Lehrstuhl für
Software & Systems Engineering

Techniken im Software-Test

München, 4. Juli 2000

Heiko Lötzbeyer

Institut für Informatik
Lehrstuhl für Software & Systems Engineering
Technische Universität München

Inhalt

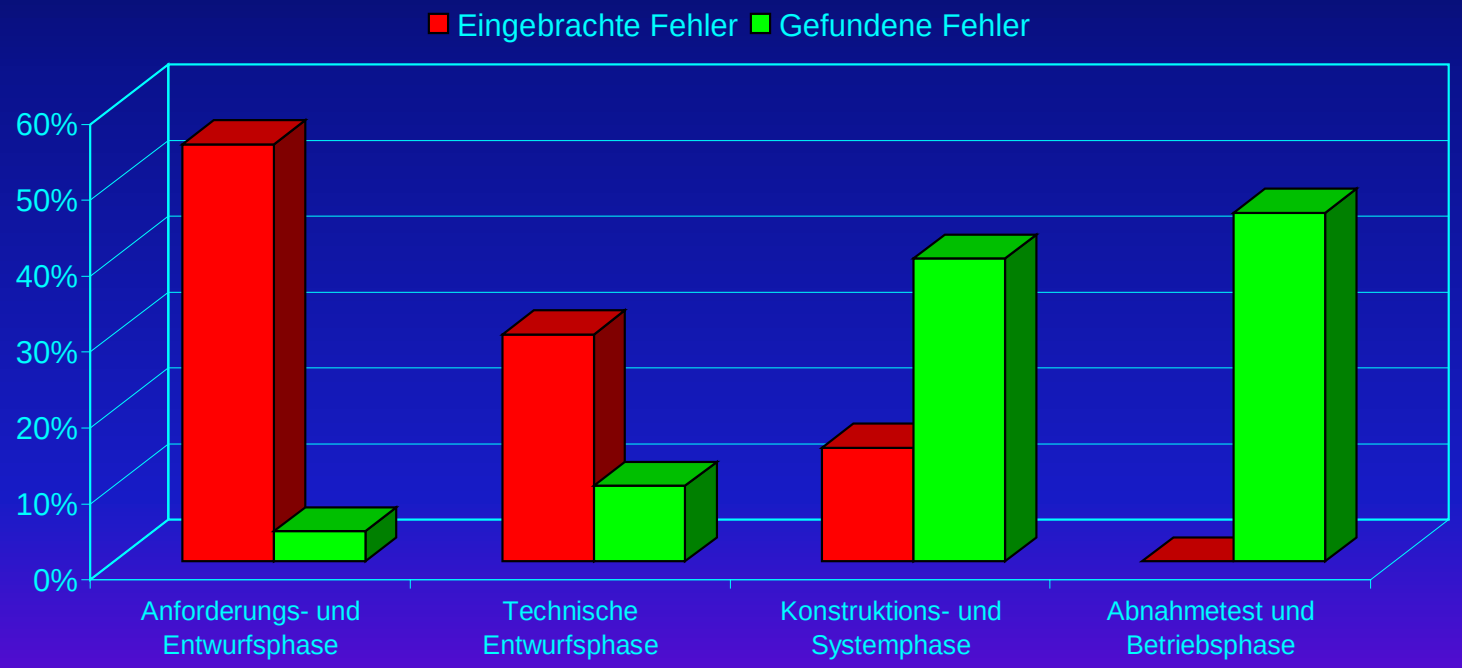
- Ziele des Software Tests
- Überblick
- Teststufen
  - Unit-Test
  - Integrationstest
  - Systemtest
- Testverfahren
  - manuelle Testverfahren
  - Whitebox
  - Blackbox
- Testfallermittlung
  - Spezifikationsbasierte Testfallermittlung
- Schluß

Ziele

- Was ist Testen?
  - G. J. Myers, ´79:
    „Testen ist der Prozeß, ein Programm mit der Absicht auszuführen, Fehler zu finden.“
  - Hetzel ´83:
    „Messung der ***Softwarequalität***“
  - Die ***Überprüfung***, daß ein System die spezifizierten Anforderungen erfüllt.
- Ziele:
  - Fehler finden
  - auch: Fehler vermeiden (B. Beizer)
  - nicht: Korrektheitsnachweis

