LEHRSTUHL 14
SOFTWARE ENGINEERING

Vorlesung (WS 2014/15)

Softwarekonstruktion

Prof. Dr. Jan Jürjens

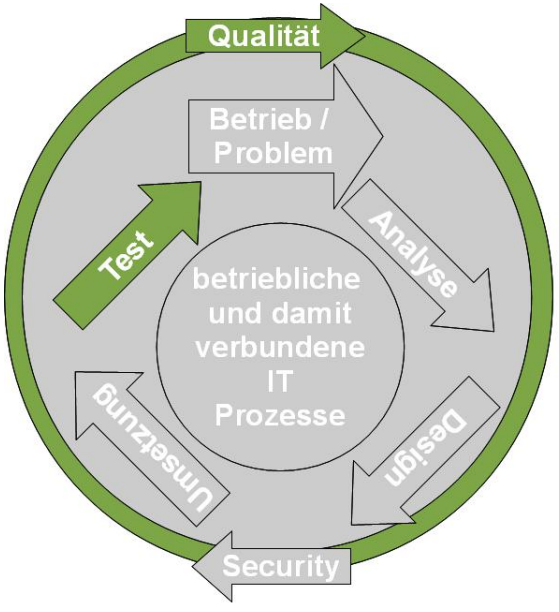
TU Dortmund, Fakultät Informatik, Lehrstuhl XIV

Teil 2.5: Testen im Softwarelebenszyklus

v. 08.02.2015

Einordnung
Testen im Softwarelebenszyklus

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



Basierend auf „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. **Kapitel 3.**
- Eike Riedemann: Testmethoden für sequentielle und nebenläufige Software-Systeme. **Kapitel 3, 13.**

Einleitung Testen im Software-Lebenszyklus

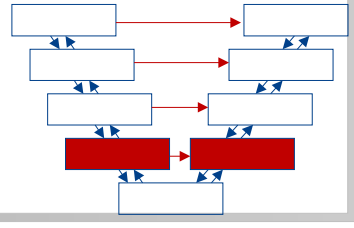
Vorheriger Abschnitt: White Box Test → Analyse innerer Struktur des Testobjekts

Dieser Abschnitt:

- Testen im Softwarelebenszyklus, insbesondere:
- Strategien für Komponenten- und Integrationstests

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.



Integrationstest
Begriffsklärung

Voraussetzung:

- Übergebene Testobjekte getestet.
- Aufgezeigte Fehlerzustände möglichst korrigiert.

Integration:

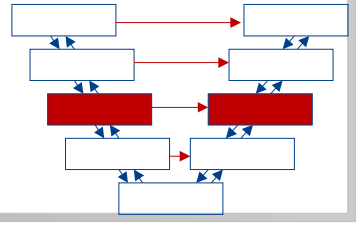
- Verbinden von Gruppen dieser Komponenten zu größeren Teilsystemen durch Tester.

Integrationstest*:*

- Testen der Funktionalität des Zusammenspiels aller Einzelteile miteinander.

Ziel:

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



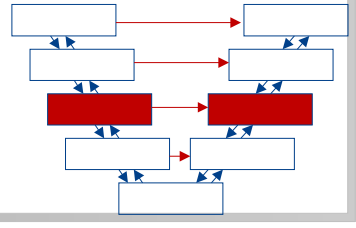
In welcher **Reihenfolge** sind Einzelkomponente zu integrieren, um notwendige Testarbeiten einfach und schnell durchzuführen ?

- 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:

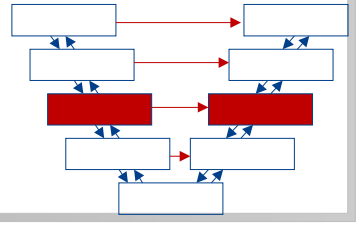
1. Beginn mit einem Modul X des Systems. X ist das bisherige Teilsystem. Rest wird durch Treiber bzw. Platzhalter ersetzt.
2. Füge Modul M zu bisherigem Teilsystem hinzu, wenn für M gilt:
   - M benutzt keine anderen Module oder
   - ein von M benutztes Modul gehört zum bisherigen Teilsystem oder
  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

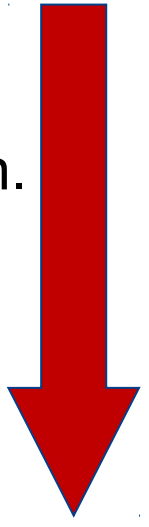
- Testaufwand höher als beim nicht-inkrementellen Testen.
- Bei Schritten 2 und 3 kann wenig parallel gearbeitet werden.



