LEHRSTUHL 14
SOFTWARE ENGINEERING

Vorlesung (WS 2013/14)

Softwarekonstruktion

Prof. Dr. Jan Jürjens

TU Dortmund, Fakultät Informatik, Lehrstuhl XIV

Teil 3.5: Testen im Softwarelebenszyklus

v. 31.01.2014

[Basierend auf dem Foliensatz „Basiswissen Softwaretest - Certified Tester“ des „German Testing Board“ (nach Certified Tester Foundation Level Syllabus, deutschsprachige Ausgabe, Version 2011)]

Literatur:

Andreas Spillner, Tilo Linz: Basiswissen Softwaretest, Aus- und Weiterbildung zum Certified Tester Foundation Level nach ISTQB-Standard. 4.

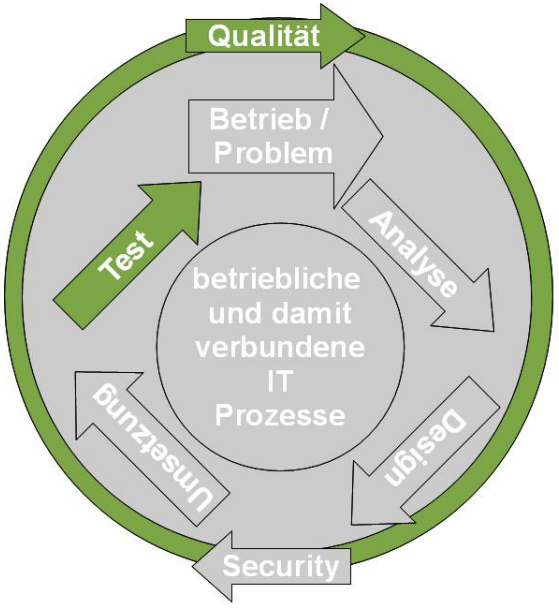
- **Kapitel 3.**

Eike Riedemann: Testmethoden für sequentielle und nebenläufige Software-Systeme. .

- **Kapitel 3, 13.**

Einordnung
Testen im Softwarelebenszyklus

- Modellgetriebene SW-Entwicklung
- Qualitätsmanagement
- Testen
  - Grundlagen Softwareverifikation
  - White-Box-Test
  - Softwaremetriken
  - Algebraische Spezifikationen
  - Black-Box-Test
  - Testen im Softwarelebenszyklus



Inhalte des Testen im Softwarelebenszyklus:

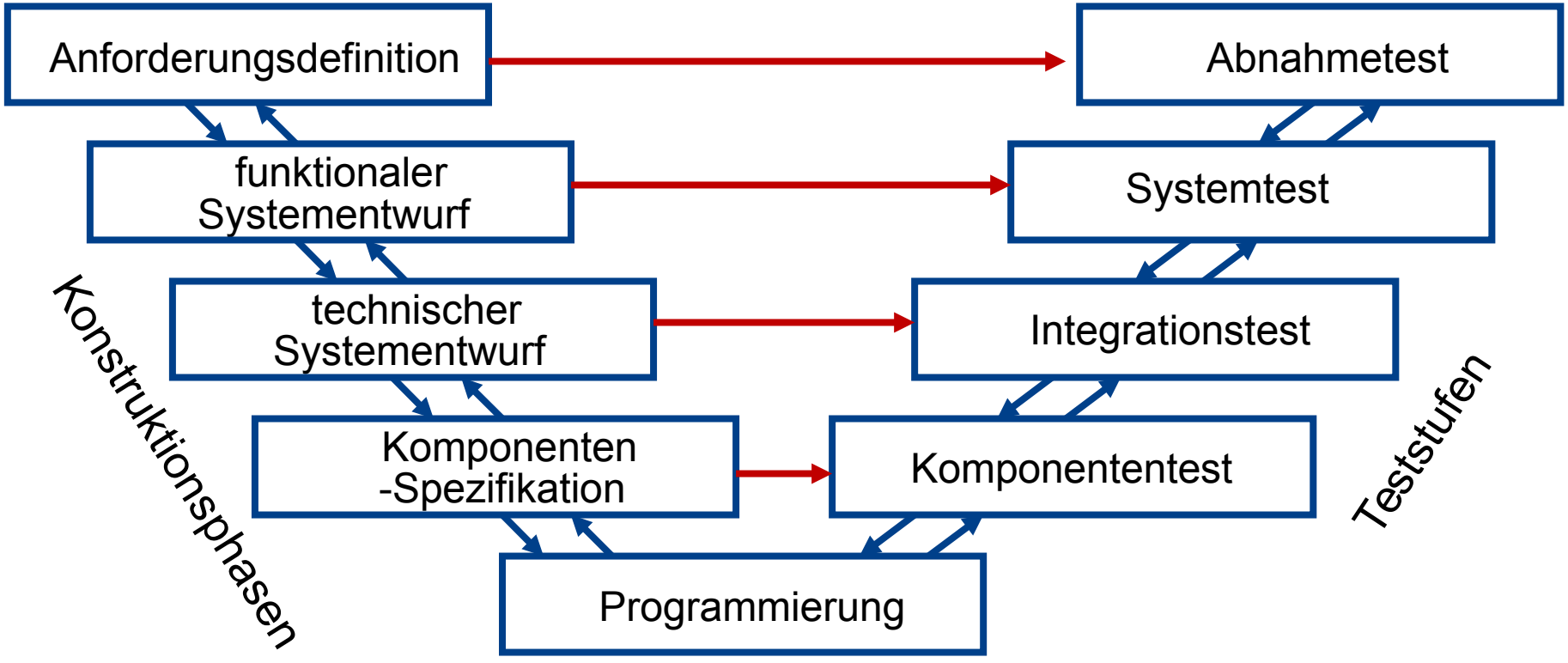
- **Zusammenhang** zwischen Entwicklungs- und Testaktivitäten,
- Komponenten-, Integrations-, System- und Abnahmetest,
- **Testen** von neuen Produktversionen,
- Wartungstests,
- Regressionstests und Auswirkungsanalysen.

3.5 Testen im Software-Lebenszyklus

3.5 Testen im Software-Lebenszyklus

- Testen in Softwareentwicklungsmodellen
- Teststufen (Komponenten-/Integrations-/System-/Abnahmetest)
- Test neuer Produktversionen
- Übersicht über die Testarten

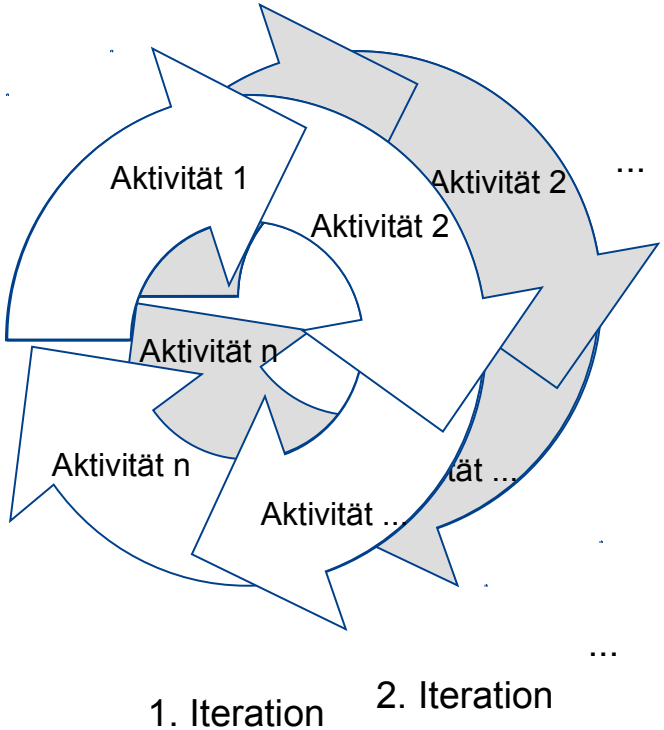
Allgemeines V-Modell
(sequentielles Entwicklungsmodell)



Legende
Testfälle basieren auf den entsprechenden Dokumenten

Iterativ-inkrementelle Entwicklungsmodelle

- Durchlaufung kleiner **Entwicklungsschritte**.
- System wird in einer geplanten Abfolge von **Versionsständen** und **Zwischenlieferungen** (Inkrementen).
- Bei jeder Iteration werden Erweiterungen vorgenommen.
- Jede Iteration ergibt ein wachsendes **unvollständiges System**.