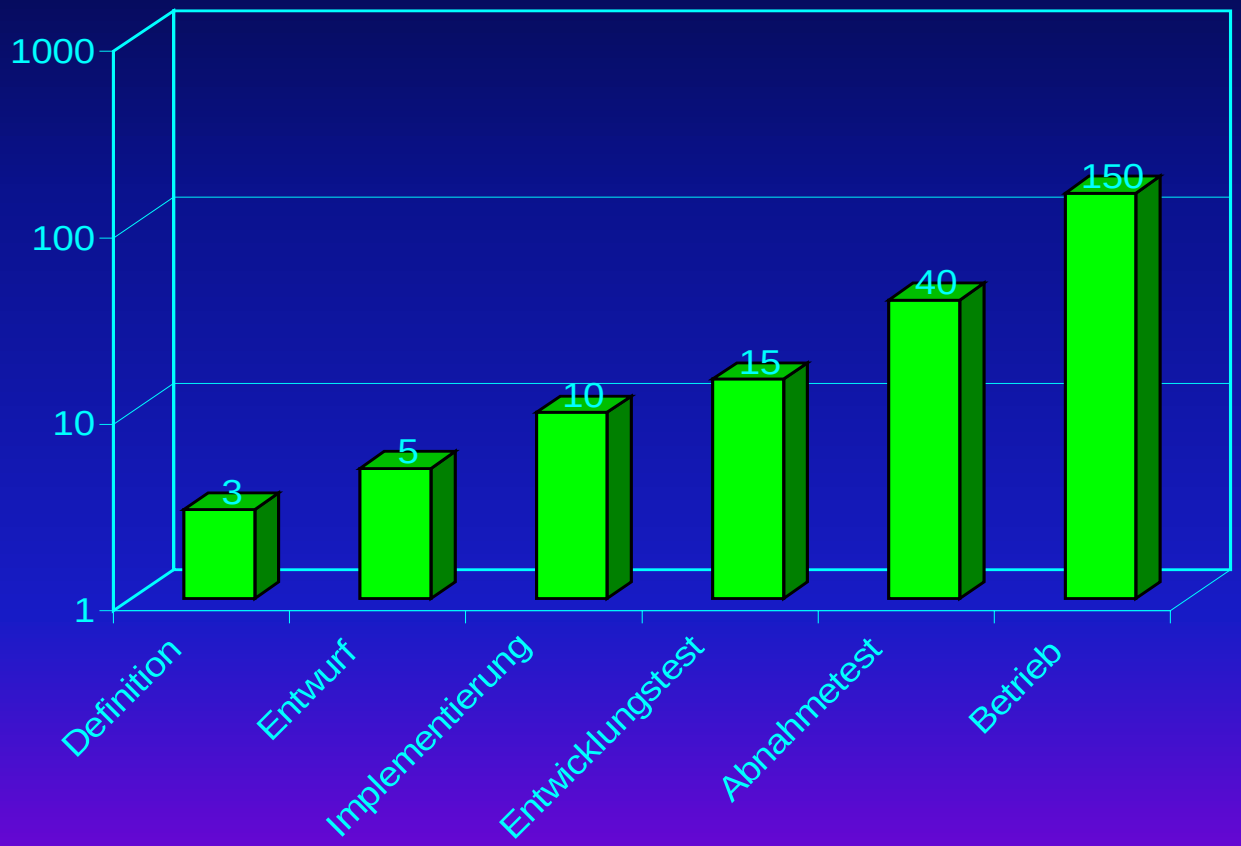
Überblick

- Entstehung und Entdeckung von Fehlern
  (aus Balzert, IEEE Software, Jan. 1985, S.83)



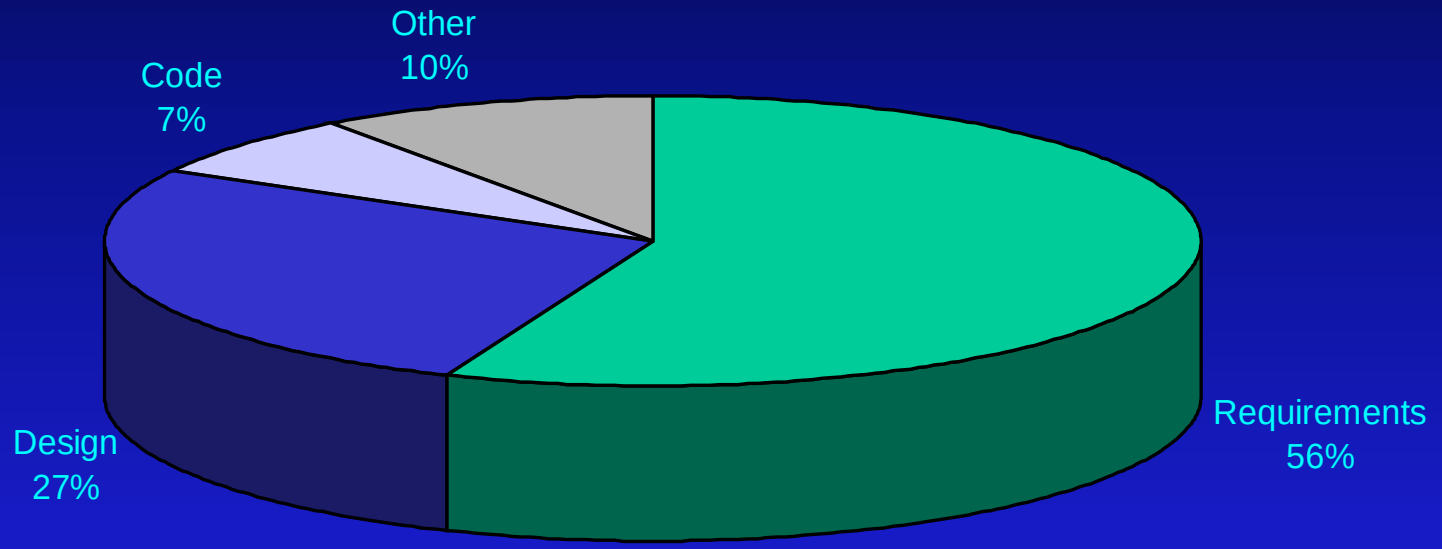
Fehlerbeseitigung

- Relative Kosten zur Fehlerbeseitigung
  (H. Balzert, 1998; Boehm, 1976)



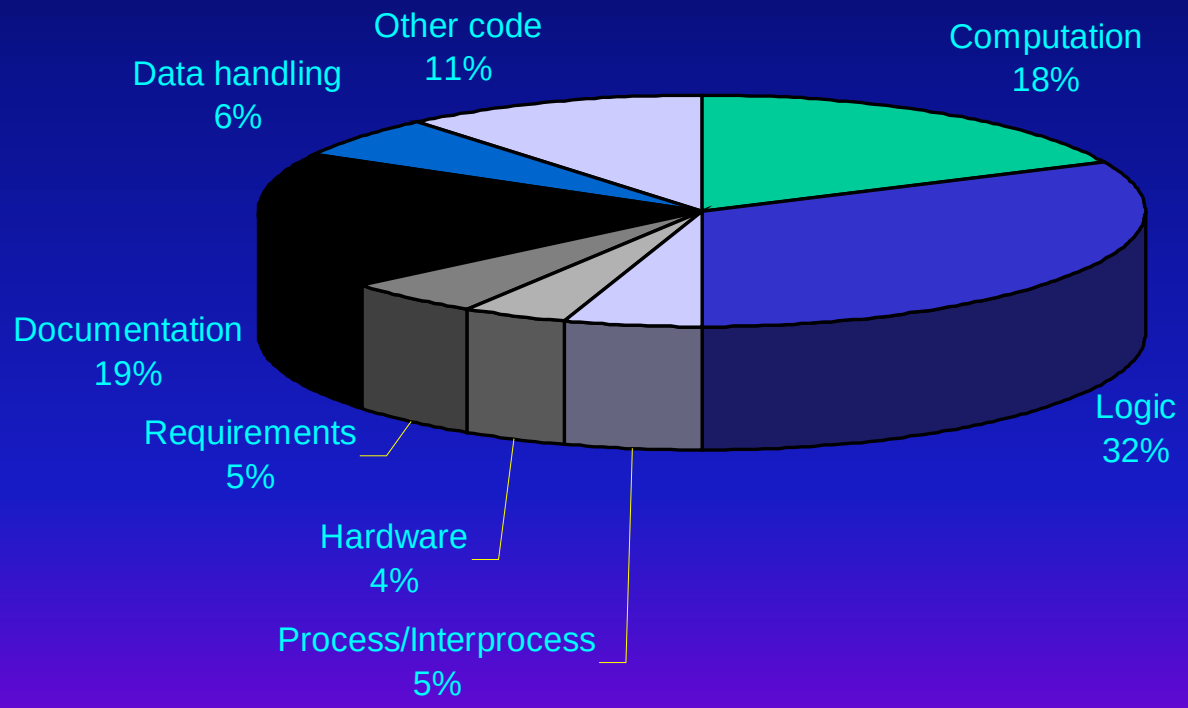
Fehlerverteilung

- Fehlerverteilung (Dick Bender, 1993)