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:

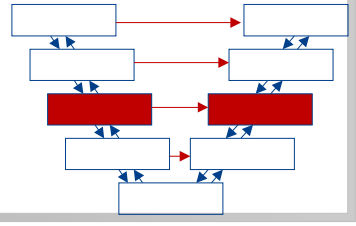
- Keine bzw. nur einfache Test-Treiber benötigt.

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

Nachteil:

- Untergeordnete, noch nicht integrierte Komponenten durch Platzhalter ersetzen.

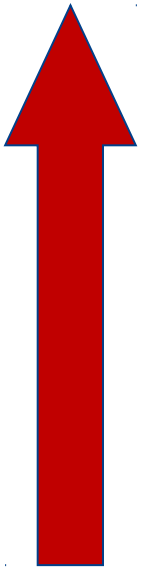
  → Kann aufwändig sein.



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.

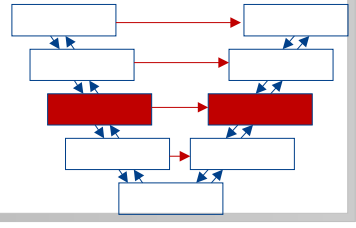


Vorteil:

- Keine Platzhalter benötigt.

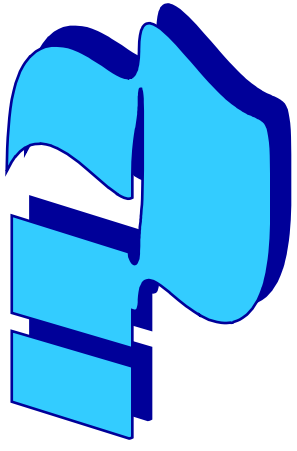
Nachteil:

- 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.

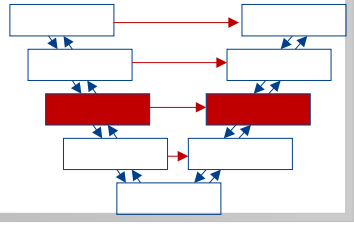


Vorteil:

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

Nachteil:

- Notwendigkeit von Platzhalter und Treiber.

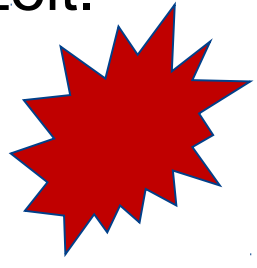


Nicht inkrementelle Integration – *big-bang*-Integration:

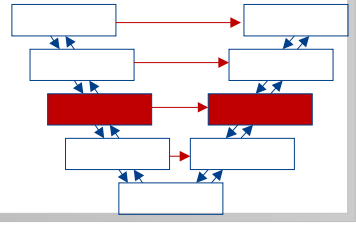
- Nachdem alle Softwarebauteile entwickelt und getestet sind, wird alles auf einmal zusammengeworfen.
- **Im schlimmsten Fall**: Verzicht auf vorgelagerte Komponententests.

Nachteile:

