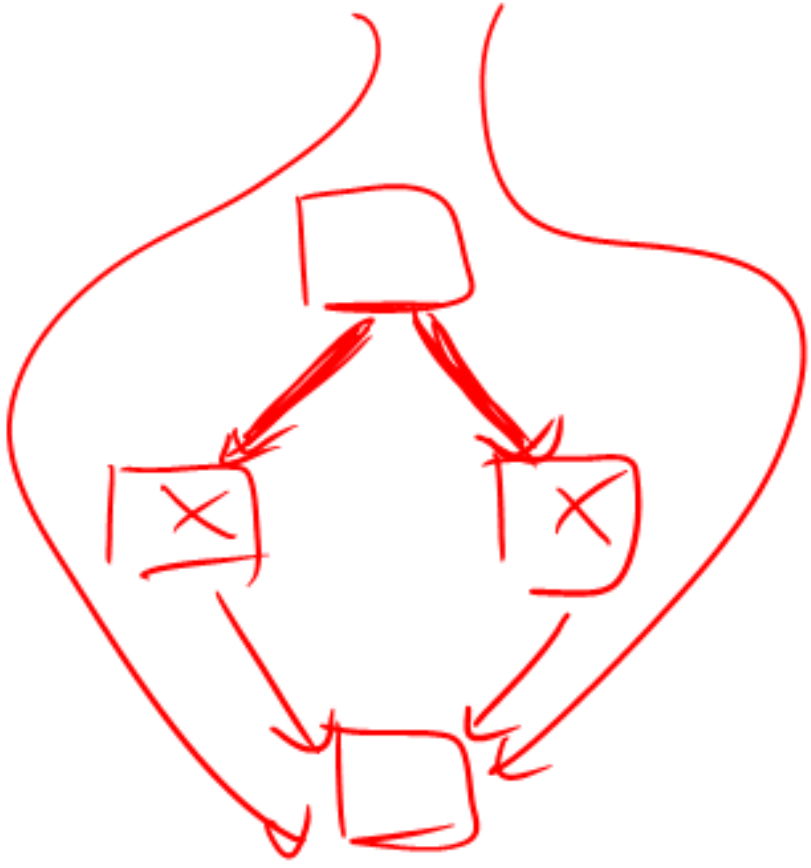
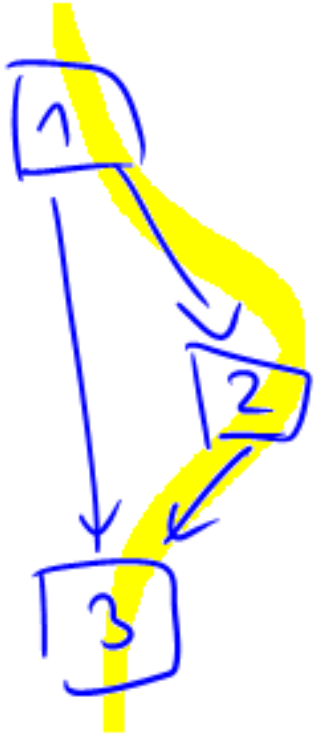
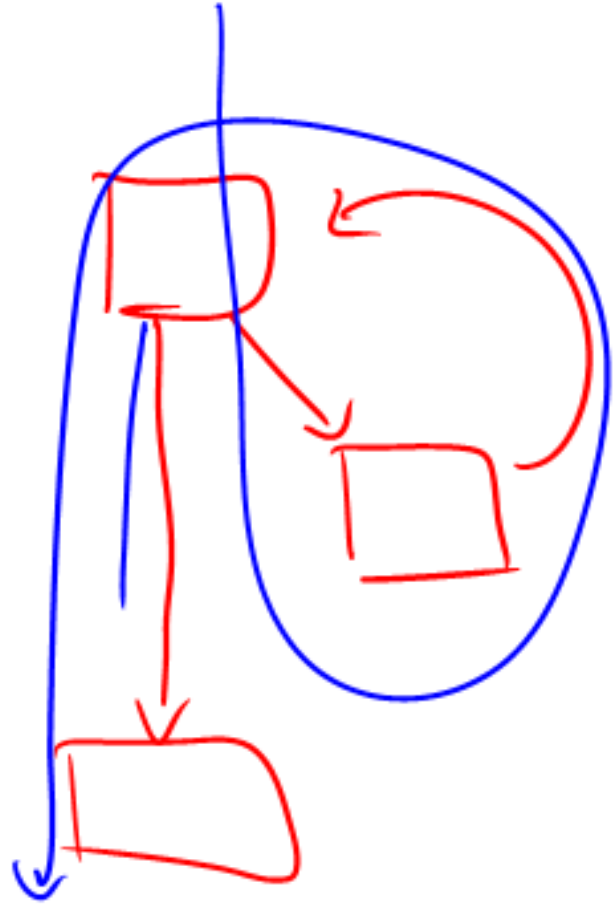





```
if (B) {
    x = 1;
}
put (x);
```



Vergleich von White- und Black-Box-Tests

Black-Box-Tests (Funktionstests) betrachten den Prüfling als schwarze Box. Sie setzen keine Kenntnisse über die Interna voraus.