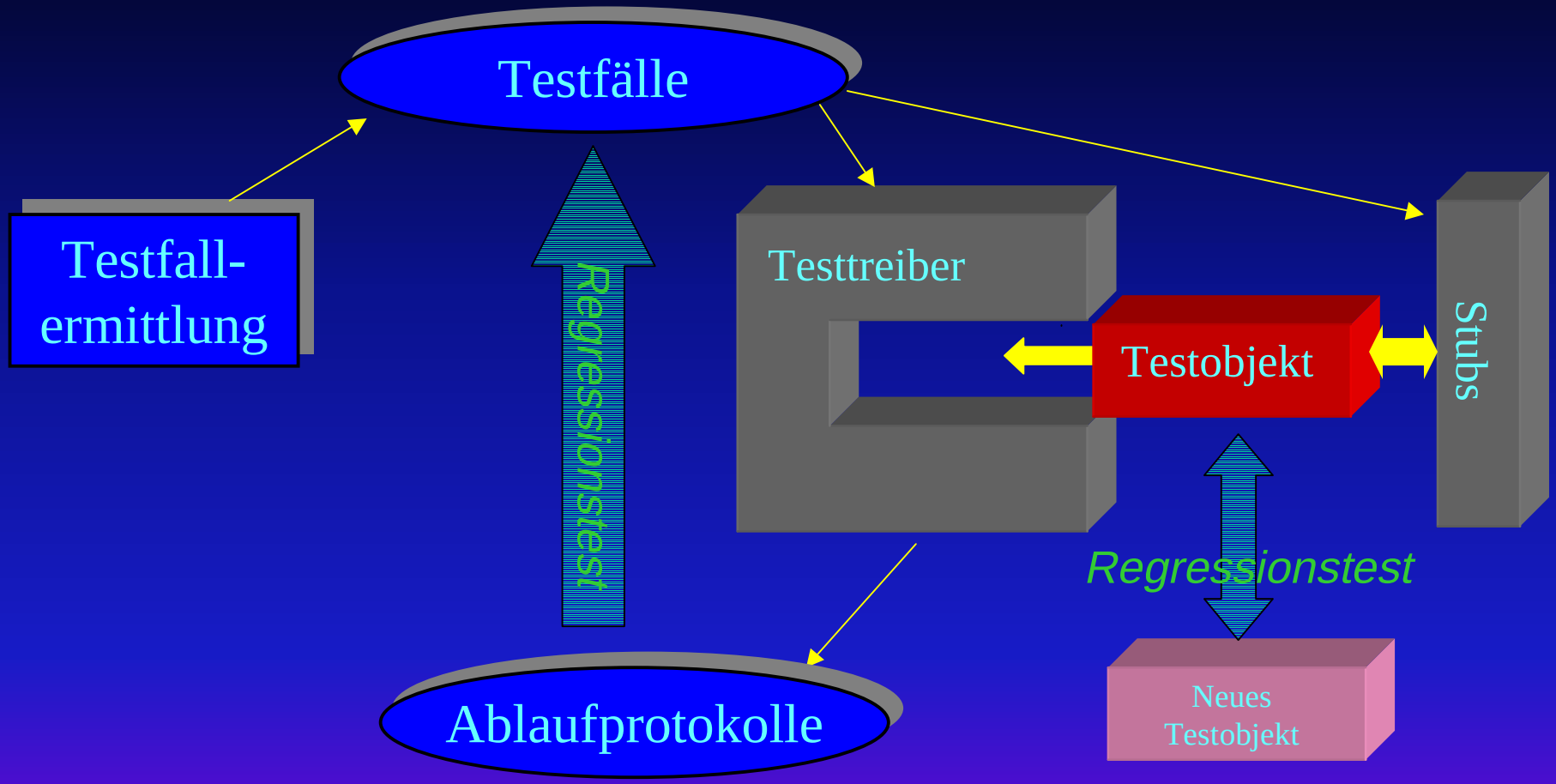


Fehlerverteilung

- Fehlerverteilung:
  (Beispiel von Hewlett-Packard, Grady 1997)



Testdurchführung



Teststufen

- Unit-, Komponenten-, Modul- oder Klassentest
  - Test einer einzelnen Einheit
- Integrationstest
  - Überprüfung des fehlerfreien Zusammenwirkens von Systemkomponenten
- Systemtest
  - abschließender Test des Gesamtsystems
    Funktion, Leistung (Massen-, Zeit-, Streßtest), Benutzbarkeit, Sicherheit, Interoperabilität
- Abnahmetest
  - Test unter Mitwirkung des Auftraggebers
  - Alpha-Test, Beta-Test

Integrationstest

- Integrationsstrategien:
  - nicht-inkrementell:
    - big bang
    - geschäftsprozeßorientiert
  - inkrementell:
    - top-down-Integration
    - bottom-up-Integration
    - funktionsorientiert
    - nach Verfügbarkeit

Testverfahren

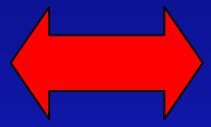
- Manuelle Prüfverfahren:
  - Programminspektion, Review, Walkthrough
  - effektiv: 30-70% (Myers, Balzert)
  - Zeit- und Personalaufwendig
- Whitebox-Test (struktureller Test)
  - Ausgangspunkt: Struktur des Prüflings
  - Messung der Testüberdeckung mittels Metrik
- Blackbox-Test (funktionaler Test)
  - Ausgangspunkt: Spezifikation des Prüflings
  - Sicherstellung der gewünschten Funktionalität

