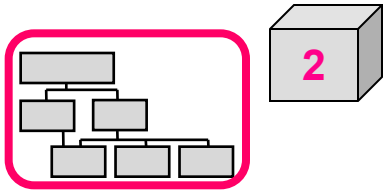


Systemtest im Überblick

- Testgegenstand ist das gesamte System, Handbücher, Konfigurationsdaten
- Testumgebung sollte möglichst nahe an der Zielumgebung sein
- Es ist meist zu riskant die Produktivumgebung selbst zu verwenden (Verlust von „echten“ Daten, potentielle Betriebsstörungen, usw.)
- Die Testtypen zerfallen in zwei Klassen:

  Funktionale Tests
  - Basierend auf funktionalen Anforderungen
  - Basierend auf Geschäftsprozessen
  - Meist Black-Box-Testverfahren, aber strukturorientierte Tests (White-Box-Testverfahren) sind möglich (z.B. zur Überdeckung der Menüstrukturen oder Navigationsstrukturen eines Client-Systems)

  Nichtfunktionale (technische) Tests
  - Basierend auf technischen Qualitätsmerkmalen (siehe ISO9126) wie z.B. Performance, Stabilität, Ausfallsicherheit, Datensicherheit
  - Der Systemtest wird häufig von unabhängigen Teams mit speziellen Fähigkeiten in den einzelnen Testtypen durchgeführt.



Definitionen: Funktionaler Systemtest

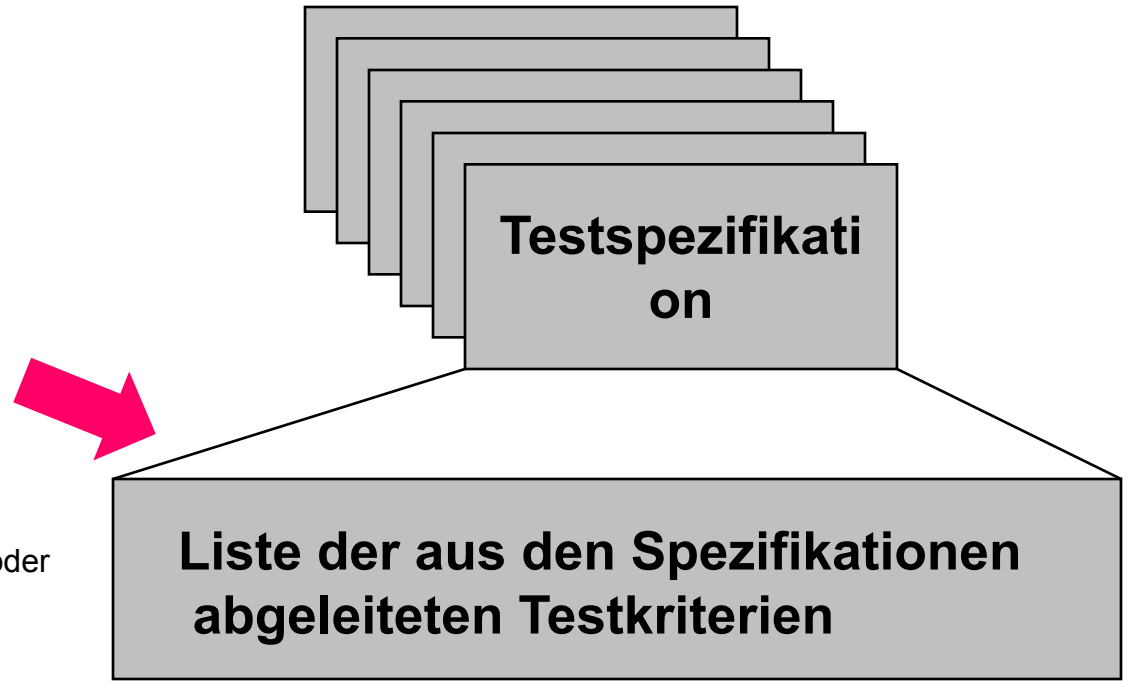
G *Funktionale Anforderung:*

- Anforderung, die ein funktionales Verhalten spezifiziert, das ein System oder eine Systemkomponente erbringen muss. [IEEE 610]

- Funktionaler Systemtest:
  - „Ein Test bzw. eine Testphase, in dem das gesamte System gegen seine Spezifikationen getestet wird“

Anforderungsbasiertes Testen

Spezifikation der Systemanforderungen
oder
Benutzeranforderungen (Pflichtenheft, Lastenheft)

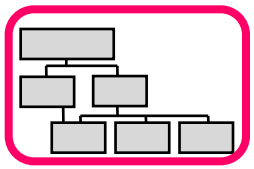


Bemerkung:
Der Tester muss oft mit unvollständigen oder nicht aktuellen Anforderungsdokumenten arbeiten.

Was tun?