White-Box-Tests (Strukturtests, Glass-Box-Tests) betrachten Interna des Prüflings, um Testfälle zu entwickeln.

Eigenschaft	Black-Box-Test	White-Box-Test
Test auf Basis von	Schnittstellen-spezifikation	Lösung
Wiederverwendung bei Änderung der Struktur	ja	eingeschränkt
Geeignet für Testart	alle	Komponententest
Finden Fehler aufgrund von	Abweichung von Spez.	eher Kodierfehler

Black- versus White-Box-Tests

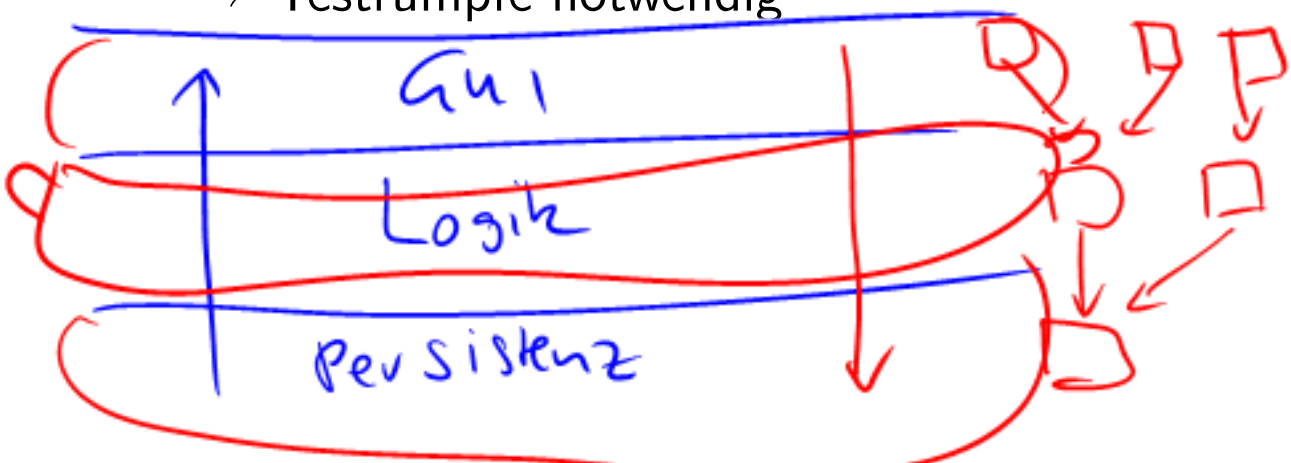
Nicht entweder-oder, sondern sowohl-als-auch!

Komplementäre Verwendung:

1. Erstelle Funktionstest
2. Messe Abdeckung
3. Wenn Abdeckung nicht ausreichend; ergänze durch Strukturtest; zurück zu 2.

Strategien des Integrationstests I

- Urknalltest: alle Komponenten werden einzeln entwickelt und dann in einem Schritt integriert
  - → erschwert Fehlersuche
- Bottom-Up: Integration erfolgt inkrementell in umgekehrter Richtung zur Benutzt-Beziehung
  - → keine Testrümpfe notwendig (aber Testtreiber)
  - → Fehler in der obersten Schicht werden sehr spät entdeckt; die enthalten jedoch die Applikationslogik
- Top-Down: Integration erfolgt in Richtung der Benutzt-Beziehung
  - → Fehler in der obersten Schicht werden sehr früh entdeckt
  - → Testrümpfe notwendig