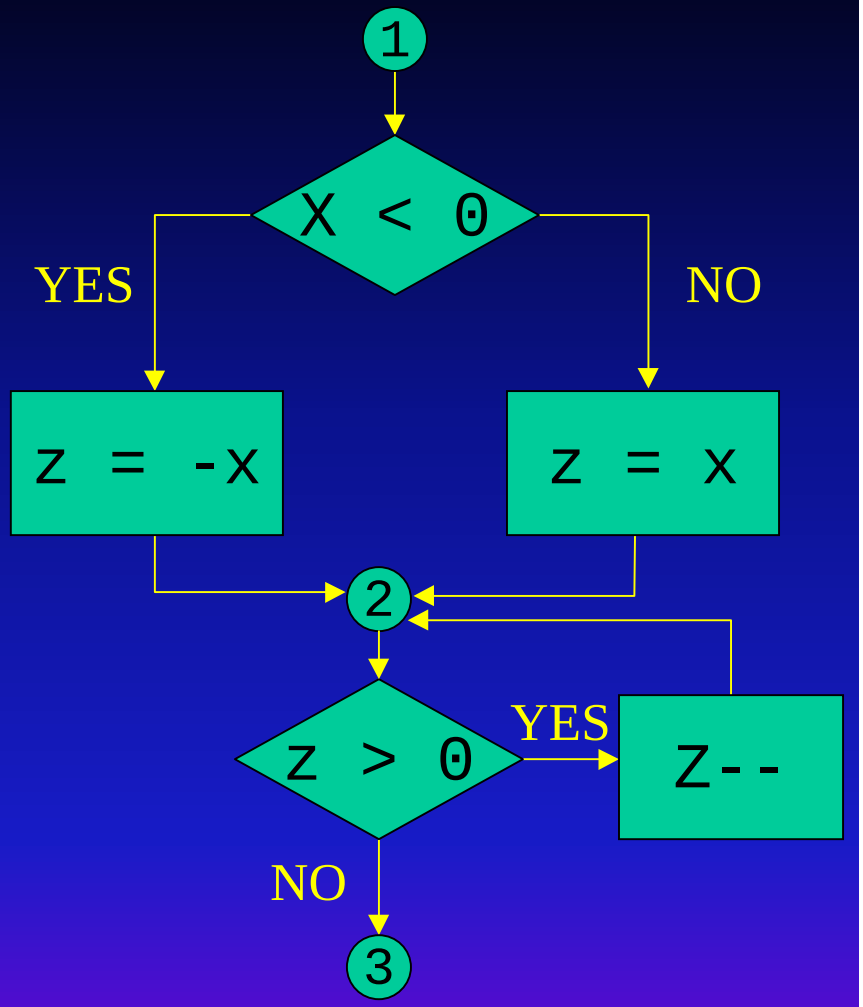
Whitebox-Test

- Instrumentierung des Codes (Zähler einfügen)
- Tests durchführen
- Messung der
  - verarbeiteten Anweisungen
  - ausgewerteten Bedingungen
  - Schleifendurchläufe
- Auswertung der erreichten Überdeckung anhand von Metriken
  - kontrollflußorientiert
  - datenflußorientiert

Kontrollflußorientiert

```
Function count(int x) {
 int z;
 if x < 0 then z = -x;
          else z = x;
 while z > 0 { z--; }
}
```



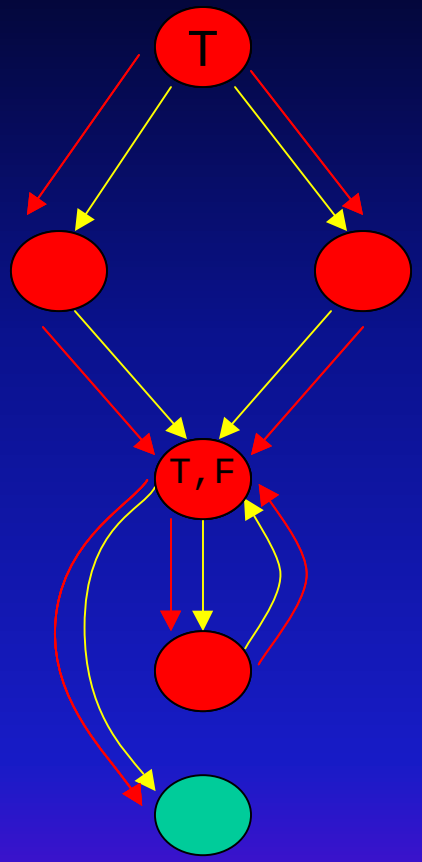


Programm

Flußdiagramm

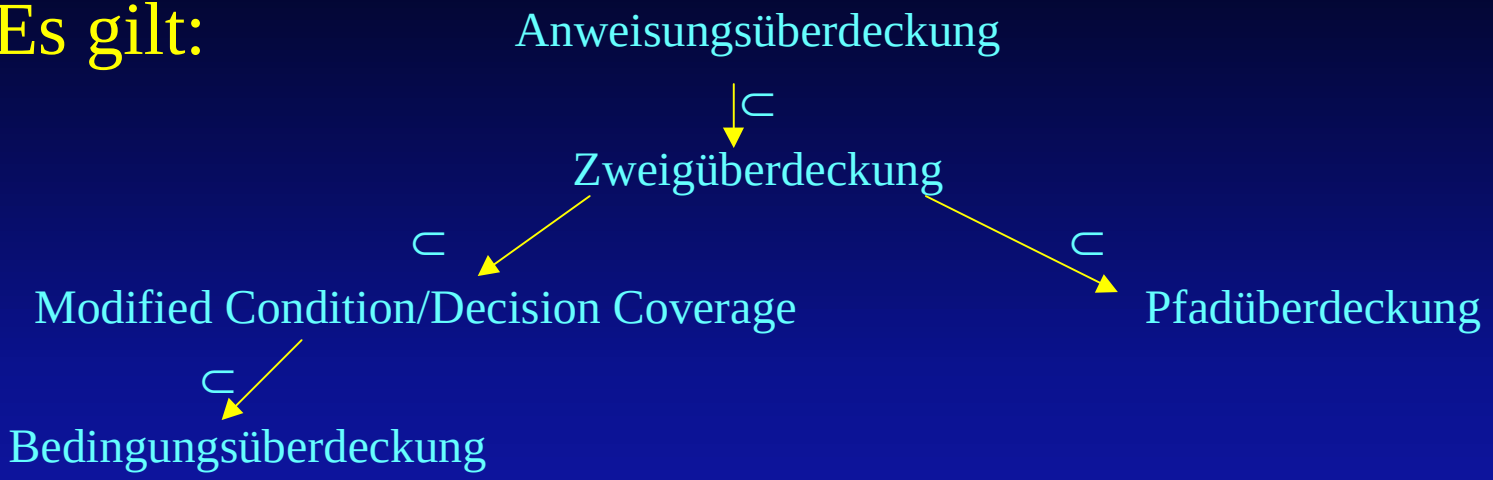
Kontrollflußorientiert

- Metriken:
  - Anweisungsüberdeckung
    - jede Anweisung mind. 1 mal durchlaufen
    - hier: 3 Anweisungen können mit 2 Testdurchläufen überdeckt werden
  - Zweigüberdeckung
    - jeder Zweig (jede Bedingung: *true*, *false*)
    - hier: 2 Bedingungen mit jeweils 2 Möglichkeiten können mit 2 Testdurchläufen überdeckt werden
  - Pfadüberdeckung
    - jeder Pfad muß durchlaufen werden
    - hier: unendlich viele Pfade
  - Bedingungsüberdeckung, MC/DC