Testen in iterativ-inkrementellen Entwicklungsmodellen

LEHRSTUHL 14
SOFTWARE ENGINEERING

- Testen dem **Entwicklungsablauf** anpassen.
- Für jedes **Inkrement:** Wiederverwendbare Tests.
- Wiederverwendung der vorhandenen Tests in jeder Iteration.
- **Zusätzliche Tests** für neue Funktionalität.
- Jedes Inkrement innerhalb einer Iteration kann verschiedene Teststufen durchlaufen.
- **Integrations-** und **Regressionstests** notwendig.
- Durchführung von Verifizierung und Validierung für jedes Inkrement.

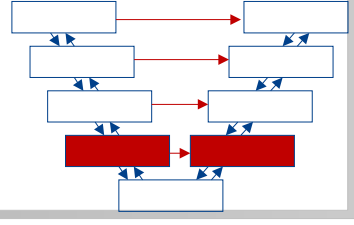
3.5 Testen im Software-Lebenszyklus

3.5 Testen im Software-Lebenszyklus

- Testen in Softwareentwicklungsmodellen
- Teststufen → Komponententest
- Test neuer Produktversionen
- Übersicht über die Testarten

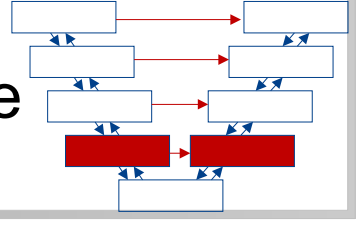
Komponententest
Begriffsklärung

- **Komponententest:**
  - Erstellte Softwarebausteine nach der Programmierphase einem systematischen Test unterziehen.
- Unterschiedliche Bezeichnung der **Softwareeinheiten**:
  - Zum Beispiel als Module, Units oder Klassen (objektorientierte Programmierung).
  - Tests werden Modul-, Unit- bzw. Klassentest genannt.
- Von der verwendeten Programmiersprache abstrahiert: Komponente oder Softwarebaustein.
- Test eines einzelnen **Softwarebausteins:** Komponententest.



Komponententest
Testobjekte

- Isolierte Überprüfung einzelner Softwarebausteine.
  → **Komponentenexterne Einflüsse** beim Test ausschließen.
- Deckt der Test eine Fehlerwirkung auf ?
  → Zuordnung der Ursache der getesteten Komponente.
- Zu testende Komponente: Aus **mehreren Bausteinen** zusammengesetzte Einheit.
- Prüfung von **komponenteninternen Aspekten**.
- **Testbasis**:
  - Spezifikation der Komponente
  - Programmcode
  - Alle anderen Dokumentteile, die sich auf die zu testende Komponente beziehen.



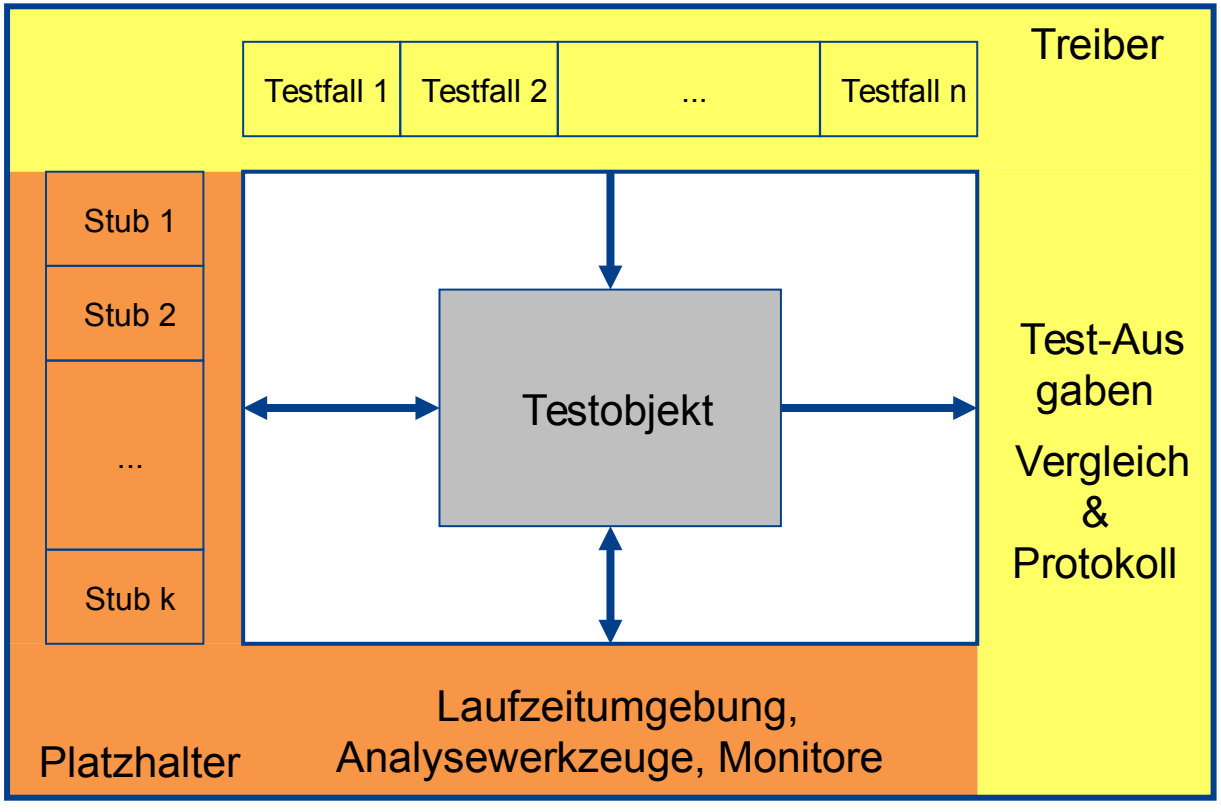
Komponententest
Testumgebung (1)

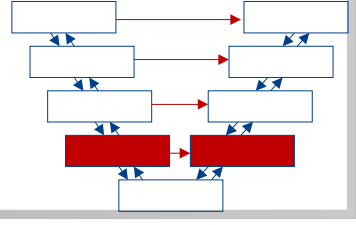
Testumgebung besteht aus

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

  und/oder

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





»Test-first«-Ansatz im Komponententest

Idee:

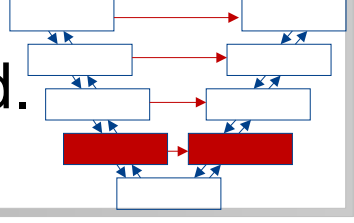
- Testfälle erstellen und Testausführung automatisieren.
- Gewünschte Komponententeile programmieren.