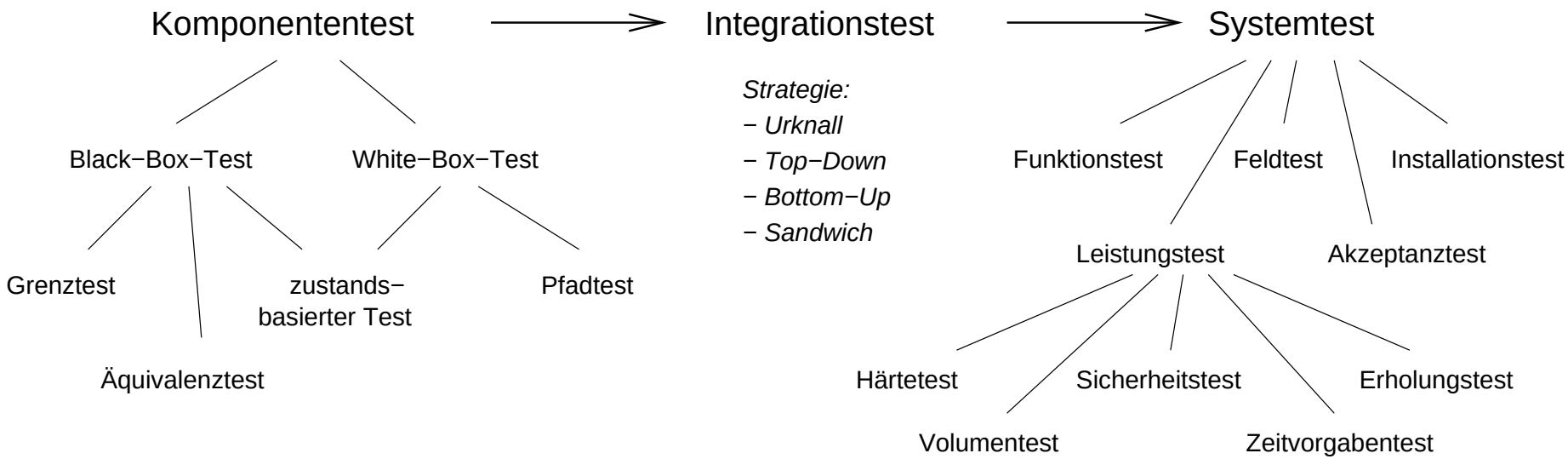
Strategien des Integrationstests II

- Sandwich-Test-Strategie: Kombination von Top-Down und Bottom-Up
  - Identifikation der zu testenden Schicht: Zielschicht
  - zuerst: individuelle Komponententests
  - dann: kombinierte Schichttests:
    - Oberschichttest mit Zielschicht
    - Zielschicht mit Unterschicht
    - alle Schichten zusammen

Leistungstests

- Härtetest: viele gleichzeitige Anfragen
- Volumentest: große Datenmengen
- Sicherheitstests: Sicherheitslücken aufspüren
- Zeitvorgabentests: werden spezifizierte Antwortzeiten eingehalten?
- Erholungstests: Tests auf Erholung von fehlerhaften Zuständen

Zusammenfassung der Testarten



Testdokumentation: Aufzeichnung der Testaktivitäten

- Testplan: Projektplan fürs Testen
- Testfallspezifikation: Dokumentation eines jeden Testfalls
- Testvorfallbericht (Testprotokoll): Ergebnisse des Tests und Unterschiede zu erwarteten Ergebnissen
- Testübersicht: Auflistung aller Fehler (entdeckt und noch zu untersuchen)
  $\rightarrow$ Analyse und Priorisierung aller Fehler und deren Korrekturen

Testplan und Testfallspezifikation können sehr früh schon erstellt werden.

Testplan (IEEE Std. 829-1998 1998)

1. Einführung
2. Beziehung zu anderen Dokumenten
3. Systemüberblick
4. Merkmale, die getestet/nicht getestet werden müssen
5. Abnahmekriterien
6. Vorgehensweise
7. Aufhebung und Wiederaufnahme
8. Zu prüfendes Material (Hardware-/Softwareanforderungen)
9. Testfälle
10. Testzeitplan: Verantwortlichkeiten, Personalausstattung und Weiterbildungsbelange, Risiken und Schadensmöglichkeiten, Zeitplan

Testfallspezifikation

- Testfallbezeichner: eindeutiger Name des Testfalls; am Besten Namenskonventionen benutzen
- Testobjekte: Komponenten (und deren Merkmale), die getestet werden sollen
- Eingabespezifikationen: erwartete Eingaben
- Ausgabespezifikationen: erwartete Ausgaben
- Umgebungserfordernisse: notwendige Software- und Hardware-Plattform sowie Testrümpfe und -stümpfe
- Besondere prozedurale Anforderungen: Einschränkungen wie Zeitvorgaben, Belastung oder Eingreifen durch den Operateur
- Abhängigkeiten zwischen Testfällen

Testschadensbericht

- welche Merkmale wurden getestet?
- wurden sie erfüllt?
- bei Störfällen: wie können sie reproduziert werden?

$\rightarrow$ Sammlung und Priorisierung in der Testübersicht
$\rightarrow$ Test $\neq$ Fehlersuche bzw. -korrektur!

Quizz

Woher kommt der englische Begriff *Bug* für Fehler?

Auflösung: *Bug* wird auf Grace Hopper zurückgeführt; bezeichnet einen Fehler, der auftrat als ein Nachtfalter mit einem Rechnerrelais in Konflikt geriet – mit dem Effekt, dass ein Programm anhielt.

Implementierung

1. Implementierung
   - Motivation
   - Architekturkonformität
   - Bauhaus-Werkzeug zur Reflektionsmethode
   - Programmierrichtlinien
   - Anhang

Lernziele

- Feinentwurf eines Systems durchführen können
- Programmierrichtlinien entwerfen, kennen und einhalten
- Verständnis für Codequalität entwickeln

Feinentwurf

Der Feinentwurf ist die Brücke zwischen abstrakter Architektur und detailliertem Code.

Er legt fest:

- Datenstrukturen
  - → Klassendiagramme mit zusätzlichen Implementierungsdetails; (z.B. Navigierbarkeit, Implementierung von Assoziationen, Behandlung von Mehrfachvererbung oder weiteren Implementierungsklassen etc.)
- Abläufe
  - → Zustandsautomaten
  - → Aktivitätsdiagramme
  - → Sequenzdiagramme