Kontrollflußorientiert

- Es gilt:



- Probleme:
  - Infeasible Paths (nicht erreichbare Pfade)
    - z.B. if A do x; [...] if ¬A do y;
    - aus Redundanzen (z.B. Range-Checking-Code vom Compiler)
    - bei Exceptions
  - geeignete Testdaten zu finden

Datenflußorientiert

- Betrachtet werden
  - Definition,
  - lesende Zugriffe,
  - schreibende Zugriffe

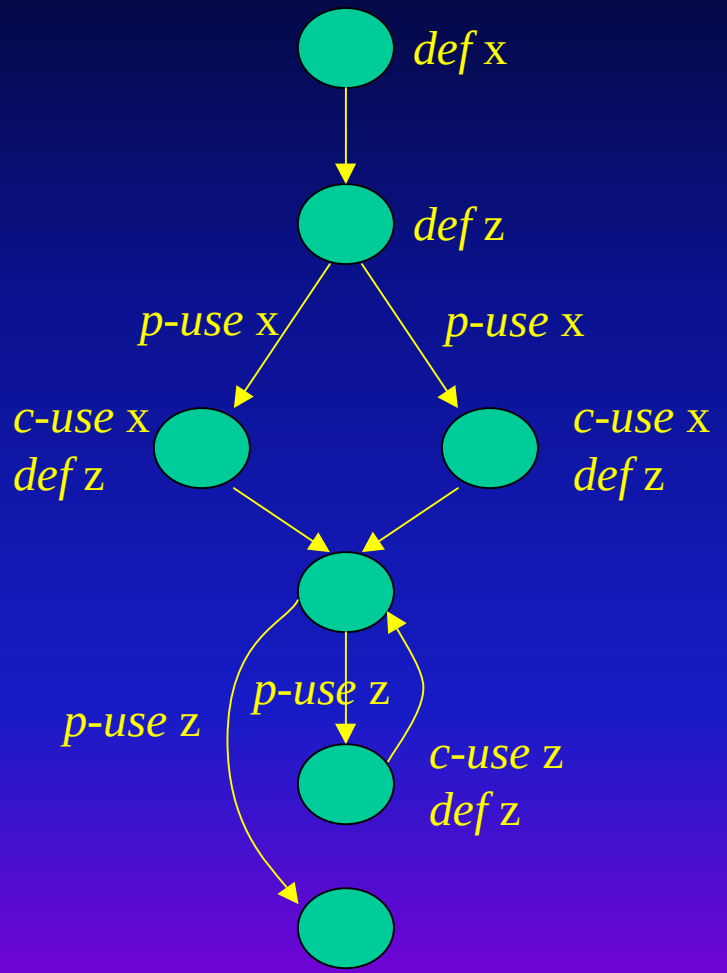
  von Variablen

- *Defs/Uses*-Verfahren:
  - *def:* Wertzuweisung, auch Definitionen, Initialisierung
  - *c-use*: berechnende Benutzung (computational use)
  - *p-use*: prädikative Benutzung (predicate use)

Datenflußgraph

- Kontrollflußgraph mit Datenflußdarstellung:

```
Function count(int x) {
 int z;
 if x < 0 then z = -x;
          else z = x;
 while z > 0 { z--; }
}
```