Iterativer Ansatz:

- Programmtext mit Testfällen überprüfen.
- Verbessern bis die Tests keine Fehlerwirkungen mehr aufzeigen.

Testgetriebene Entwicklung (test-driven development):

- Zyklen aus Testfallentwicklung, Programmierung und Integration von kleinen Programmstücken:
- Ausführung von Komponententests bis sie bestanden sind.



Diskussionsfrage

- Wie unterscheiden sich **Testtreiber und Platzhalter?**
- **Antwort:**

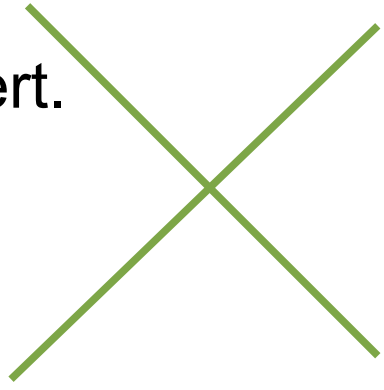
Simulation der Dienste, die das Testobjekt importiert.	**Testtreiber**
Aufruf der Dienste des Testobjektes.	**Platzhalter**

Diskussionsfrage

- Wie unterscheiden sich **Testtreiber und Platzhalter?**
- **Antwort:**

Simulation der Dienste, die das Testobjekt importiert.

Aufruf der Dienste des Testobjektes.



Testtreiber

Platzhalter

3.5 Testen im Software-Lebenszyklus

3.5 Testen im Software-Lebenszyklus

- Testen in Softwareentwicklungsmodellen
- Teststufen → Integrationstest
- Test neuer Produktversionen
- Übersicht über die Testarten

Integrationstest
Begriffsklärung

Voraussetzung:

- Übergebene Testobjekte getestet.
- Aufgezeigte Fehlerzustände möglichst korrigiert.

Integration:

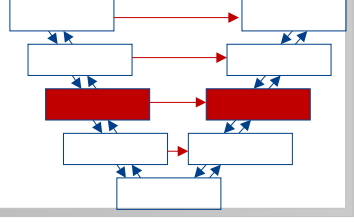
- Verbinden von Gruppen dieser Komponenten zu größeren Teilsystemen durch Tester.

Integrationstest*:*

- Testen der Funktionalität des Zusammenspiels aller Einzelteile miteinander.

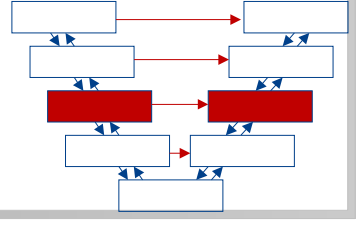
Ziel:

- Fehlerzustände in Schnittstellen und im Zusammenspiel zwischen integrierten Komponenten finden.



Diskussionsfrage

Zu welchen Typen von Fehlerzuständen könnte es Ihrer Meinung nach in der Kommunikation zwischen zwei Komponenten kommen ?



Diskussionsfrage

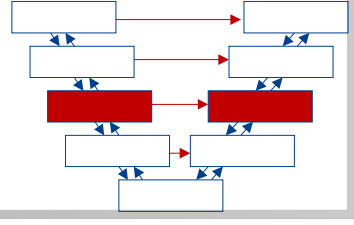
Zu welchen Typen von Fehlerzuständen könnte es Ihrer Meinung nach in der Kommunikation zwischen zwei Komponenten kommen ?

- Komponente übermittelt **keine oder syntaktisch falsche Daten.**
  → Empfangende Komponente kann nicht arbeiten oder stürzt ab.
- Kommunikation funktioniert, aber **unterschiedliche Interpretation** der übergebenen Daten.

  Beispiele:
  - „Several functions inside the OpenSSL library incorrectly check the result after calling the EVP_VerifyFinal function.“ http://www.ocert.org/advisories/ocert-2008-016.html
  - Verlust des Mars Climate Orbiters: "The 'root cause' ... was the failed translation of English units into metric units“ http://www5.in.tum.de/~huckle/bugs.html
- Daten werden richtig übergeben, aber **zum falschen oder verspäteten Zeitpunkt** oder in zu kurzen Zeitintervallen.

→ Diese Fehlerzustände sind im Komponententest nicht findbar.

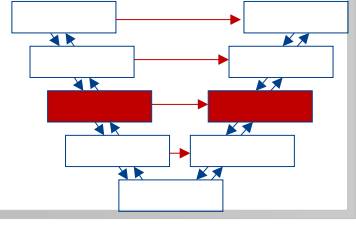
→ Fehlerwirkung äußert sich in der Wechselwirkung zwischen zwei Softwarebausteinen.



Integrationstest - ohne Komponententest ?

LEHRSTUHL 14
SOFTWARE ENGINEERING

Kann auf den Komponententest verzichtet werden, sodass alle Testfälle erst nach erfolgter Integration durchgeführt werden ?



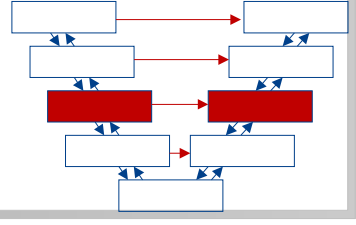
Integrationstest - ohne Komponententest ?

LEHRSTUHL 14
SOFTWARE ENGINEERING

Kann auf den Komponententest verzichtet werden, sodass alle Testfälle erst nach erfolgter Integration durchgeführt werden ?

Das ist möglich und leider eine in der Praxis oft anzutreffende Vorgehensweise.

Welche **Nachteile** könnten damit verbunden sein ?



Integrationstest ohne Komponententest?
Nachteile

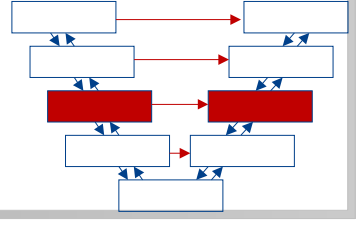
Kann auf den Komponententest verzichtet werden, sodass alle Testfälle erst nach erfolgter Integration durchgeführt werden ?

Das ist möglich und leider eine in der Praxis oft anzutreffende Vorgehensweise.

Welche **Nachteile** könnten damit verbunden sein ?

Gravierende Nachteile:

- Fehlerwirkungen durch funktionale **Fehlerzustände** einzelner Komponenten
  → In nicht geeigneter Testumgebung durchgeführter **impliziter Komponententest** erschwert den Zugang zur Einzelkomponente.
- **Kein geeigneter Zugang** zur Einzelkomponente möglich.
  → Fehlerwirkungen können nicht provoziert und Fehlerzustände nicht gefunden werden.
- Auftretung einer **Fehlerwirkung** oder eines **Ausfall** im Test
  → Eingrenzung seines Entstehungsorts und seiner Ursache: Schwierig oder unmöglich.



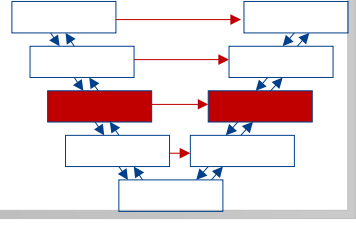
In welcher **Reihenfolge** sind Einzelkomponente zu integrieren, um notwendige Testarbeiten einfach und schnell durchzuführen ?

- Softwarekomponenten: Zu **unterschiedlichen Zeitpunkten** fertig.

  → Entstehen in verschiedenen Projekten.

- Kein Projektmanager oder Testmanager toleriert, dass seine Tester **untätig warten.**

  → Bis Komponenten fertig sind und gemeinsam integriert werden können.