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

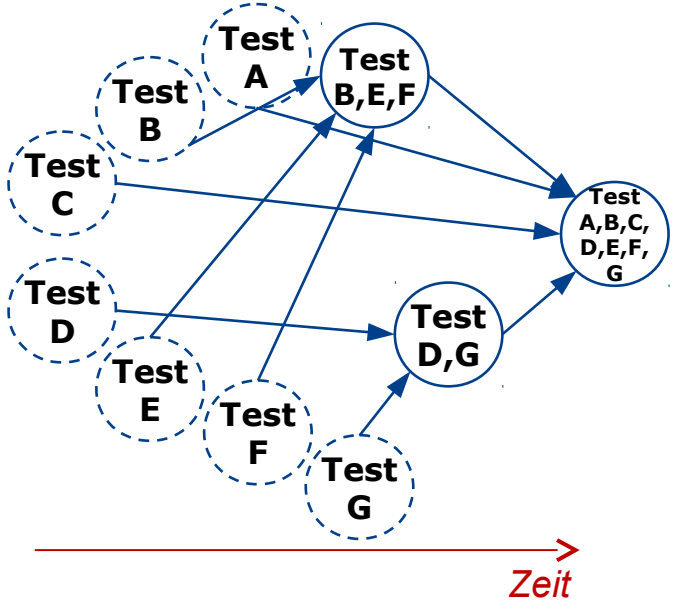


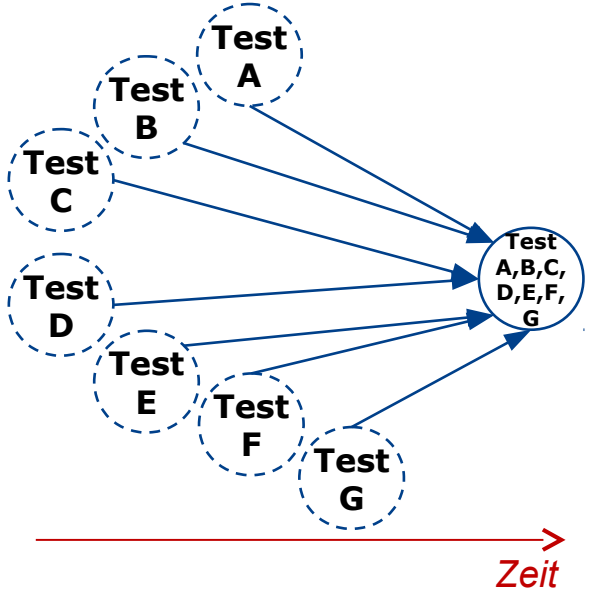
- **Fehlerwirkungen** treten geballt auf.
  → System zum Laufen zu bringen wird schwierig oder unmöglich.
- **Lokalisierung und Behebung** von Fehlerzuständen:
  Schwierig und zeitraubend.

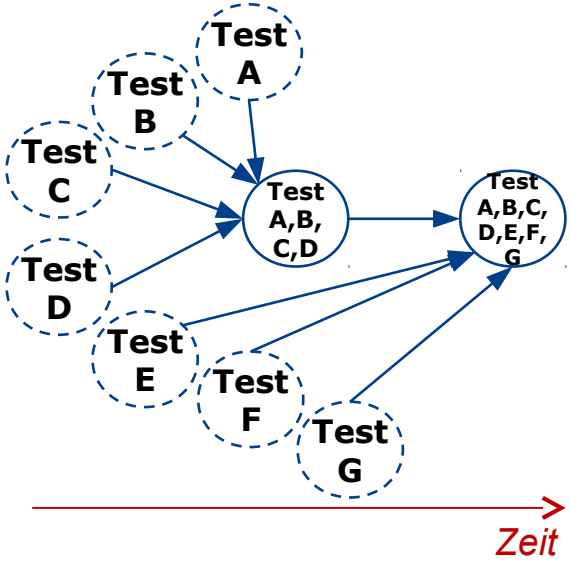


Integrationsstrategien am Beispiel

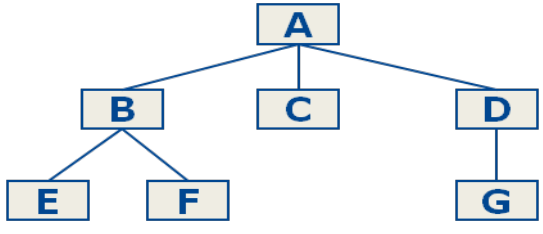
Welche Strategie liegt jeweils vor ?
(big-bang, bottom-up, top-down)





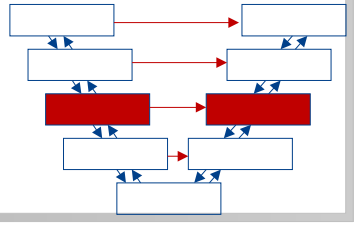


Beispielhierarchie:



Komponententest

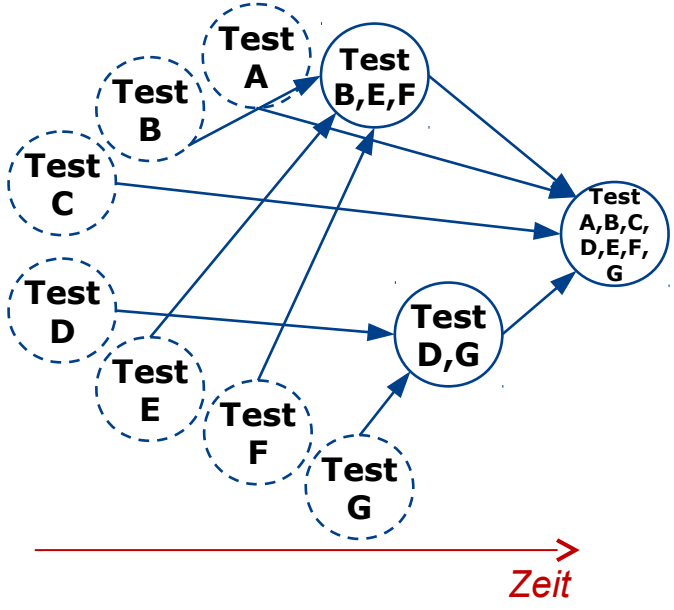
Integrationstest

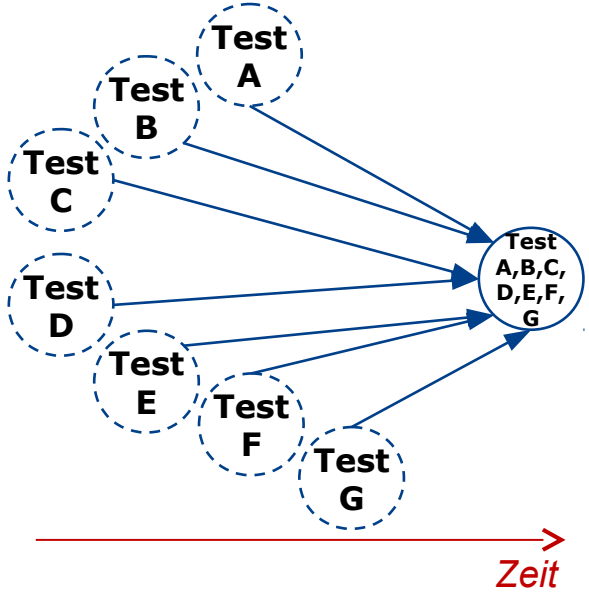


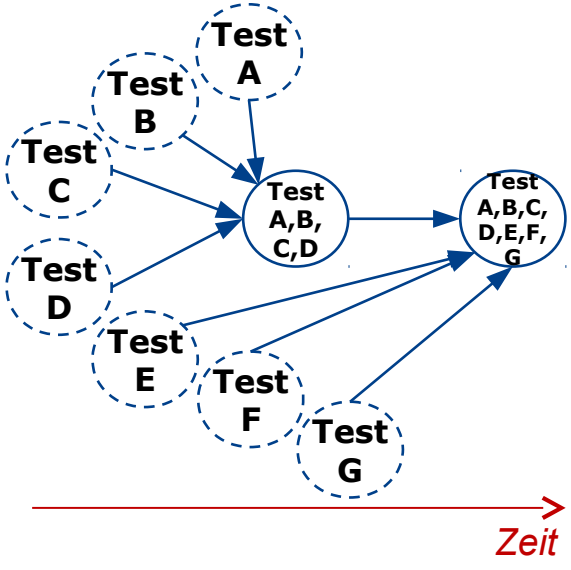
Integrationsstrategien am Beispiel
Bottom-up Integration

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

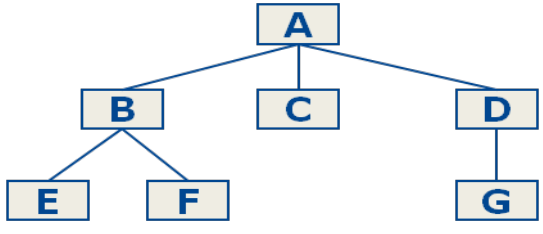
Bottom-up Integration:

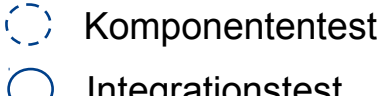


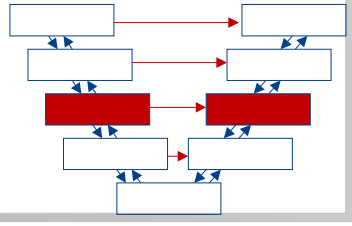




Beispielhierarchie:



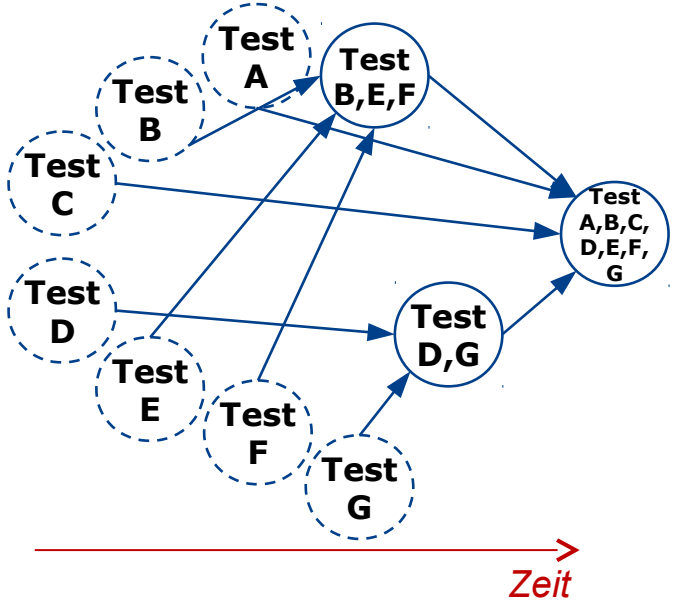




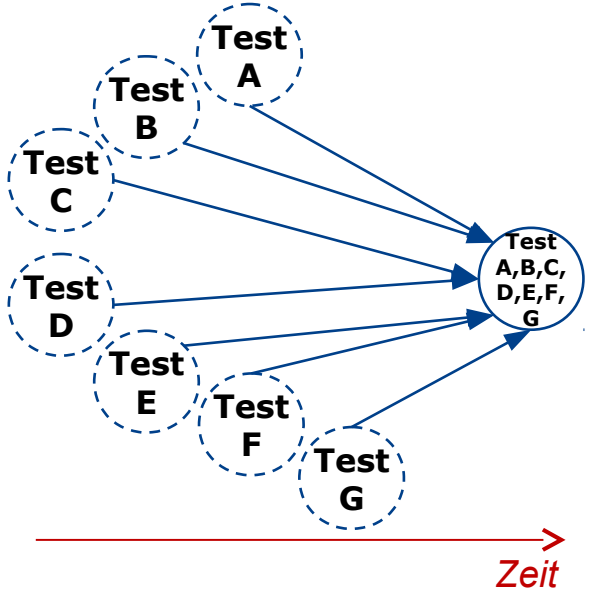
Integrationsstrategien am Beispiel
Big-Bang Integration

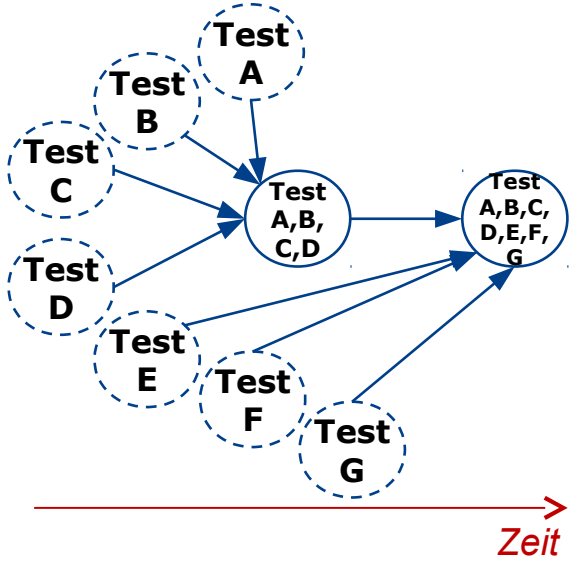
Big-Bang Integration: Integrationstest des Gesamtsystems nach Vorliegen aller Komponententests.