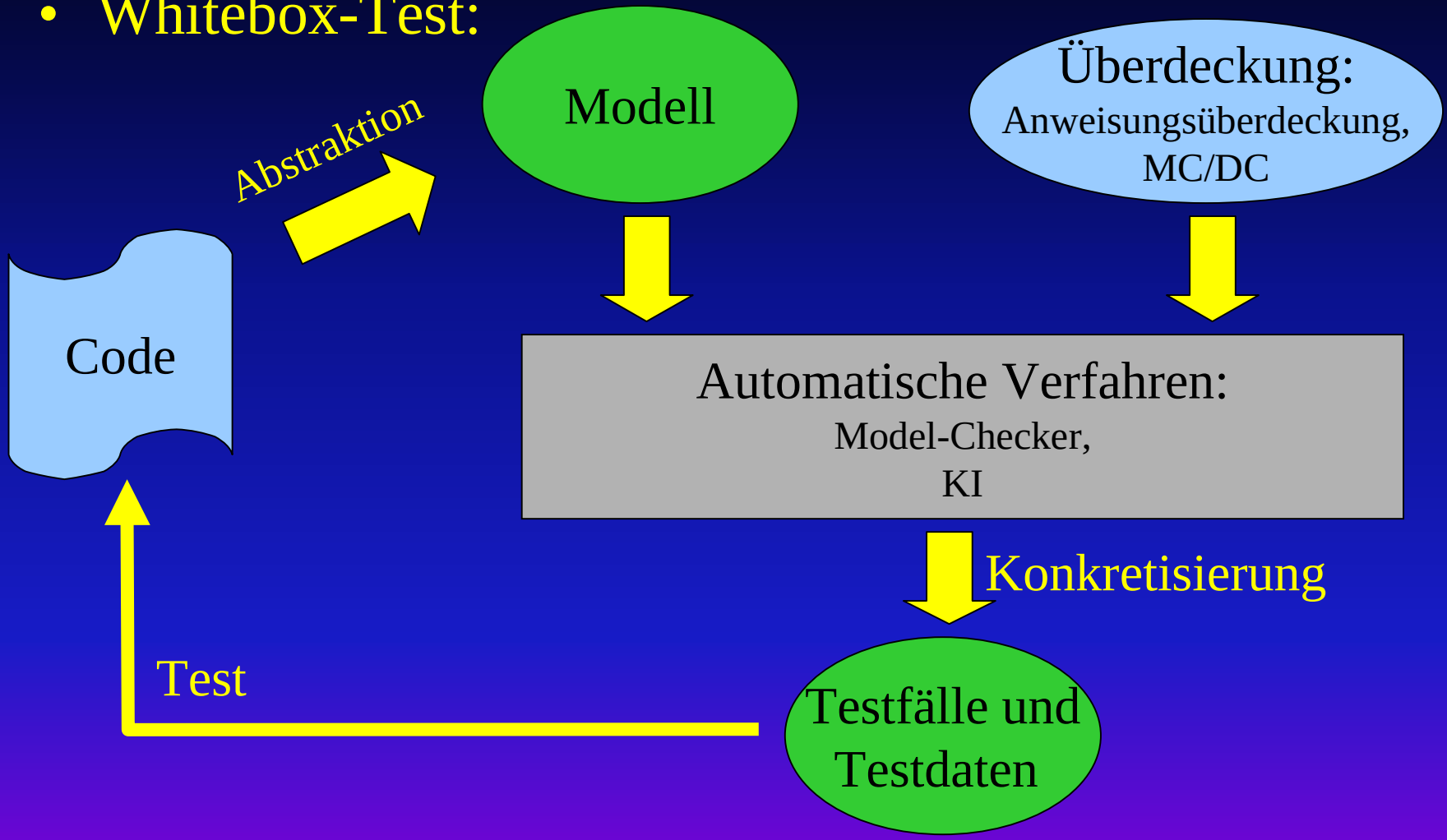


Datenflußmetriken

- Metriken: (Statistiken aus Girgis, Woodware `86)
  - *all defs*: (24%, keine Kontrollflußfehler)
    - jede Definition muß in einer Berechnung oder Bedingung benutzt werden
  - *all p-uses*: (34%, identifiziert sicher Kontrollflußfehler)
    - Jede Kombination aus Variablendefinition und deren prädikative Nutzung muß getestet werden
    - Beinhaltet Zweigüberdeckung
  - *all c-uses*: (48%, identifiziert Berechnungsfehler)
    - Jede Kombination aus Variablendefinition und deren berechnende Benutzung muß getestet werden
  - *all uses*: (insgesamt ca. 70% der Programmierfehler)
    - *c-uses* und *p-uses* zusammen

Testfallermittlung

- Whitebox-Test:



Blackbox-Test

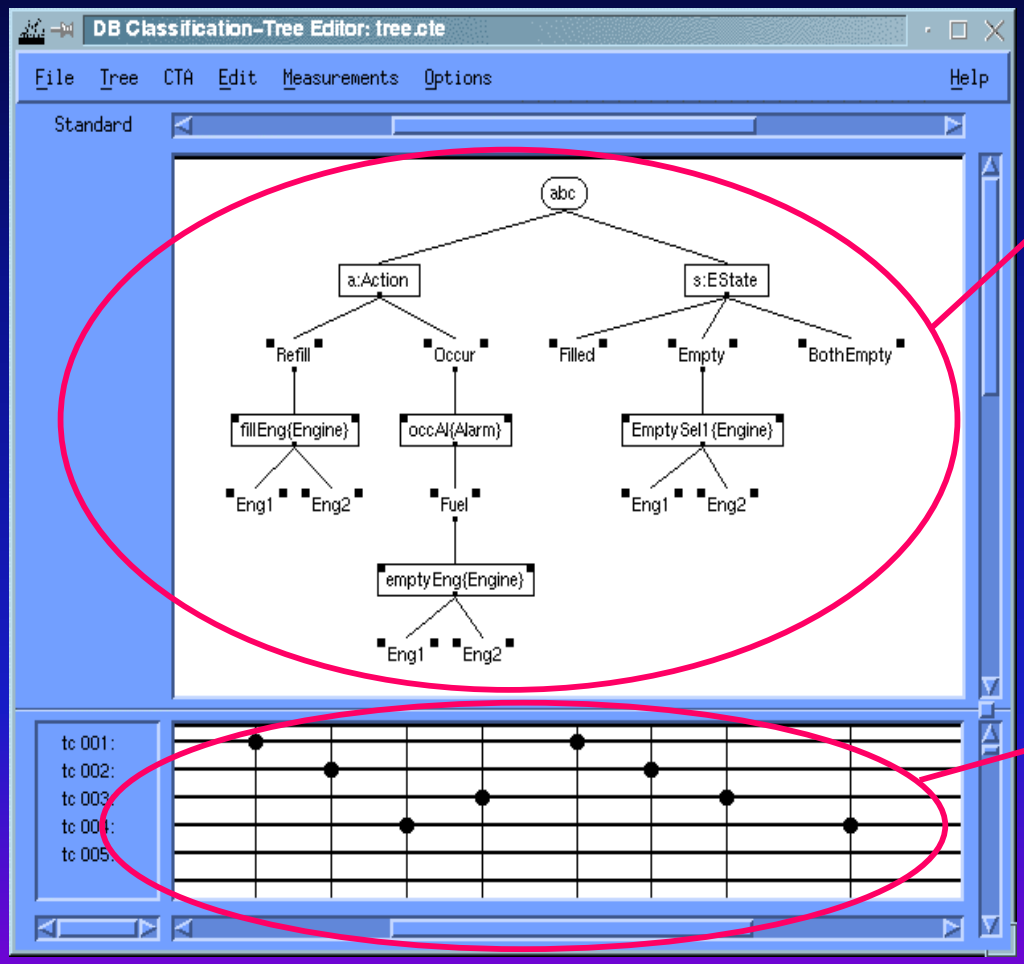
- Blackbox-Test:
  - Ermittlung der Testfälle ausschließlich aus der Spezifikation
  - Ziel ist eine möglichst umfassende und redundanzarme Prüfung der Funktionalität

- Techniken:
  - Äquivalenzklassenbildung
  - Grenzwertanalyse
  - Testsequenzermittlung aus
    - Use-Cases
    - Beispielabläufen
    - Zustandsdiagrammen

Äquivalenzklassen

- Äquivalenzklassenbildung:
  - Definitions- und Wertebereich in Äquivalenzklassen zerlegen
    - gültige Äquivalenzklassen => Funkionstests
    - ungültige Äquivalenzklassen => Stabilitätstests
  - Bsp: Monat : [1..12] zerlegen in
    - eine gültige Äquivalenzklasse : 1 <= Monat <= 12
    - 2 ungültige Äquivalenzklassen: Monat < 1, Monat > 12
- Testfallauswahl:
  - Auswahl eines Repräsentanten aus jeder Äquivalenzklasse
  - max. 1 ungültige Äquivalenzklasse
  - *Grenzwertanalyse:* Werte an Rändern der Äquivalenzklassen
  - *spezielle Werte:* z.B. 0 bei int, Sonderzeichen usw.