Integrationsstrategie
Inkrementelles Testen

Vorgehen:

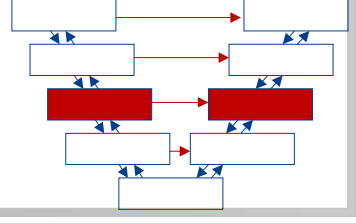
1. Beginn mit einem Modul X des Systems. X ist das bisherige Teilsystem. Rest wird durch Treiber bzw. Platzhalter ersetzt.
2. Füge Modul M zu bisherigem Teilsystem hinzu, wenn für M gilt:
   - M benutzt keine anderen Module oder
   - ein von M benutztes Modul gehört zum bisherigen Teilsystem oder
   - M wird von einem Modul benutzt, das zum bisherigen Teilsystem gehört
   - Rest wird durch Treiber bzw. Platzhalter ersetzt.
3. Wiederhole Schritt 2 bis das „Teilsystem“ das ganze System ist.

Vorteile

- Schnittstellenfehler werden bei jedem Dazubinden entdeckt.
- Fehlerlokalisierung wird vereinfacht.
- Zuerst ausgewählte Module werden gründlich getestet.

Nachteile

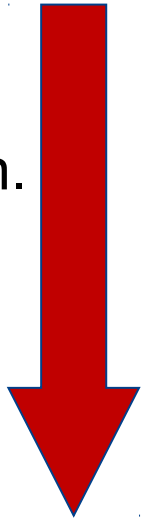
- Testaufwand höher als beim nicht-inkrementellen Testen.
- Bei Schritten 2 und 3 kann wenig parallel gearbeitet werden.



Inkrementelle Integrationsstrategie
Top-down-Integration

Top-down-Integration:

- Test beginnt mit der Komponente des Systems, die weitere Komponenten aufruft, aber selbst nicht aufgerufen wird.
- Untergeordnete Komponenten: Durch Platzhalter ersetzt.
- Sukzessive Integration der Komponenten niedrigerer Systemschichten.
- Getestete höhere Schicht: Treiber.



Vorteil:

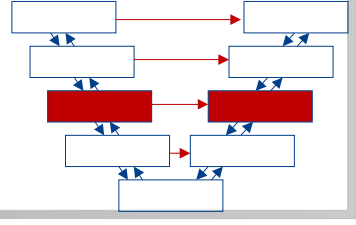
- Keine bzw. nur einfache Test-Treiber benötigt.

  → Übergeordnete, bereits getestete Komponenten bilden den wesentlichen Teil der Ablaufumgebung.

Nachteil:

- Untergeordnete, noch nicht integrierte Komponenten durch Platzhalter ersetzen.

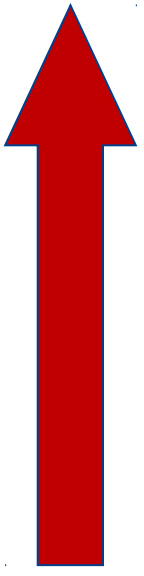
  → Kann aufwändig sein.



Inkrementelle Integrationsstrategie
Bottom-up-Integration

Bottom-up-Integration:

- Beginn mit elementaren Komponenten des Systems.
  → Kein Aufruf weiterer Komponenten.
- Sukzessive Zusammensetzung größerer Teilsysteme aus getesteten Komponenten.
- Test der Integration.

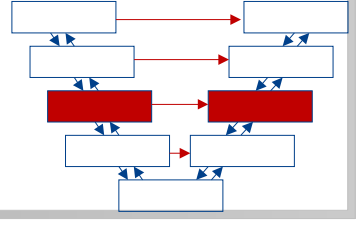


Vorteil:

- Keine Platzhalter benötigt.

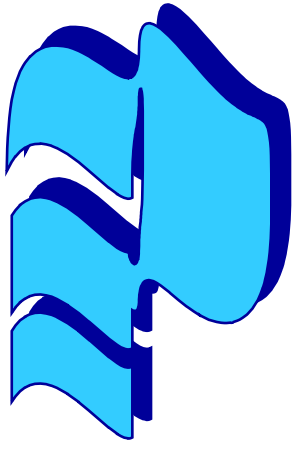
Nachteil:

- Übergeordnete Komponenten durch Test-Treiber simulieren.



Ad-hoc-Integration:

- Integration der Bausteine in der Reihenfolge ihrer Fertigstellung.
- Nach dem Komponententest wird geprüft, ob die Komponente
  - zu einer anderen vorhandenen und getesteten Komponente oder
  - zu einem teilintegrierten Subsystem passt.
- Wenn ja: Beide Teile integrieren und Integrationstest durchführen.

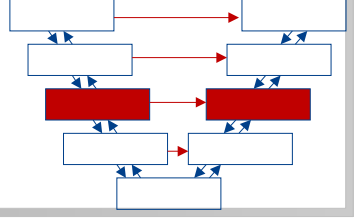


Vorteil:

- Frühe Integration jedes Bausteines in seine passende Umgebung. → Zeitgewinn

Nachteil:

- Notwendigkeit von Platzhalter und Treiber.

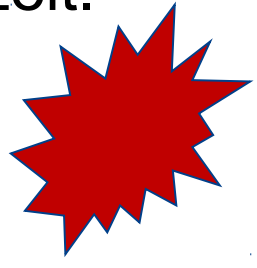


Nicht inkrementelle Integration – *big-bang*-Integration:

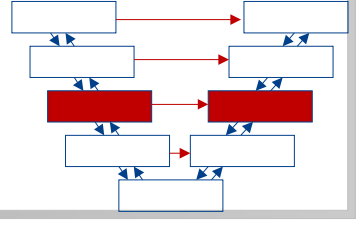
- Nachdem alle Softwarebauteile entwickelt und getestet sind, wird alles auf einmal zusammengeworfen.
- **Im schlimmsten Fall**: Verzicht auf vorgelagerte Komponententests.

Nachteile:

- **Wartezeit** bis zum *big-bang:* *V*erlorene Testdurchführungszeit.
- Testen leidet unter **Zeitmangel**.
  → Kein Testtag verschenken.

