Funktionales Testen

Software-Praktikum

Andreas Zeller • Universität des Saarlandes

Testen

Testen

Noch mehr Testen

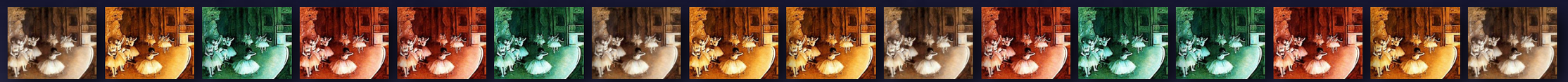
Testen

Software ist vielfältig

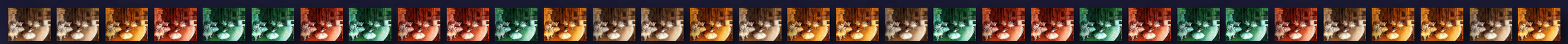
Software ist vielfältig

Software ist vielfältig

Software ist vielfältig



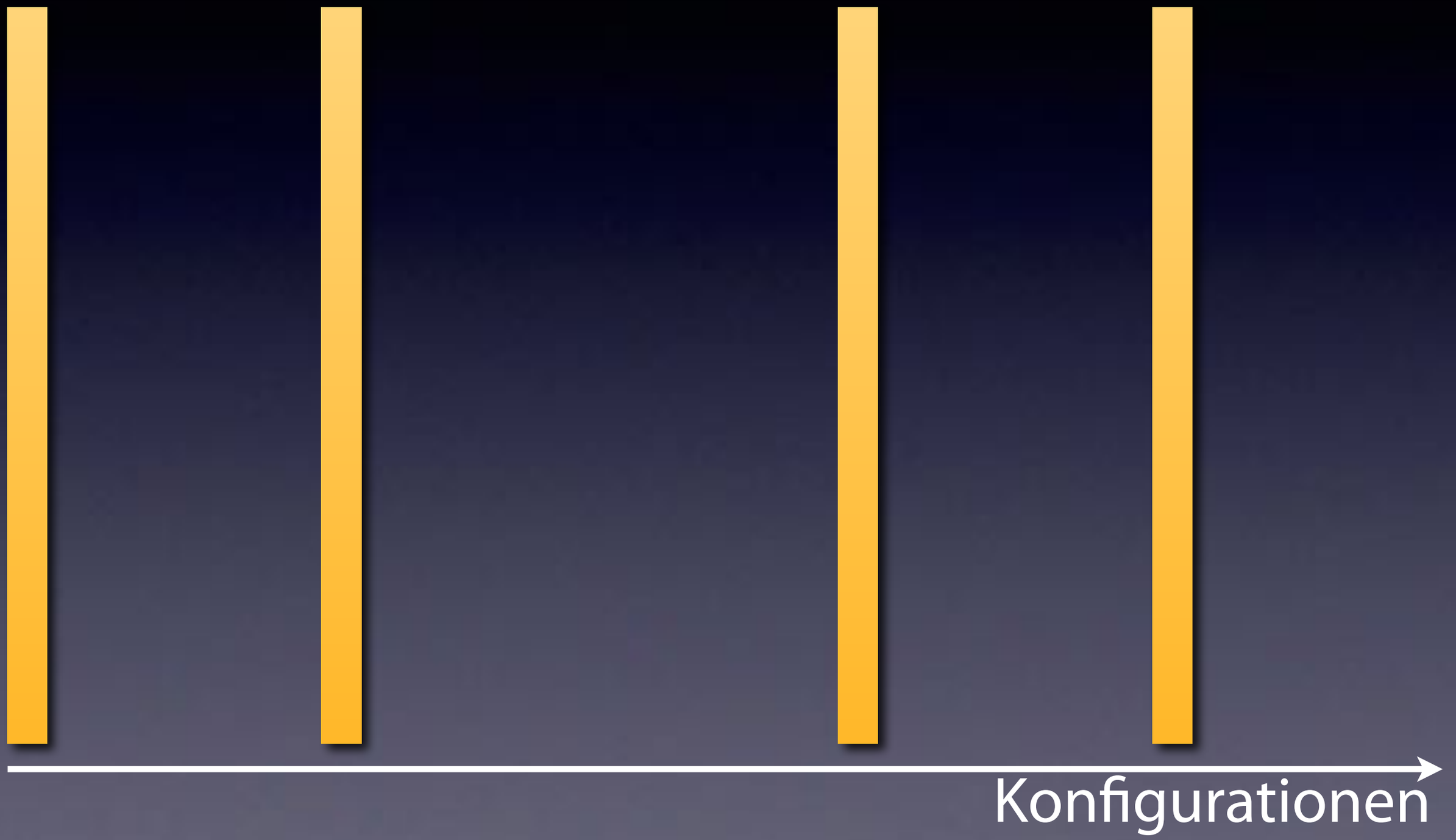
Software ist vielfältig



Software ist vielfältig

Testen

Was testen?