Classifaction Tree Editor:



Klassifikationsbaum
≈ Kombination von
Äquivalenzklassen

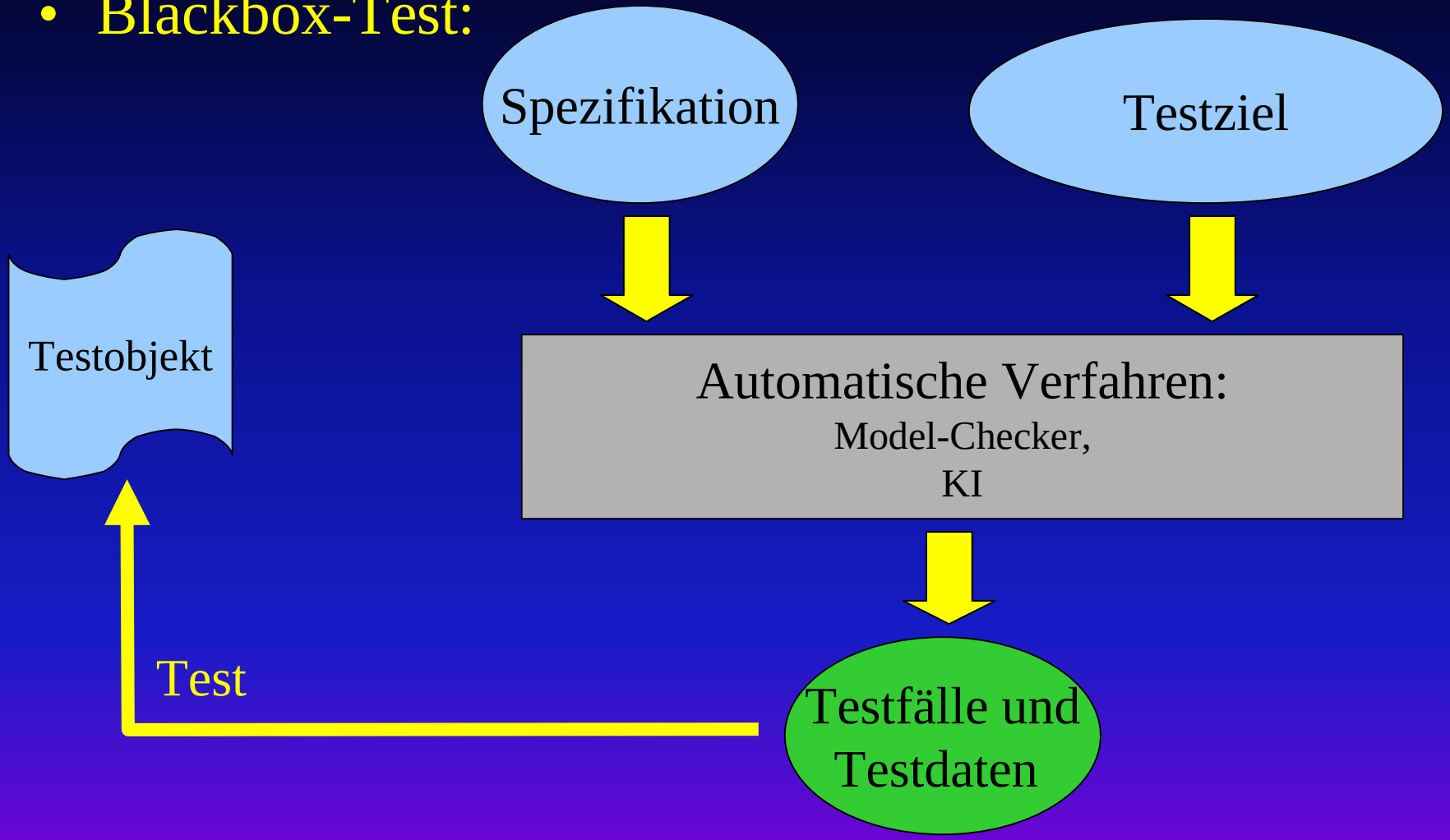
Kombinationstabelle

Zustandsautomaten

- Ableitung von Testfällen aus Zustandsautomaten:
  - über Graphen:
    - Transitionsüberdeckung (Transitionstour)
    - Zustandsüberdeckung
    - Pfadüberdeckung
  - durch Simulation
    - manuell mit Aufzeichnung
    - symbolische Ausführung
  - logikbasiert
    - Transformation in Aussagenlogik/Prädikatenlogik, deklarative Beschreibung der Testfälle

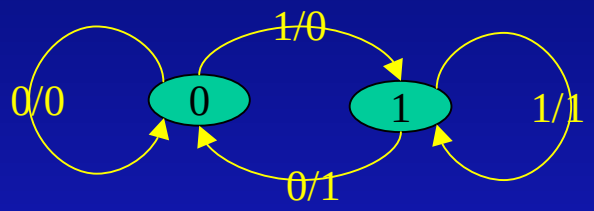
Testfallermittlung

- Blackbox-Test:



Mealy-Automaten

- Methoden:
  - Transitionstour, W-Methode, UIO-Methode
- Vorteile:
  - vollautomatisch
  - auch Korrektheitsbeweis
- Nachteile:
  - nur endliche Systeme
  - Korrektheitsbeweise erfordern starke Annahmen über Testobjekt