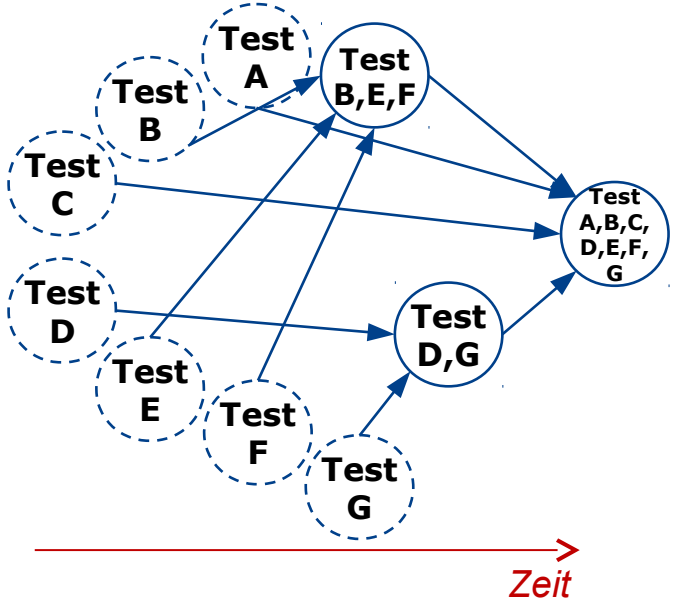


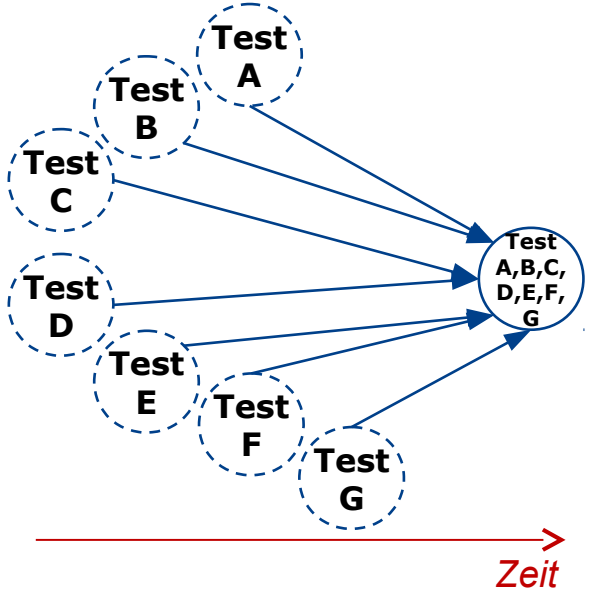
- **Fehlerwirkungen** treten geballt auf.
  → System zum Laufen zu bringen wird schwierig oder unmöglich.
- **Lokalisierung und Behebung** von Fehlerzuständen:
  Schwierig und zeitraubend.

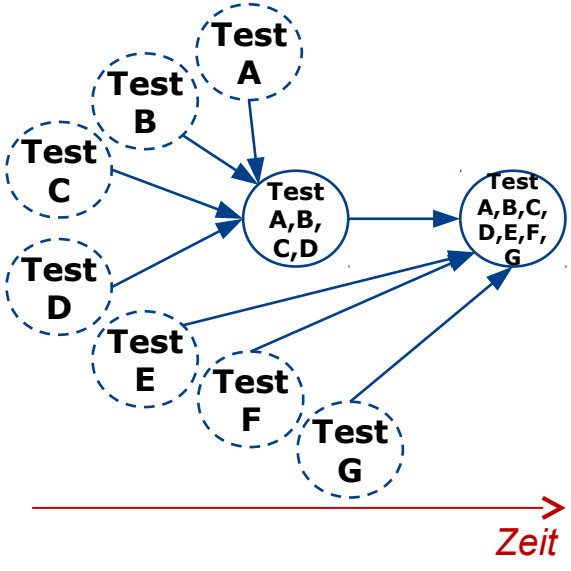


Integrationsstrategien am Beispiel

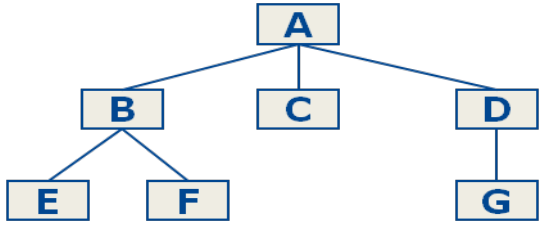
Welche Strategie liegt jeweils vor ?
(big-bang, bottom-up, top-down)





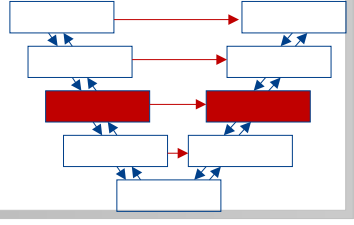


Beispielhierarchie:



Komponententest

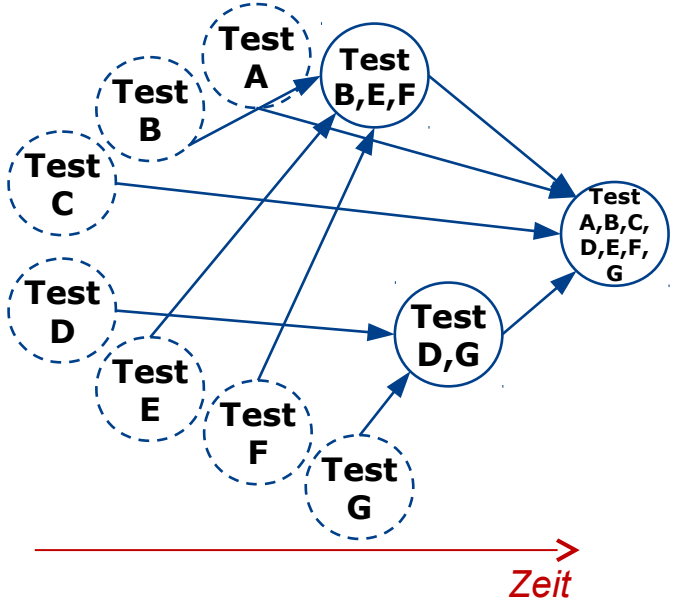
Integrationstest

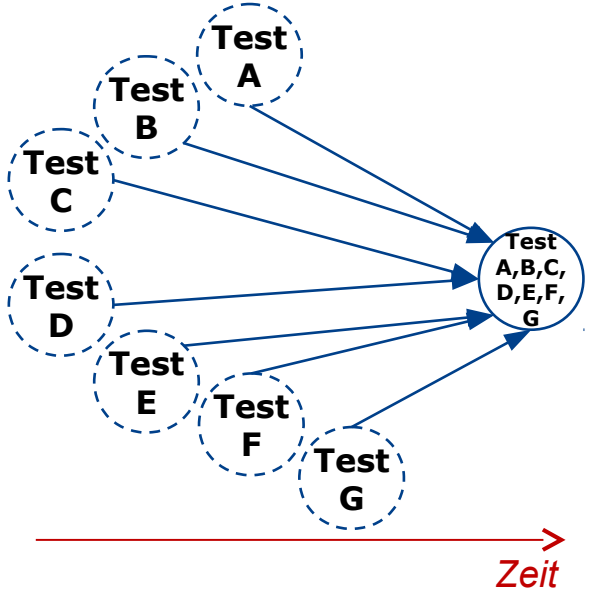


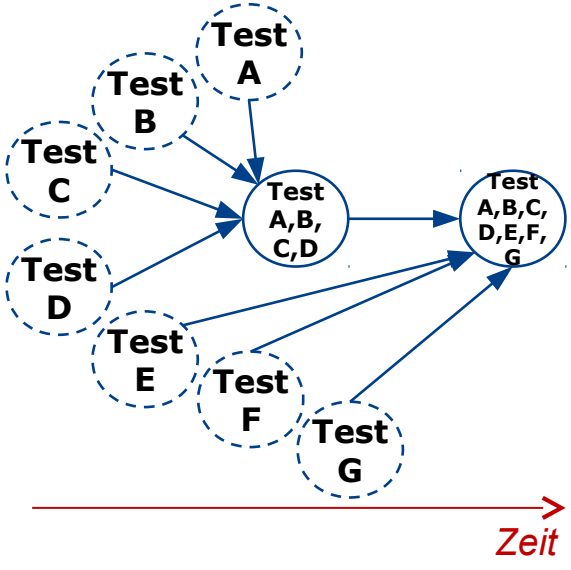
Integrationsstrategien am Beispiel
Bottom-up Integration

Bottom-up Integration: Integrationstest eines Teilsystems, sobald Komponententests aller enthaltenen Knoten vorliegen.