- Diskutieren mit wichtigen Stakeholders
- Annahmen treffen und dokumentieren
- Die Annahmen überprüfen bzw. genehmigen lassen
- „Anforderungen“ regelmäßig aktualisieren

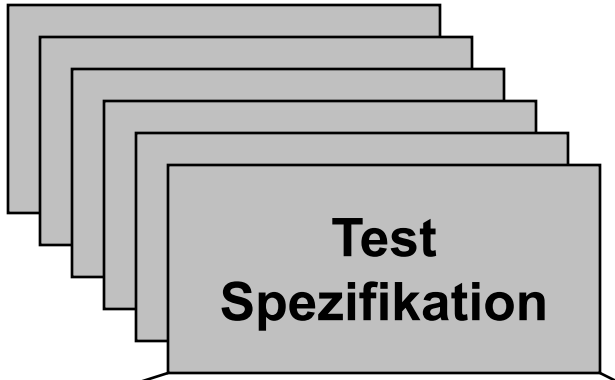
Geschäftsprozessbasiertes Testen



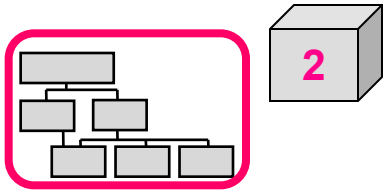
Risikobewertung
- Wahrscheinlichkeit
- Auswirkung

Geschäftsszenarien
„End-To-End“
Geschäftsabläufe

Use Cases
Anwendungsfälle aus realen Situationen



Liste der aus den Prozessen abgeleiteten Testkriterien



Nicht-funktionaler Systemtest

- Nicht-funktionale Anforderung
  - G Eine Anforderung welche sich nicht auf die Funktionalität des Systems bezieht sondern auf Merkmale wie Zuverlässigkeit, Benutzbarkeit, Effizienz, Änderbarkeit und Übertragbarkeit.
    - Beschreibt Attribute des Systemverhaltens, also wie gut bzw. mit welcher Qualität das System seine Funktion erbringen soll. Um-setzung beeinflusst stark, wie zufrieden der Kunde bzw. Anwender mit dem Produkt ist. [ISO 9126]
- Nicht-funktionaler Test
  - G Testen der Eigenschaften eines System, die nicht direkt mit der Funktionalität in Verbindung stehen, z.B. Zuverlässigkeit, Effizienz, Benutzbarkeit, Änderbarkeit und Übertragbarkeit.
    - Mindestens genauso wichtig wie rein funktionale Tests
    - Die Vernachlässigung dieser Tests kann ein beträchtliches Risiko für den Erfolg der Anwendung bedeuten.
    - Spezifikation nicht-funktionaler Anforderungen wird oft vernachlässigt
      - Spezifikation wird häufig vergessen
      - Spezifikation ist oft nicht testbar (d.h. Anforderungen sind manchmal schwer zu quantifizieren)

Abnahmetest

G Formales Testen (ggf. unter Beteiligung des Auftraggebers) hinsichtlich der Benutzeranforderungen und -bedürfnisse bzw. der Geschäftsprozesse. Es wird durchgeführt, um einem Auftraggeber oder einer bevollmächtigten Instanz die Entscheidung auf der Basis der Abnahmekriterien zu ermöglichen, ob ein System anzunehmen ist oder nicht [nach IEEE 610]

Abnahmetest im Überblick

- Test unter Beteiligung des Auftraggebers, der Anwender des Systems und anderer Stakeholder
- Testbasis
  - Die expliziten *Anforderungen* des Auftraggebers, wie sie in einem Dokument (z.B. Pflichtenheft oder Fachkonzept) für beide Seiten verbindlich festgelegt sind.
  - Die impliziten Erwartungen des Auftraggebers, die dem allgemeinen Stand der Technik entsprechen.
  - Risikoanalysen
- Testfälle ähnlich wie im *Systemtest*, in der Regel jedoch nicht so ausführlich bzw. umfangreich.
- Fokus liegt darauf
  - Vertrauen aufzubauen (funktionale und technische Qualitätsaspekte).
  - Betriebsbereitschaft festzustellen.
  - Restrisiken für die Produktion einzuschätzen.

Abnahmetest: Aspekte

- Folgende Aspekte können relevant sein:
  - *Benutzer-Abnahmetest*
  - *Betrieblicher Abnahmetest*
  - Vertraglicher und regulativer Abnahmetest
  - *Alpha- und Beta-Test (Feldtest)*

Siehe folgende Folien

- Teilabnahmen können sinnvoll sein, z.B.
  - Bei der Integration von Standard-Software (Datenbanken, *COTS*)
  - Bei der Abnahme von spezifischen Qualitätseigenschaften (Benutzbarkeit, Sicherheit)
  - Bei der Abnahme von neuen Funktionen

- Zeitliche Aspekte:
  - Teilabnahmen können in früheren Teststufen durchgeführt werden
  - Nach der Abnahme können weitere Tests folgen (z.B. Systemintegrationstests)

Benutzer-Abnahmetest

- Test auf Benutzerakzeptanz (Validierung)
- Starke Einbeziehung der Endanwender
  (auch wenn sie die Tests nicht selbst durchführen)
- Hauptsächlich funktionale Tests, die auf Geschäftsprozessen basieren
- Realistische *Testumgebung*, meist mit „echten“ Daten
- Automatisierte Tests sind möglich
- Es kann eine formale (eventuell vertragsrelevante) Abnahme des Systems erfolgen