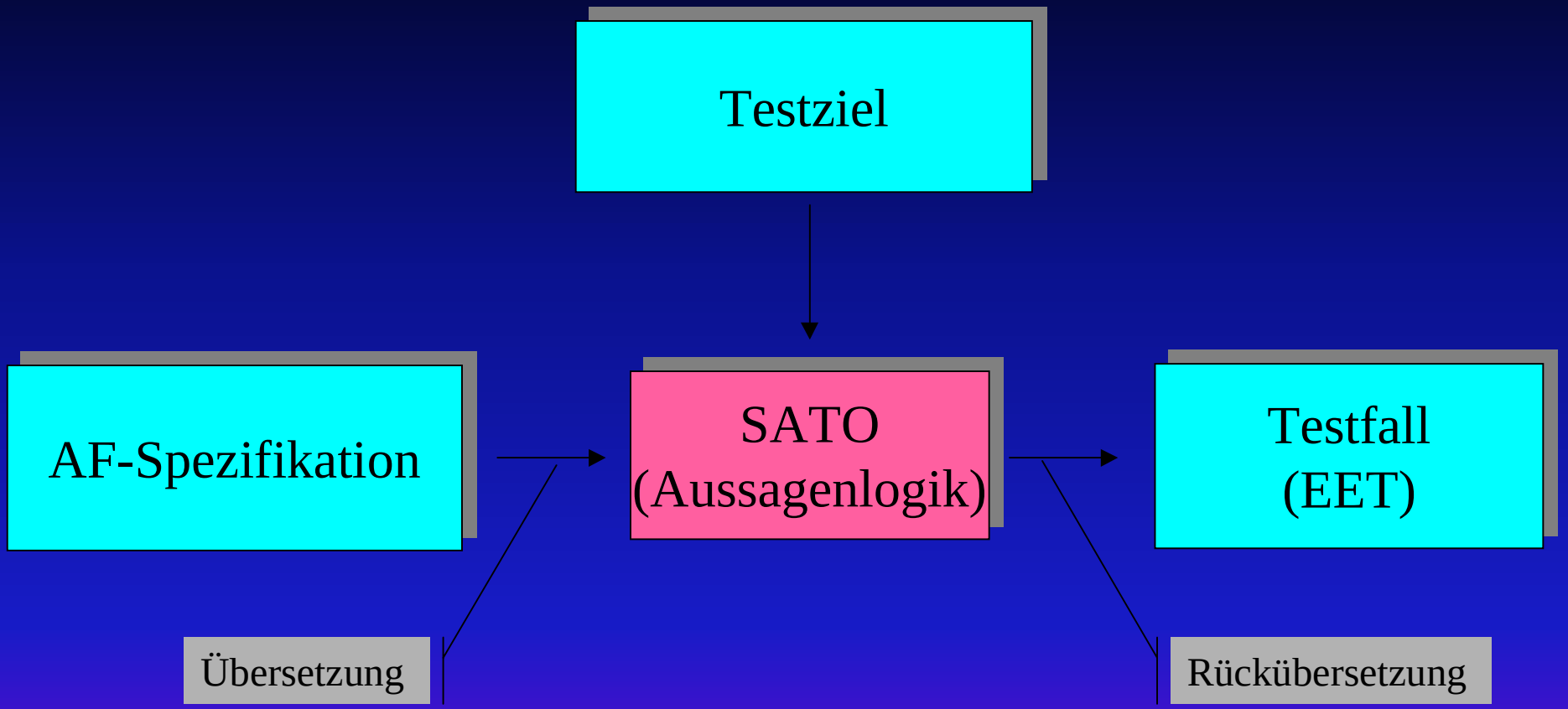
Transitionstour

- Findet den kürzesten Pfad durch den Automaten, der alle Transitionen überdeckt
- Vorteile:
  - vollautomatisch
- Nachteile:
  - Automat muß bestimmte Eigenschaften haben:
    - stark zusammenhängend
    - errechnete Transitionssequenz muß wirklich durchführbar sein

Model-Checking

- Methode:
  - Verwendung von Gegenbeispielen vom Model-Checker
  - Ermittlung der Systemläufe durch die Lösung aussagenlogischer Formeln
- Vorteile:
  - vollautomatisch
- Einschränkungen
  - nur zustandsendliche Systeme
    sonst Abstraktion notwendig

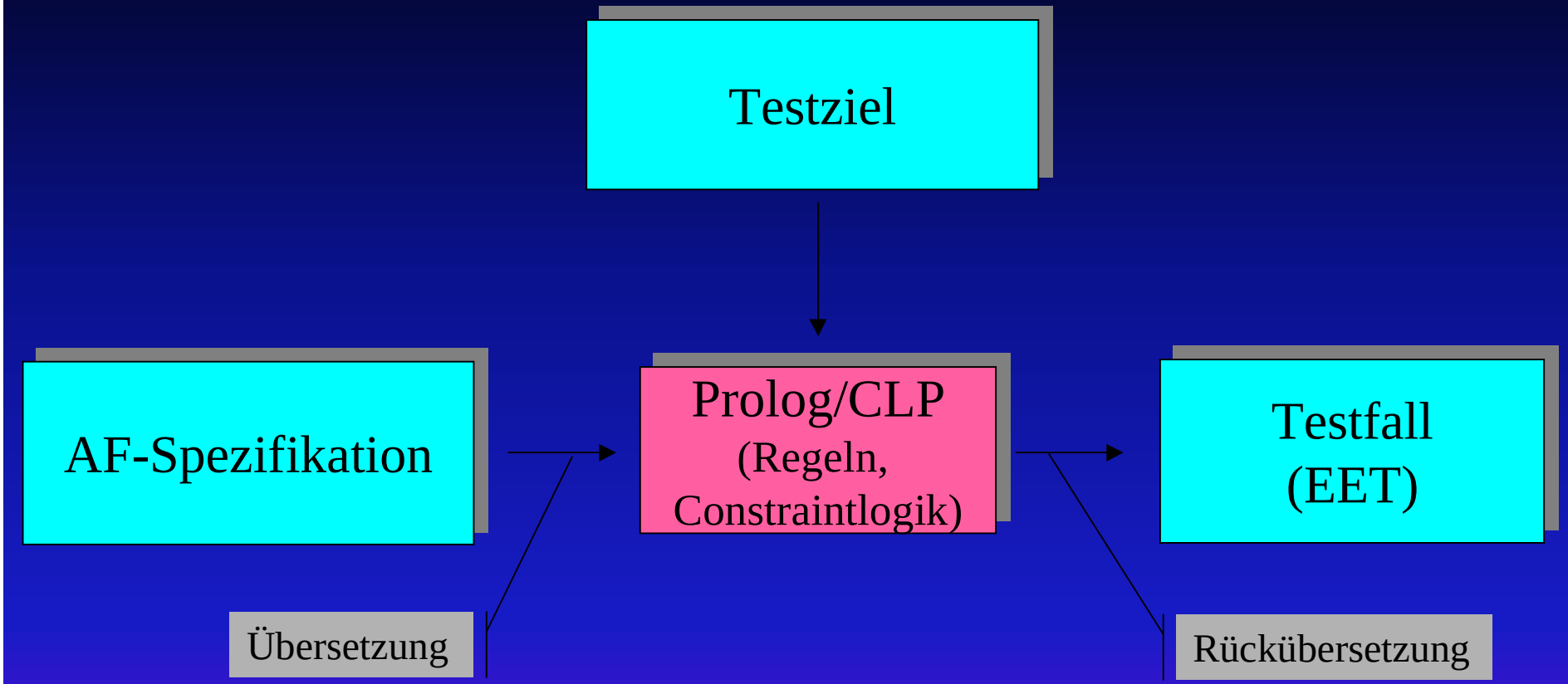
Aussagenlogik



CLP

- Methode:
  - Codierung in regelbasierte Systeme mit Constraint-Mechanismen (CLP = Constraint Logic Programming)
  - symbolische Simulation
- Vorteile:
  - vollautomatisch
  - auch unendliche Systeme
- Einschränkungen
  - Datentypen / Arithmetik
  - Schwierigkeiten mit nichtlinearen Prädikaten

Constraintlogik



Schluß

- Weitgehend gelöst:
  - Metriken für Whitebox-Tests
- Problematisch:
  - Testziele definieren
- Aktuell:
  - automatische Testfallermittlung aus
    - Use-Cases,
    - graphischen Beschreibungstechniken
  - Generierung von Testtreibern, Stubs

Next Events:

Steinheil

Alex B. Schmidt:

Anwendungen der Relationenalgebra

Dienstag 18.7.2000, 17.00 Uhr, Raum1546