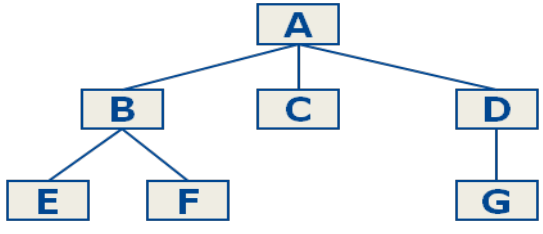
Bottom-up Integration:

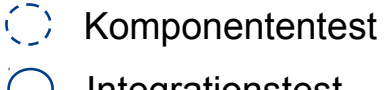


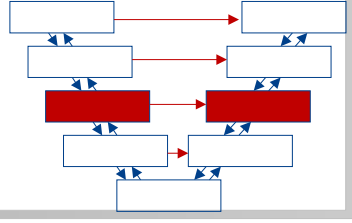




Beispielhierarchie:



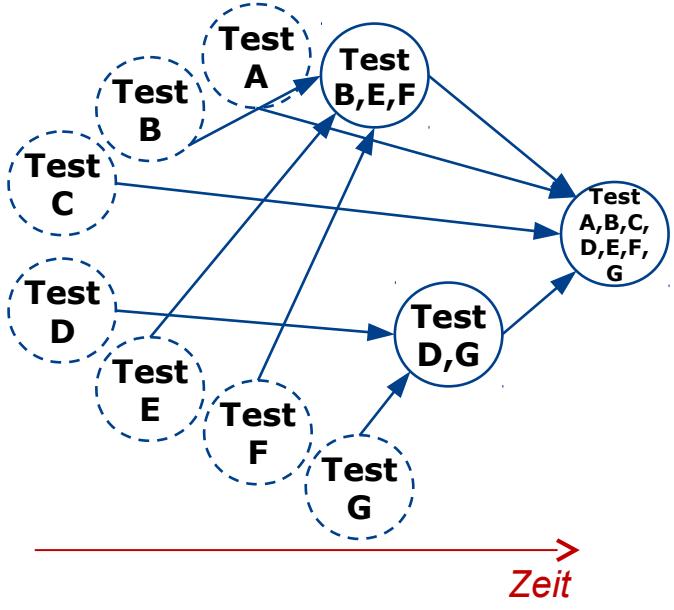




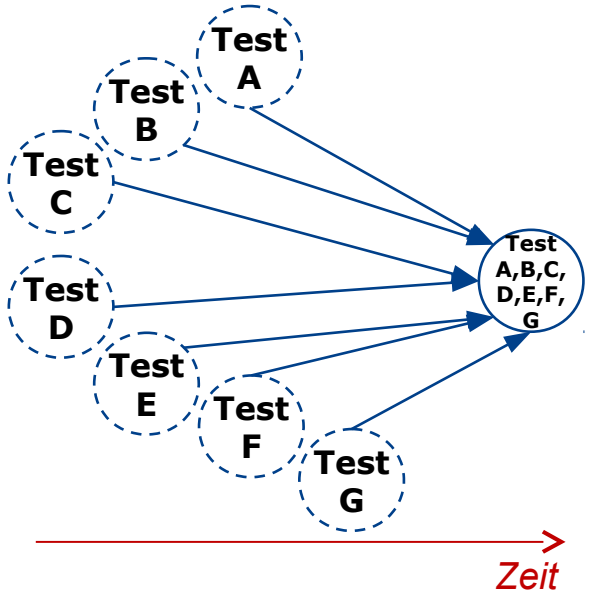
Integrationsstrategien am Beispiel
Big-Bang Integration

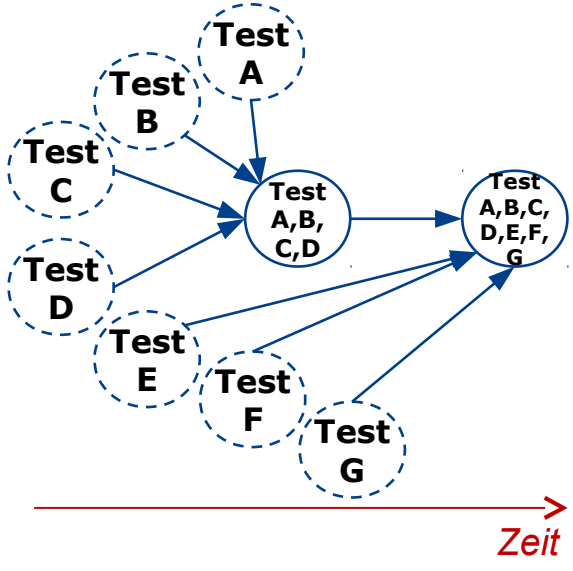
Big-Bang Integration: Integrationstest des Gesamtsystems nach Vorliegen aller Komponententests.

Bottom-up Integration:

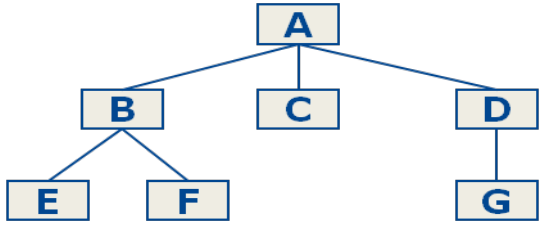


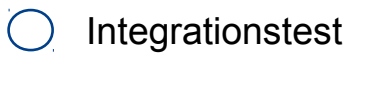
Big-Bang Integration:

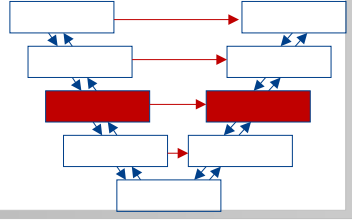




Beispielhierarchie:



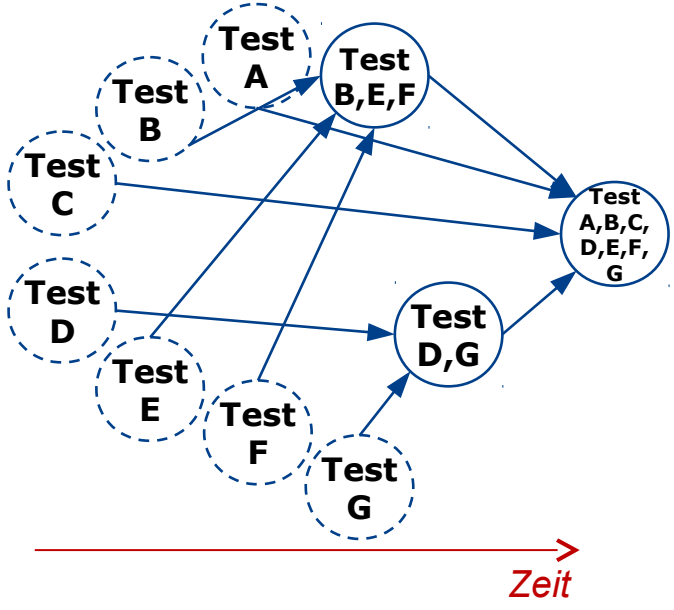




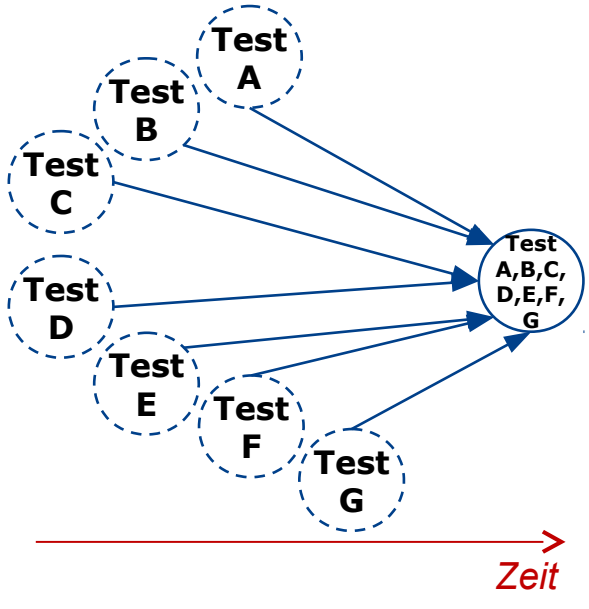
Integrationsstrategien am Beispiel
Top-down Integration

Top-down Integration: Integrationstest eines Teilsystems, sobald Komponententests der direkten Tochterknoten vorliegen.