Bottom-up Integration:

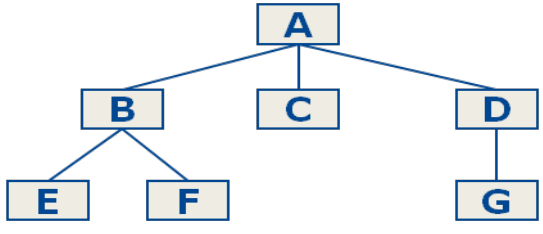


Big-Bang Integration:



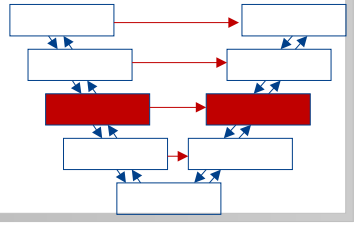


Beispielhierarchie:



Komponententest

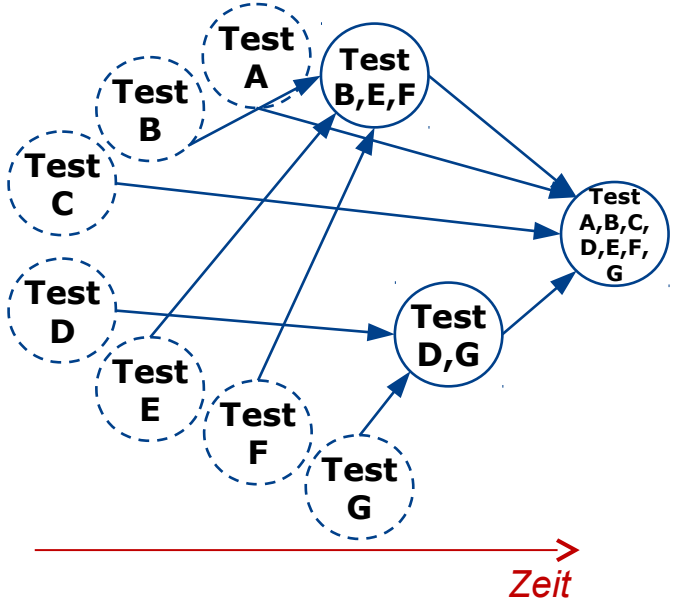
Integrationstest



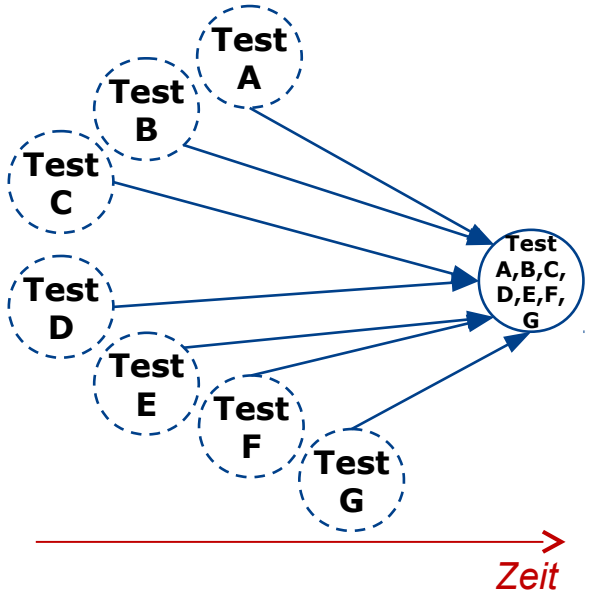
Integrationsstrategien am Beispiel
Top-down Integration