Einbeziehung der Benutzer - Vorteile

- Feedback
  - Ist das geplante oder fertige System brauchbar? □ Validierung
  - Die Benutzer kennen ihr Handwerk in all seiner Komplexität und wissen am besten, was das System leisten muss.
  - Benutzer können realistische Einsatzszenarien liefern, die auch für den Entwurf von Testfälle nützlich sind.
  - Benutzer können Workarounds für Probleme vorschlagen und Tipps zu Varianten der Standardanwendungsfälle geben.
- Bessere Akzeptanz
  - Die Einbeziehung von Benutzern in verschiedenen Stadien der Entwicklung schafft ein positives, kooperatives Klima.
  - Offene Kommunikation mildert typische Hemmschwellen der Veränderung (z.B. ungewohnte Bezeichnungen, betriebliche Risiken)
  - Benutzer werden seltener in der Abnahme „ihre Rache nehmen“

Benutzer akzeptieren ein neues System eher, wenn sie sich damit identifizieren können

Test auf vertragliche & regulative Akzeptanz

- Vertraglicher Abnahmetest
- Vertrag über Lieferung des Software-Systems
  - Definiert, ausgehandelt und unterzeichnet zum Projekt
  - Abnahmekriterien sind definiert und abgestimmt
  - Spezifizierter Liefergegenstand (Hardware, Software, Dokumente)
  - Definierte Mitwirkungen der Anwender (z.B. Entwurf der Testfälle, Bereitstellung von Testdaten, Ansprechpartner zu Fachthemen etc.)
- Der Abnahmetest erfolgt auf Basis des Vertrags und der abgestimmten Systemänderungen
  - Klar spezifizierte Anforderungen sind unabdingbar
  - Unerfüllte Anwenderwünsche müssen berücksichtigt werden, laufen aber separat z.B. als Change Request für ein neues Release
- *Regulativer Abnahmetest / Konformitätstest*
  - Bezug auf die Regularien, die zu erfüllen sind (z.B. Sicherheitsregeln für Atomkraftwerke, Bundesgesetzte für Banken und Versicherungen)

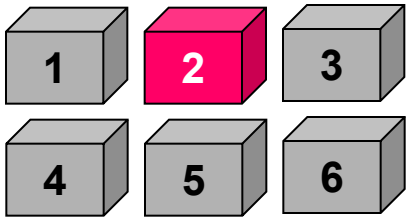
Alphatest und *Betatest* („*Feldtest*“)

- Realistischer Test des stabilen Systems durch eine Gruppe, die die Endanwender repräsentiert.
- Die *Testumgebung* ist womöglich produktionsidentisch aufgebaut.
- Feedback zu: Nutzerfreundlichkeit des Systems, Erfüllung der Geschäftsanforderungen, gefundene Fehlerwirkungen, Verbesserungsvorschläge.
- Alpha- und Betatest unterscheiden sich primär durch den Teststandort.
  - *Alphatest*: Test in (simulierter) Produktion am Entwicklungsort auf einer von der Entwicklung getrennten Umgebung.
  - *Betatest / Feldtest*: Test in Produktion an einem von der Entwicklung unabhängigen Teststandort.
- Andere Bezeichnungen: Außenabnahmetest, Fabrik-Abnahmetest, Kundenakzeptanztest

Betrieblicher Abnahmetest

G Betriebstest in der Abnahmeteststufe, üblicherweise in einer simulierten Produktionsumgebung durch den Betreiber und/oder Administrator durchgeführt, mit Schwerpunkt bei den operationalen Aspekten, z.B. Wiederherstellbarkeit, Ressourcenverwendung, Installierbarkeit und technische Konformität

- Synonym: *operationaler Abnahmetest*
- Durch den oder im Auftrag des Betreibers oder Administrators
- Untersucht in der Regel *nicht-funktionale Anforderungen*
  - Backup / Restore
  - Wiederherstellbarkeit nach Ausfällen
  - Benutzermanagement
  - Wartungsaufgaben, auch organisatorische Aspekte
  - Datenlade- und Migrationsaufgaben
  - Überprüfung von Sicherheitslücken



Testen im
Lebenszyklus

- Softwareentwicklungsmodelle
- Teststufen
- **Testarten**
- Wartungstest

Testart / Testtyp

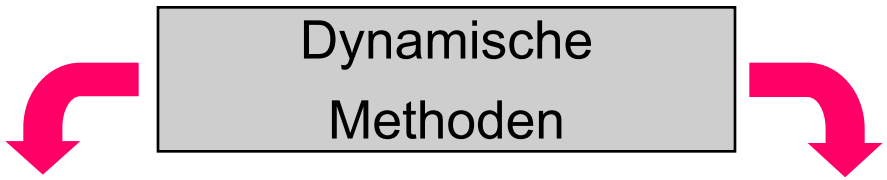
G Eine *Testart* ist eine Gruppe von Testaktivitäten, die gemeinsam ausgeführt und verwaltet werden, mit dem Ziel der Überprüfung einer Komponente und eines Systems auf einige zusammenhängende Qualitätsmerkmale.

- Eine Testart kann sich auf bestimmte Testziele beziehen, wie z.B. Zuverlässigkeitstest, Regressionstest, Benutzbarkeitstest.
- Die Testart kann sich auch auf eine oder mehrere Testebenen oder -phasen beziehen.