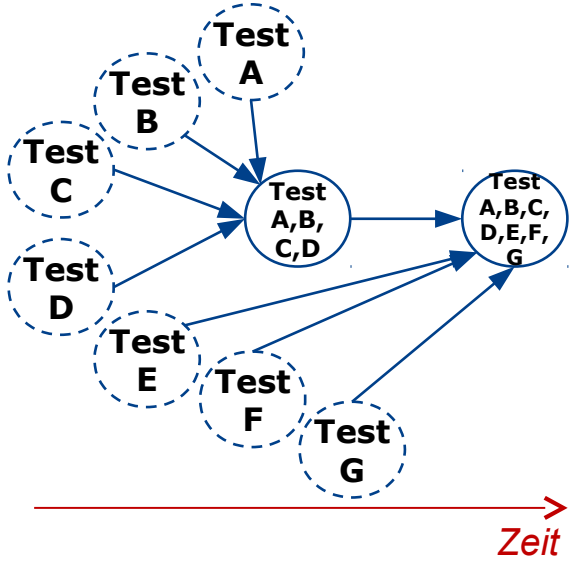
Bottom-up Integration:



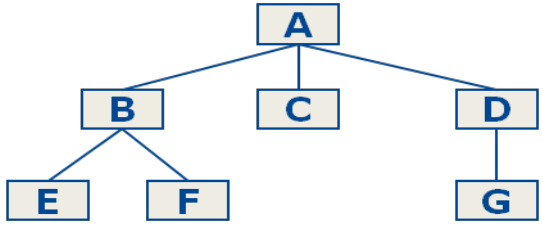
Big-Bang Integration:



Top-down Integration:

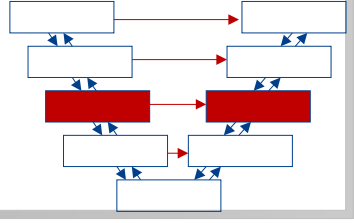


Beispielhierarchie:



Komponententest

Integrationstest



Integrationstest-Werkzeug:
CruiseControl

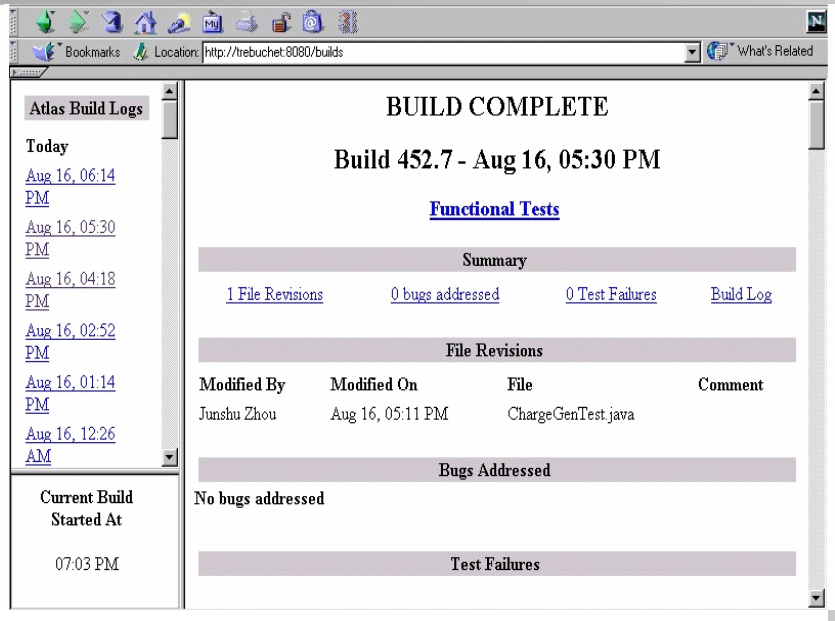
CruiseControl: Open Source Werkzeug zur kontinuierlichen Integration von Systemen (ThoughtWorks).

Kontinuierliche Integration (Prinzip von XP):

- (Mehrmals) tägliches Build der fertigen Systemeinheiten.

Vorteile der kontinuierlichen Integration:

- Fehler werden direkt nach der Integration der neuen / geänderten Komponente mit dem restlichen System identifiziert.
- Falls er nicht lokalisiert werden kann, kann entschieden werden, ob der betreffende Code wieder entfernt wird oder nicht.



Prinzipien:

- Single Source Point (z.B. CVS)
- Automatisiertes Build-Skript (z.B. Ant)
- „Selbst-testender“ Code (z.B. JUnit)

3.5 Testen im Software-Lebenszyklus

- Testen in Softwareentwicklungsmodellen
- Teststufen → Systemtest
- Test neuer Produktversionen
- Übersicht über die Testarten

Überprüft die Erfüllung der spezifizierten Anforderungen vom Produkt:

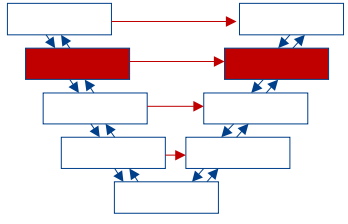
- Systemtest betrachtet das System aus der Perspektive des Kunden.

Testobjekte:

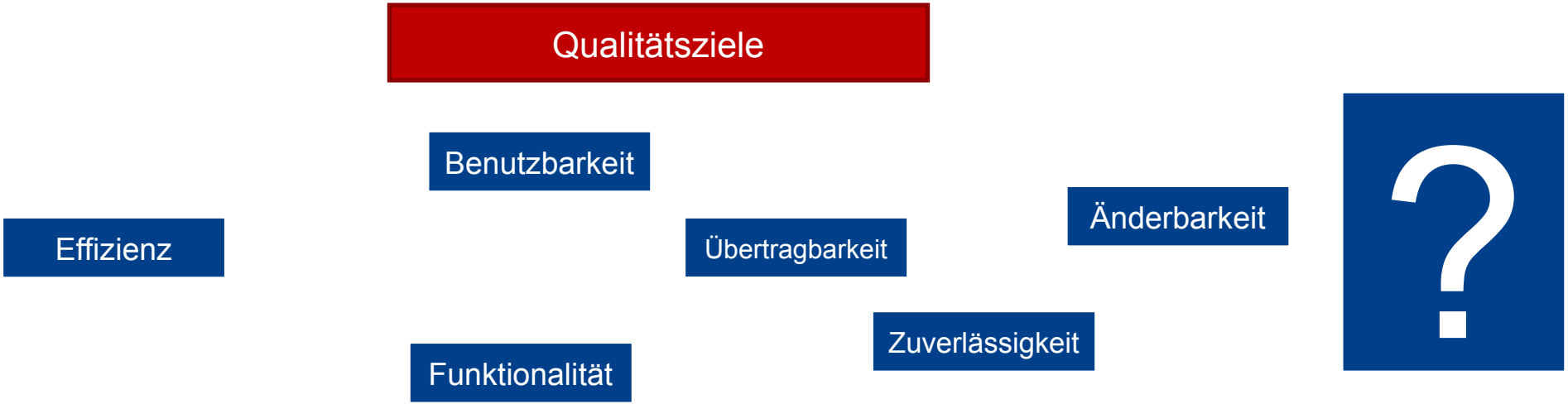
- Mit abgeschlossenem Integrationstest liegt das komplett zusammengebaute Softwaresystem vor.
- Im Systemtest wird dieses System als Ganzes betrachtet.