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

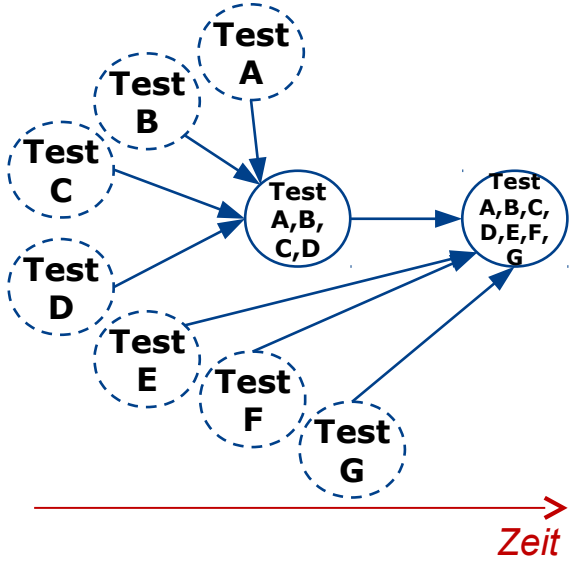
Bottom-up Integration:



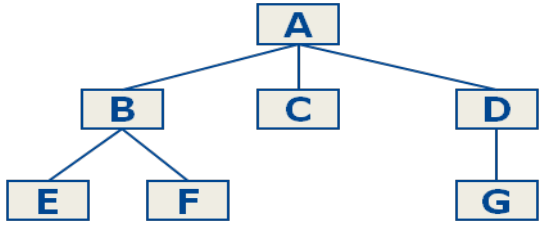
Big-Bang Integration:



Top-down Integration:

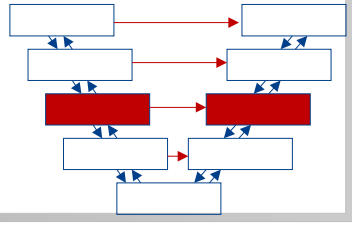


Beispielhierarchie:



Komponententest

Integrationstest



Integrationstest-Werkzeug:
CruiseControl

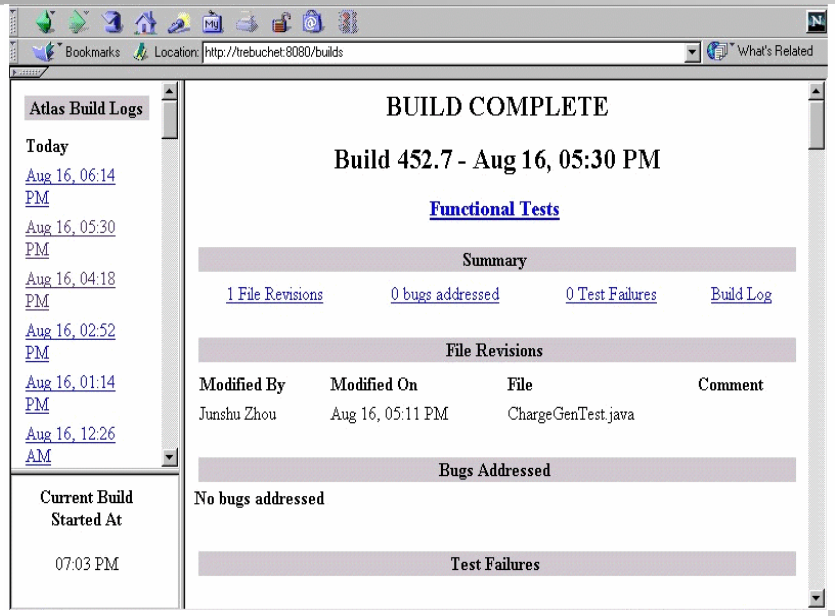
CruiseControl: Open Source Werkzeug zur kontinuierlichen Integration von Systemen (ThoughtWorks).

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)