Wir betrachten im folgenden diese 4 Testarten:

- Testen einer zu erfüllenden Funktion
- Testen einer nicht-funktionalen Anforderung
- Testen der Struktur oder Architektur der Software beziehungsweise des Systems
- Prüfen auf erfolgreiche Beseitigung eines Fehlers (Nachtest) oder Prüfen auf unbeabsichtigte beziehungsweise ungewollte Änderungen oder Seiteneffekte (Regressionstest)

Testart / Testtyp im Überblick



* siehe BS 7925-2

**

White Box
(Struktureller Test)*
Abdeckung von…
3
Codeanweisungen,
Zweigen, Pfaden, …

Black-Box (*Spezifikationsbasierter Test*)

Abdeckung von…	Funktional *	Nicht-funktional **
Anforderungen	X 1	X 2
Geschäftsprozessen	X	

4 Nachtest und Regressionstest

Testentwurfsmethoden für 1 und 3 in Modul 4

Qualitätsmerkmale nach DIN-ISO 9126

Qualitätsmerkmal	Aspekt
Funktionalität	Richtigkeit Angemessenheit Ordnungsmäßigkeit Interoperabilität Sicherheit
Zuverlässigkeit	Reife Fehlertoleranz Wiederherstellbarkeit
Benutzbarkeit	Verständlichkeit Erlernbarkeit Bedienbarkeit
Effizienz	Verbrauchsverhalten Zeitverhalten
Änderbarkeit	Prüfbarkeit Stabilität Analysierbarkeit Modifizierbarkeit
Übertragbarkeit	Austauschbarkeit Koexistenz Installierbarkeit Anpassbarkeit

- Es gibt viele Arten funktionaler und nicht-funktionaler Tests.
- Nicht alle davon müssen für eine gegebene Anwendung sinnvoll sein.
- Zusätzlicher Aspekt: Verträglichkeit mit jeweiligen Standards

Qualitätsmerkmal „*Funktionalität*“

- Die Funktionalität besagt, „was“ das System leisten soll
- Richtigkeit
  - Sind alle Berechnungen richtig?
  - Werden die richtigen Daten angezeigt oder abgespeichert?
  - Werden die richtigen Funktionen oder Routinen aufgerufen?
- Angemessenheit
  - So komplex wie nötig, so einfach wie möglich
- Ordnungsmäßigkeit
  - Erfüllung von Normen und gesetzlichen Bestimmungen
- *Interoperabilität*
  - Passen die beiden Seiten einer Schnittstelle zusammen?
  - Werden passende Daten übertragen?
  - Werden passende Datenformate verwendet?
  - Sind die Felder gleich lang?
- *Sicherheit (siehe nächste Folie)*

für alle Teststufen relevant

Qualitätsmerkmal „*Sicherheit*“

- Verschlüsselung
  - Eingabe und vorausgesagtes Ergebnis spezifizierbar?
  - Ist die Verschlüsselung leicht überwindbar?
- Berechtigungen und Passwörter
  - Sind Passwörter hinreichend geschützt?
  - Sind die Passwörter leicht zu knacken?
  - Welche Berechtigungsstufen gibt es? (Daten, Funktionen, ..)
  - Ist die Hardwareseite genügend gesichert?
- Weitere Sicherheitsmaßnahmen
  - Organisatorische Abläufe
  - Physikalische Absicherung
  - Sicherheitskonzepte
  - Systemarchitektur

Qualitätsmerkmale nach DIN-ISO 9126

Qualitätsmerkmal	Aspekt
Funktionalität	Richtigkeit Angemessenheit Ordnungsmäßigkeit Interoperabilität Sicherheit
Zuverlässigkeit	Reife Fehlertoleranz Wiederherstellbarkeit
Benutzbarkeit	Verständlichkeit Erlernbarkeit Bedienbarkeit
Effizienz	Verbrauchsverhalten Zeitverhalten
Änderbarkeit	Prüfbarkeit Stabilität Analysierbarkeit Modifizierbarkeit
Übertragbarkeit	Austauschbarkeit Koexistenz Installierbarkeit Anpassbarkeit

- Es gibt viele Arten funktionaler und nicht-funktionaler Tests.
- Nicht-funktionale Merkmale besagen, „wie“ das System arbeitet.
- Nicht alle davon müssen für eine gegebene Anwendung sinnvoll sein.
- Zusätzlicher Aspekt: Verträglichkeit mit jeweiligen Standards

Qualitätsmerkmal „*Zuverlässigkeit*“

- *Zuverlässigkeit*
  - Merkmal aus Nutzersicht
  - Wird häufig als nicht testbare Anforderung bezeichnet
  - Sinnvolle Messgröße: Mittlere Zeit zwischen Ausfällen (MTBF)
  - Die 24*7 Anforderung für E-Commerce Anwendungen

- Backup & Recovery („Wiederherstellbarkeit“)
  - Backup als automatisiertes und manuelles Verfahren
  - Wiederherstellung gesicherter Daten (Software, Dokumente, etc.)
    - Test des gesamten Ablaufs (Vorgänge, Rollen, etc.)
    - Test der Dokumentation
    - Regelmäßiger Test im Rahmen des Betriebs