Testumgebung:

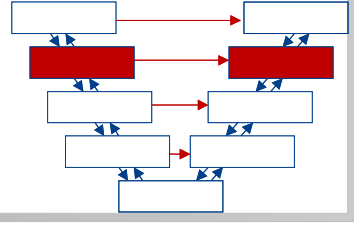
- Der späteren Produktivumgebung nahe kommen.
- Installation der zum Einsatz kommenden Hard- oder Softwareprodukten in der Testumgebung auf allen Ebenen (kein Treiber und Platzhalter)



Systemtest - Teststrategie
Nicht-funktionale Anforderungen



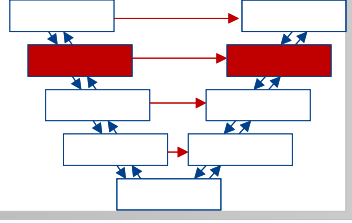
Beispiele:	Beispiele:	Beispiele:	Beispiele:	Beispiele:	Beispiele:
• Funktionaler Test • Test auf (Daten-) Sicherheit • Test auf Daten-kompatibilität / Datenkonver-sion	• Test auf Robustheit • Test auf Wiederher-stellbarkeit	• Benutzbar-keitstest • Prüfung der Benutzer-dokumenta-tion inkl. Online-Hilfe	• Lasttest • Performanz-test • Volumentest • Stresstest	• Prüfung auf Wartbarkeit (Code-Review, Architektur-Review) • Prüfung der Dokumentation • Statische Analyse	• Test unterschied-lic her Konfigur-ationen



Systemtest - Teststrategie
Nicht-funktionale Anforderungen

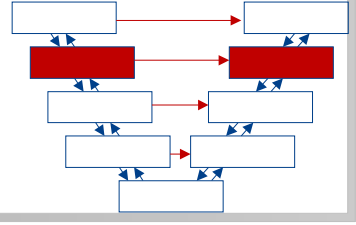
Qualitätsziele

Funktionalität	Zuverlässigkeit	Benutzbarkeit	Effizienz	Änderbarkeit	Übertragbarkeit
• Angemessenheit • Richtigkeit • Interoperabilität • Sicherheit • Ordnungsmäßigkeit	• Reife • Fehlertoleranz • Wiederherstellbarkeit	• Verständlichkeit • Erlernbarkeit • Bedienbarkeit • Attraktivität	• Zeitverhalten • Verbrauchsverhalten	• Analysierbarkeit • Modifizierbarkeit • Stabilität • Testbarkeit	• Anpassbarkeit • Installierbarkeit • Koexistenz • Austauschbarkeit
Beispiele: • Funktionaler Test • Test auf (Daten-) Sicherheit • Test auf Datenkompatibilität / Datenkonversion	Beispiele: • Test auf Robustheit • Test auf Wiederherstellbarkeit	Beispiele: • Benutzbarkeitstest • Prüfung der Benutzerdokumentation inkl. Online-Hilfe	Beispiele: • Lasttest • Performanztest • Volumentest • Stresstest	Beispiele: • Prüfung auf Wartbarkeit (Code-Review, Architektur-Review) • Prüfung der Dokumentation • Statische Analyse	Beispiele: • Test unterschied-licher Konfigur-ationen



Unklare Kundenanforderungen:

- **Keine Anforderungen**.
  → Jedes Systemverhalten zulässig bzw. nicht bewertbar.
- Anwender oder Kunde hat **Erwartung** von »seinem« Softwaresystem.
  → Existenz von Anforderungen!
- Nur »in den Köpfen« einiger am Projekt beteiligten Personen vorhanden. → **Nicht nachlesbar!**
- Tester muss alle diese Informationen über das gewünschte **Soll-Verhalten** nachträglich zusammentragen.

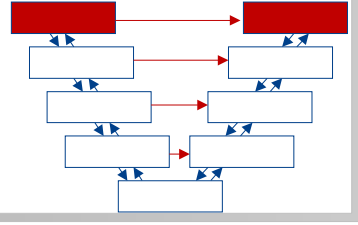


3.5 Testen im Software-Lebenszyklus

- Testen in Softwareentwicklungsmodellen
- **Teststufen → Abnahmetest**
- Test neuer Produktversionen
- Übersicht über die Testarten

Abnahmetest
Begriffsklärung

- **Bis jetzt:** Durchführung der Testarbeiten in Verantwortung des Herstellers.
- Vor Inbetriebnahme der Software: **Abnahmetest**.
- **Sicht** und **Urteil** des Kunden im Vordergrund.
- Abnahmetest zielt nicht auf das Finden von Fehlerzuständen ab.
  → **Vertrauen** in das Produkt gewinnen.
- Abnahmetest:
  - Einziger Test an dem der **Kunde direkt** beteiligt ist.
  - **Spezielle Form** des Systemtests.
  - Beim Kunden durchgeführt.



3.5 Testen im Software-Lebenszyklus

3.5 Testen im Software-Lebenszyklus

- Testen in Softwareentwicklungsmodellen
- Teststufen → Abnahmetest
- Test neuer Produktversionen
- Übersicht über die Testarten

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.	Prüfung, ob die spezifizierten Anforderungen (funktional, nicht-funktional) vom Produkt erfüllt werden.	Vertrauen in das System oder in bestimmte nicht-funktionale Eigenschaften gewinnen.
Testbasis	→Komponenten-spezifikation →Detaillierter Entwurf →Datenmodell →Programmcode	→Software- und →Systementwurf →Architektur →Nutzungsabläufe, Workflows →Anwendungsfälle	→System- und Anforderungsspezifikation →Anwendungsfälle →funktionale Spezifikation →Geschäftsprozesse →Risikoanalyseberichte	→Benutzer-anforderungen →Systemanforderungen →Anwendungsfälle →Geschäftsprozesse →Risikoanalyseberichte
Typische Test-objekte	Isolierte Softwarebausteine (Klasse, Unit, Modul) →Komponenten, Programme →Datenumwandlungs- / Migrationsprogramme →Datenbankmodule	Zu integrierende Einzelbausteine, Subsysteme und zugekaufte Standard-Komponenten →Datenbank-implementierungen →Infrastruktur →Schnittstellen →Systemkonfiguration und Konfigurationsdaten	→System-, Anwender- und Betriebshandbücher →Systemkonfiguration und Konfigurations-daten	→Geschäftsprozesse des integrierten Systems →Betriebs- und Wartungsprozesse →Anwenderverfahren →Formulare →Berichte →Konfigurationsdaten
Testwerk-zeuge	Entwicklungsumgebung, Debugging-Unterstützung Stat. Analysewerkzeuge, Komponententestumgebung	Testmonitore zur Überwachung des Datenaustauschs zwischen Komponenten	Testmanagement-Werkzeuge, GUI-Automatisierungs-werkzeuge	
Testumge-bung	Platzhalter, Treiber, Simulatoren	Wiederverwendung / Erweiterung der Platzhalter, Treiber, Simulatoren aus dem Komponententest	Test- und Produktivumgebung sollten so weit wie möglich übereinstimmen.	Test- und Produktivumgebung sollten so weit wie möglich übereinstimmen.