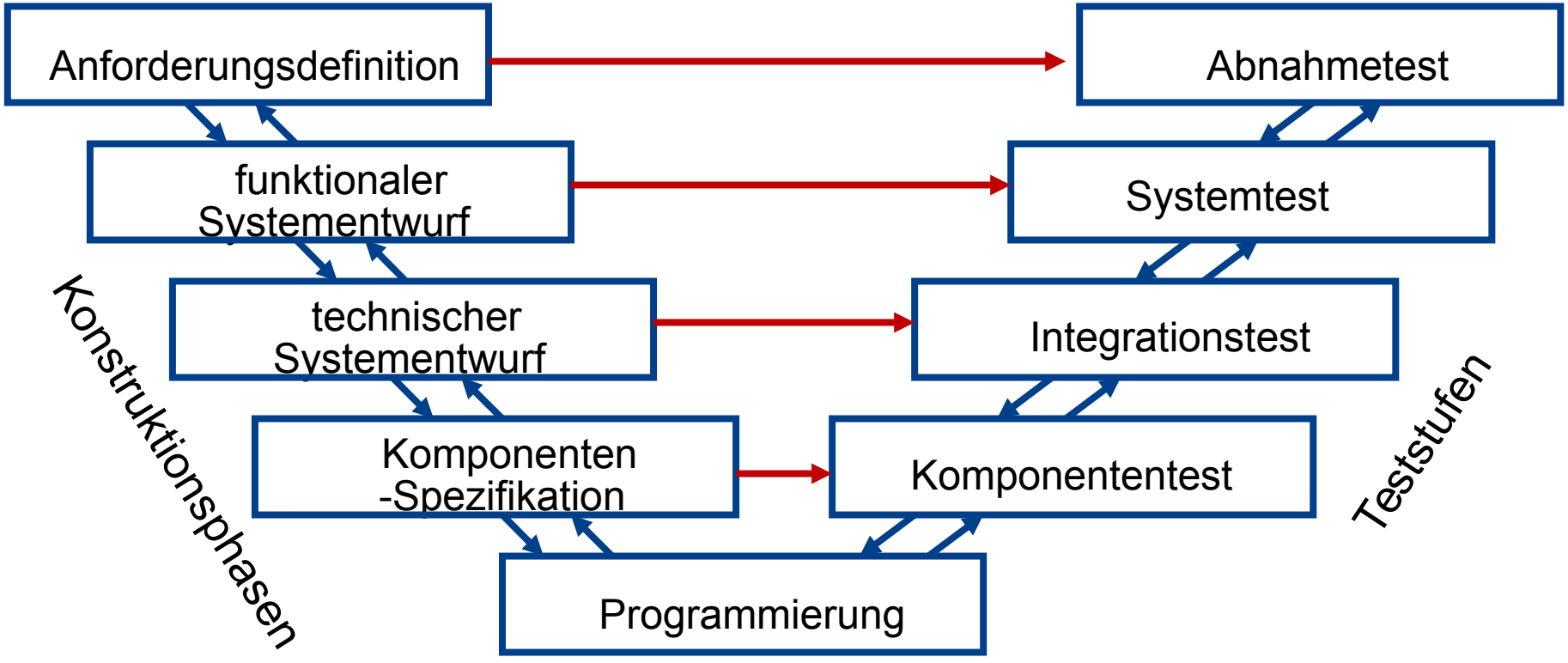
Zusammenfassung: Testen im Software-Lebenszyklus

Dieser Abschnitt:

- Testen im Softwarelebenszyklus, insbesondere:
- Strategien für Komponenten- und Integrationstests

Anhang
(zum selbständigen Nachlesen)

Allgemeines V-Modell
(sequentielles Entwicklungsmodell)



Legende
Testfälle basieren auf den entsprechenden Dokumenten

Integrationstest ohne Komponententest?
Nachteile

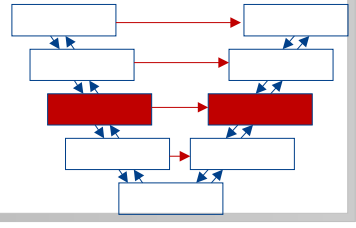
Kann auf den Komponententest verzichtet werden, sodass alle Testfälle erst nach erfolgter Integration durchgeführt werden ?

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.



Systemtest
Begriffsklärung

Überprüft die Erfüllung der spezifizierten Anforderungen vom Produkt:

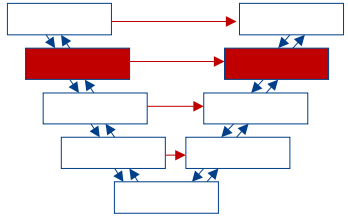
- Systemtest betrachtet das System aus der Perspektive des Kunden.

Testobjekte:

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

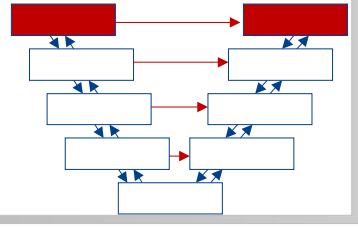
Testumgebung:

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



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.



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.