Qualitätsmerkmal „*Benutzbarkeit*“

- Merkmal aus Nutzersicht
- Aussagefähigkeit von Meldungen
  - Kann der Benutzer darauf reagieren?
  - Werden für den Benutzer verständliche Begriffe verwendet?
- Konsistentes Erscheinungsbild der Nutzeroberfläche
  - Sind die relevanten Standards berücksichtigt?
- Verständlichkeit und Erlernbarkeit für den Benutzer
  - Überflutung mit Auswahlmöglichkeiten oder kritischer Information?
  - Ist dem Benutzer klar, was das System tut? (Feedback)
- Vorgehensweise beim Test der Nutzerfreundlichkeit
  - Wer testet? Endanwender oder unabhängige Tester?
  - Wann? Nach Fertigstellung oder prozessbegleitend?

Qualitätsmerkmal „Effizienz“

- Merkmal aus Nutzersicht
- *Performance*
  - Antwortzeit auf eine Benutzereingabe in Sekunden
  - Dauer einer komplexen Datenbanksuche, in Millisekunden
  - Bearbeitungsdauer einer Unterroutine, in CPU Zyklen
- *Lasttest*, Massentest (Kapazität, Datenvolumen)
  - Test unter hoher Nutzungsintensität
  - Maximale geforderte Bearbeitungsrate (z.B. Transaktionen pro Sekunde)
  - Maximaler geforderte Datendurchsatz (z.B. Kilobyte pro Sekunde)
  - Maximalzahl paralleler Zugriffe oder Prozesse
- *Stress*- und Skalierungstest
  - Wo liegen die Belastungsgrenzen des Systems?
  - Werden kurze Lastspitzen oder das geplante Wachstum verkraftet?
  - Wie reagiert das System bei Erreichen bzw. Überschreiten der Belastungsgrenzen?

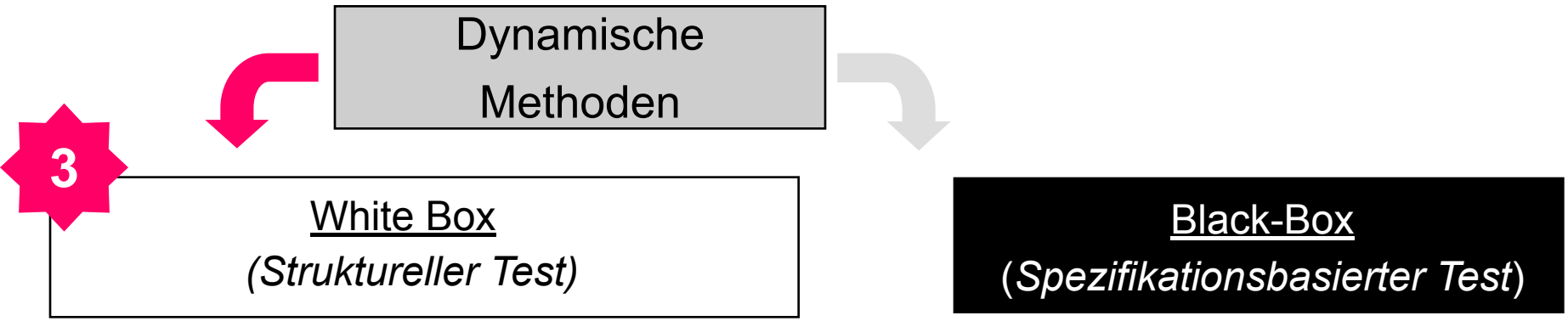
Qualitätsmerkmal „Änderbarkeit / Wartbarkeit“

- Merkmal aus der Sicht der Entwicklung und Wartung
- Analysierbarkeit
  - Aufwand, um Mängel oder Ursachen von Versagen zu diagnostizieren oder um änderungsbedürftige Teile zu bestimmen
- Modifizierbarkeit
  - Aufwand zur Ausführung von Verbesserungen, zur Fehlerbeseitigung oder zur Anpassung an Umgebungsänderungen
- Stabilität
  - Wahrscheinlichkeit des Auftretens unerwarteter (Neben-) Wirkungen von Änderungen
- Prüfbarkeit
  - Aufwand, der zur Prüfung der geänderten Software notwendig ist

Qualitätsmerkmal „Übertragbarkeit / *Portabilität*“

- Merkmal aus der Sicht der Entwicklung und Wartung
- Anpassbarkeit (Konfiguration)
  - Kombination verschiedener Hardware- und Softwareplattformen
  - Mögliche Systemkonfigurationen
  - Konfiguration der eigenen Software und fremder Software
- Installierbarkeit
  - Wie wird die Software für den Wirkbetrieb installiert?
    - Verteilungskanal (CD, “Push”-Technologie, Netzinstallation)
  - Installationsverfahren
    - für die einmalige Installation und für geplante Updates
  - Verfahren und Nebeneffekte der De-Installation
  - Physikalische Rahmenbedingungen: Elektromagnetische Abschirmung, Hitze, Feuchtigkeit, Vibration, Stromversorgung, ...
- Austauschbarkeit
  - Wie leicht lässt sich Software austauschen?

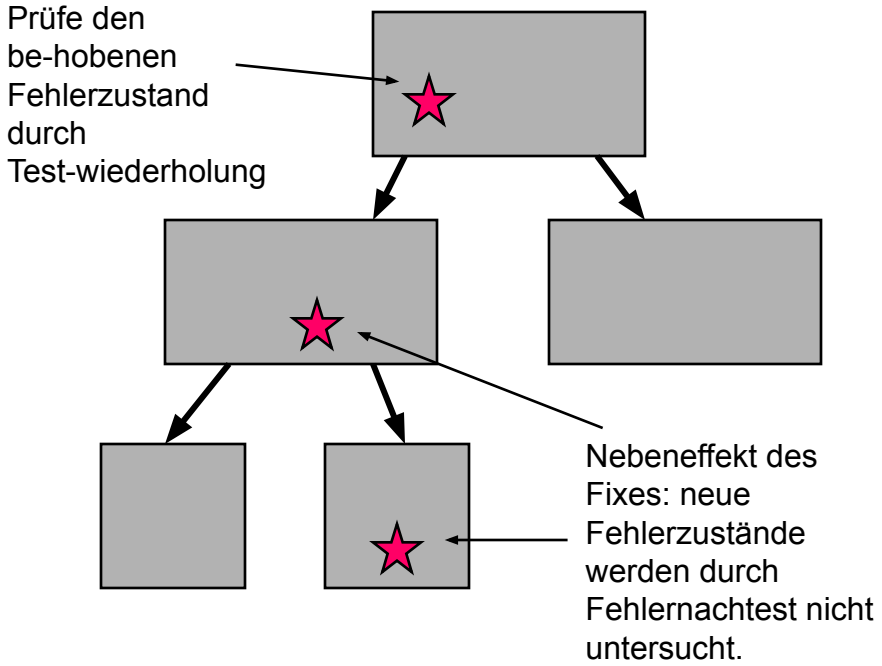
Struktureller Test



- Strukturelles Testen (strukturorientierter Test) kann in allen Teststufen angewendet werden
- Verschiedene Testüberdeckungen werden als Maß für eine Vollständigkeit einzelner Testsuiten verwendet, z.B.
  - Anweisungsüberdeckung
  - Entscheidungs- oder Zweigüberdeckung
- Unterstützung durch entsprechende Testwerkzeuge ist notwendig
- Strukturelle Tests in höheren Teststufen beziehen sich auf die Systemarchitektur (z.B. Menüstrukturen, Aufrufhierarchien, Geschäftsmodelle)

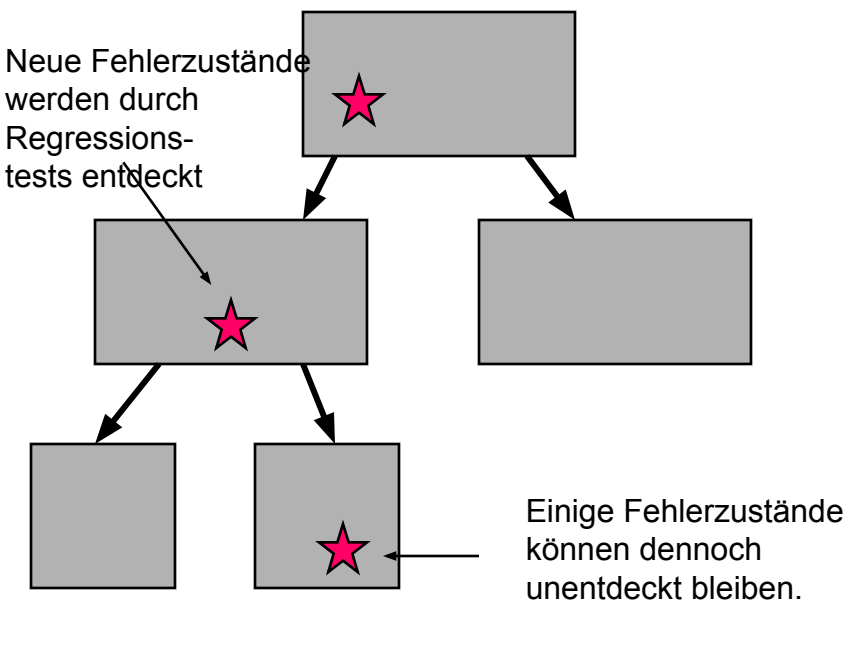
Nachtest und *Regressionstest*

G Wiederholung aller Testfälle, die vor der Fehlerkorrektur eine Fehlerwirkung erzeugt haben; dient der Überprüfung, ob die Korrektur des ursächlichen Fehlerzustands erfolgreich war.



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

G 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.

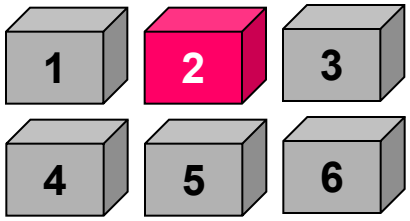


Regressionstests:
- Benötigen mehr Aufwand.
- Umfang ist abhängig vom Risiko hinsichtlich neu eingebrachter Fehlerzustände.
- Sind für die Erhaltung der SW-Qualität wichtig.

Regressionstests

- Können in jeder Teststufe angewendet werden.
- Können funktionale, nicht-funktionale und strukturelle Testinhalte abdecken.
- Bestehen aus einer vordefinierten Standardgruppe von Tests („Regression *Test Suite*“), die regelmässig laufen.
- Organisiere die „Suite“ durch Untergruppen von Tests, die immer zusammen laufen können (z.B. Tests, die bestimmte Funktionen abdecken)
- Redundante, ineffektive oder nicht mehr zutreffende Tests müssen entfernt und neue hinzugefügt werden.
- Die Erstellung und insbesondere die Wartung einer „Regression Test Suite“ erfordern Aufwand.
  Plane den Aufwand!
- Sind erste Kandidaten für eine Automatisierung.

mehr im Modul 6:
Testwerkzeuge



Testen im
Lebenszyklus

- Softwareentwicklungsmodelle
- Teststufen
- Testarten
- **Wartungstest**

Wartungstest

- Anlass: ein SW-System, das sich (seit einiger Zeit) im Betrieb befindet, muss geändert (modifiziert) werden.
- Modifikationen am System erfordern Wartungstests
  - Geplante Spezifikationsänderung (z.B. Funktionserweiterung auf Wunsch des Kunden)
  - Notfallkorrekturen (z.B. Hotfix)
  - Änderung der HW- oder SW-Plattform (Datenbank, Betriebssystem)
- Datenmigrationen in ein neues System erfordern Wartungstests
  - Z.B. nach Außerbetriebnahme/Einzug („Abschalten“) eines veralteten Systems, hier auch ggf. Test der Archivierung
- Fehlende oder veraltete Spezifikationen können sich als ein großes Problem herausstellen
- Änderungen, Upgrades oder Hotfixes können ungewollte Nebenwirkungen haben

Wartungstest: die Strategie

- Entscheide aufgrund des Umfangs und des Inhalts der Änderungen, welche Auswirkungen die Änderungen auf das System haben (engl. impact analysis)
- Starte mit breit angelegten Tests („Smoke Tests“), um ein Grundvertrauen zu gewinnen, dass das System noch intakt ist
- Im Anschluss Durchführung tieferer Tests für besonders kritische Bereiche basierend auf der Risikoanalyse
- Darüber hinaus Durchführung von Regressionstests aus der zur Verfügung stehenden „Regression Test Suite“ sofern nötig
- Die Auswirkungsanalyse gibt auch Aufschluss darüber, welche Teile der „Regression Test Suite“ durchgeführt werden sollen.

Umfang und Tiefe der Tests sind vom Risiko bestimmt!

Zusammenfassung: Testen im SW-Lebenszyklus

- V-Modell unterscheidet Teststufen und motiviert frühen Testentwurf
- Verschiedene Softwareentwicklungmodellen erfordern verschiedene Testansätze
- Der Komponententest liegt in der Verantwortung der Entwicklung
- Der Integrationstest erfolgt zunächst auf Komponentenebene
- Der Systemtest prüft funktionale und nicht-funktionale Kriterien
- Anschließend Test der Integration auf Systemebene
- Der Abnahmetest bezieht die Benutzer mit ein
- Unterschiedliche Testziele und Testarten sind Basis für eine erfolgreiche Teststrategie
- Der Wartungstest dient dem Erhalt der Systemqualität

Änderungsverzeichnis

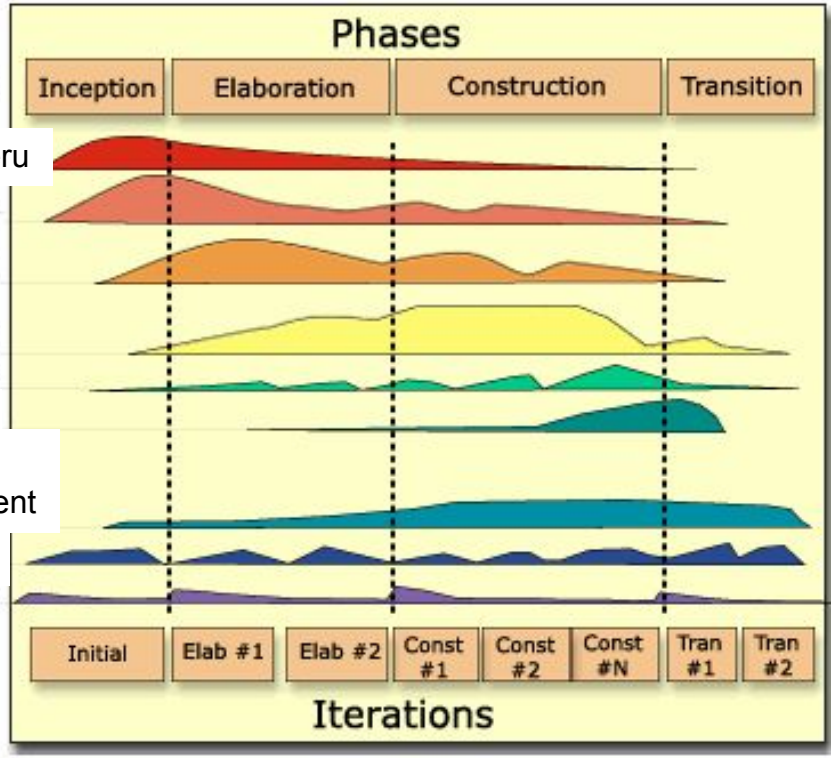
Datum	Version	Autor	Bemerkung
28.02.2011	3.0	S.Massonet	Re-Akkreditierung zum LP 2010

RUP – Rational Unified Process

Anforderung sphase | Entwurfs-ph ase | Entwicklung s-phase | Übergangs-- phase

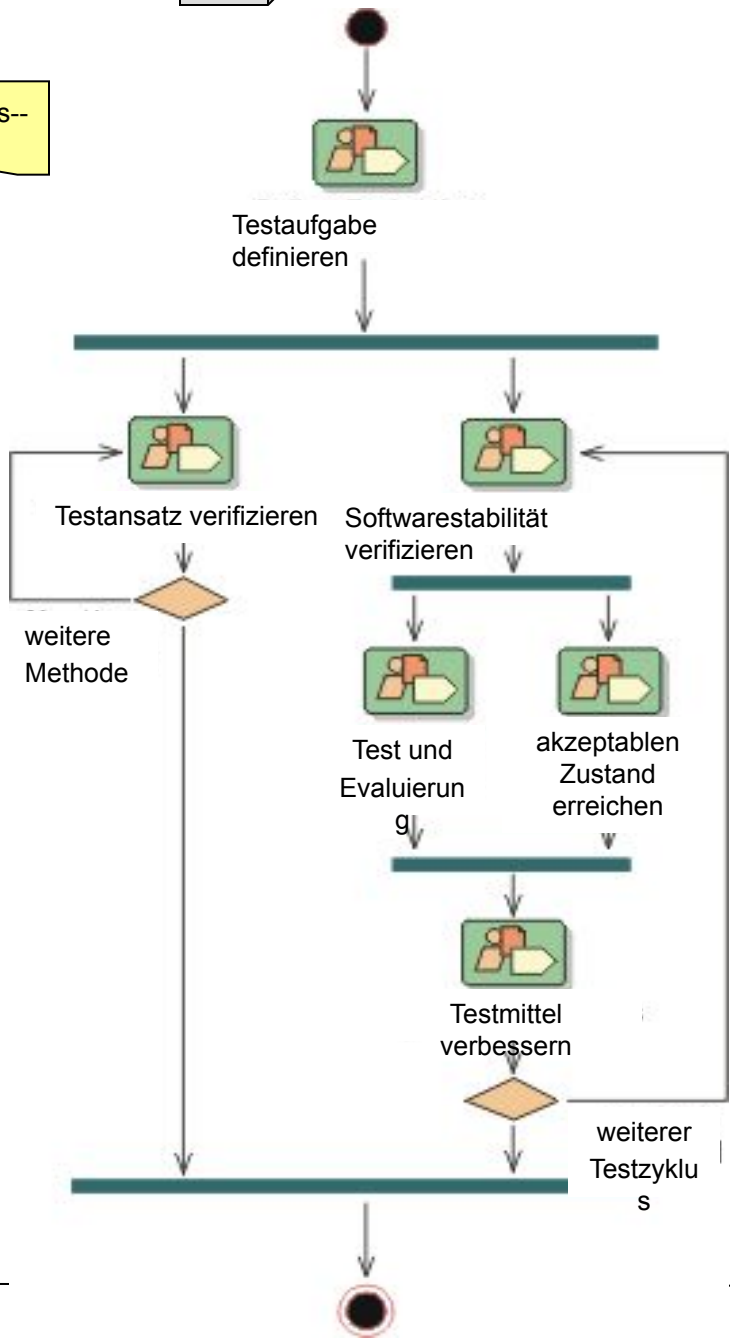
Aufgaben

Anforderungsmodellierung
Anforderunge
Analyse &
Implementierung
Tes
Ausrollen
Konfigurations- und Änderungsmanagement
Projektmanagement
Umgebung

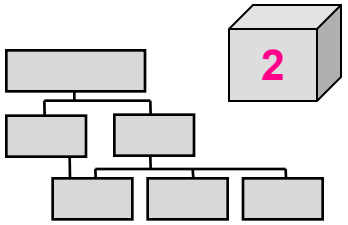


Rational Unified Process
Version
2003.06.01.06

Iterationen



Andere Integrationsstrategien



- Funktionale Integration
  - Bausteine, die eine spezifische Funktion des Systems darstellen, werden integriert.
  - Die Bausteine können eine Mischung sein aus elementaren Komponenten und Komponenten, die weitere Komponenten aufrufen.
  - Individuelle Funktionen können frühzeitig bereitgestellt werden.
  - Gute Strategie für inkrementelle Entwicklungsprozesse (z.B. RAD).

- Ad-Hoc-Integration
  - Bausteine werden in der Reihenfolge der Implementierung integriert
  - Software kann frühzeitig integriert werden
  - Viele Platzhalter und Treiber werden benötigt

- Prozessintegration
  - Ähnlich zu funktionaler Integration.
  - Bausteine werden integriert in der zeitlichen Reihenfolge eines Prozessablaufs (engl. „Thread Integration“)