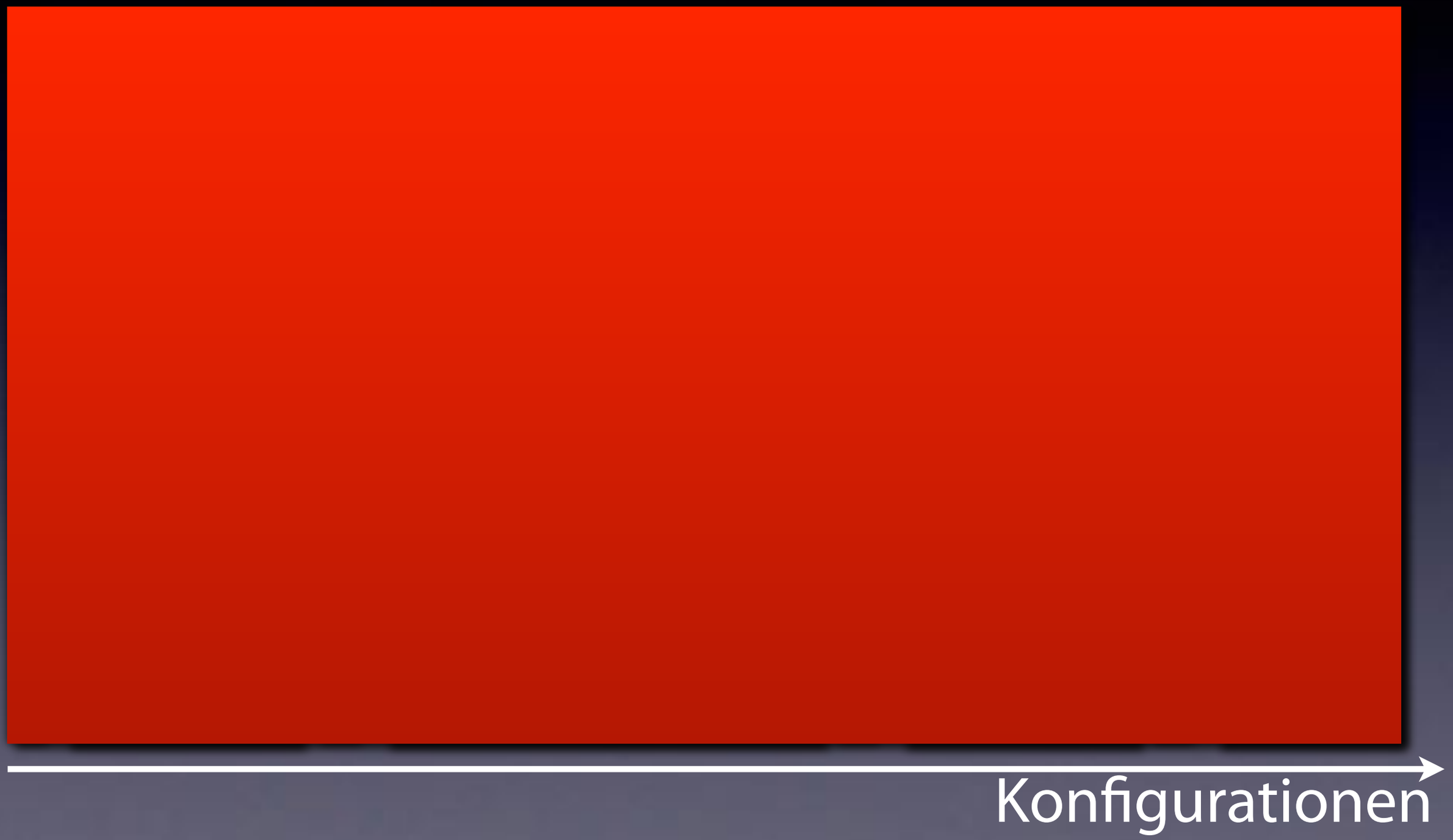


Dijkstras Fluch

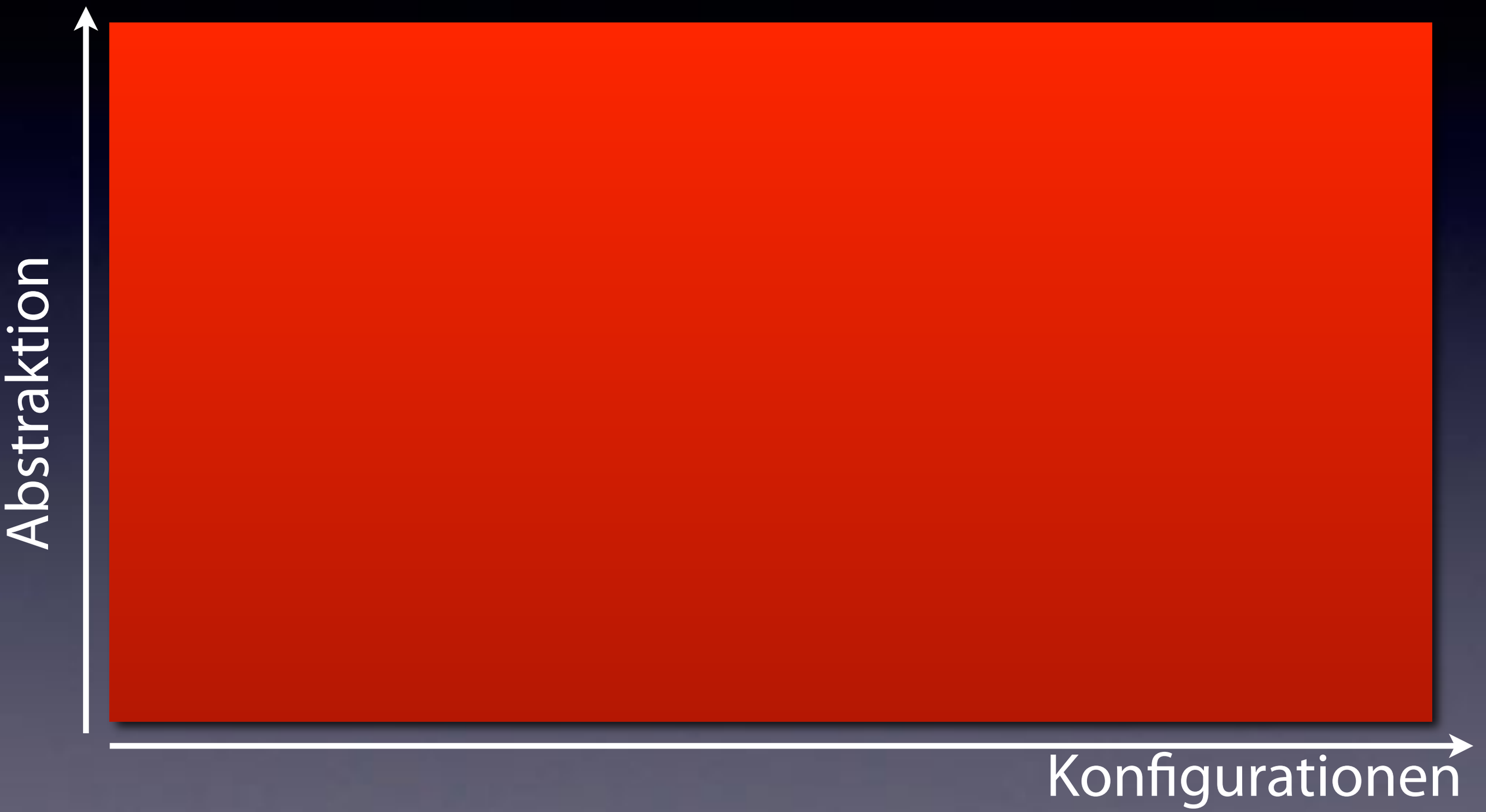
Testen kann nur
die *Anwesenheit* von Fehlern zeigen,
nicht jedoch deren *Abwesenheit*

Konfigurationen

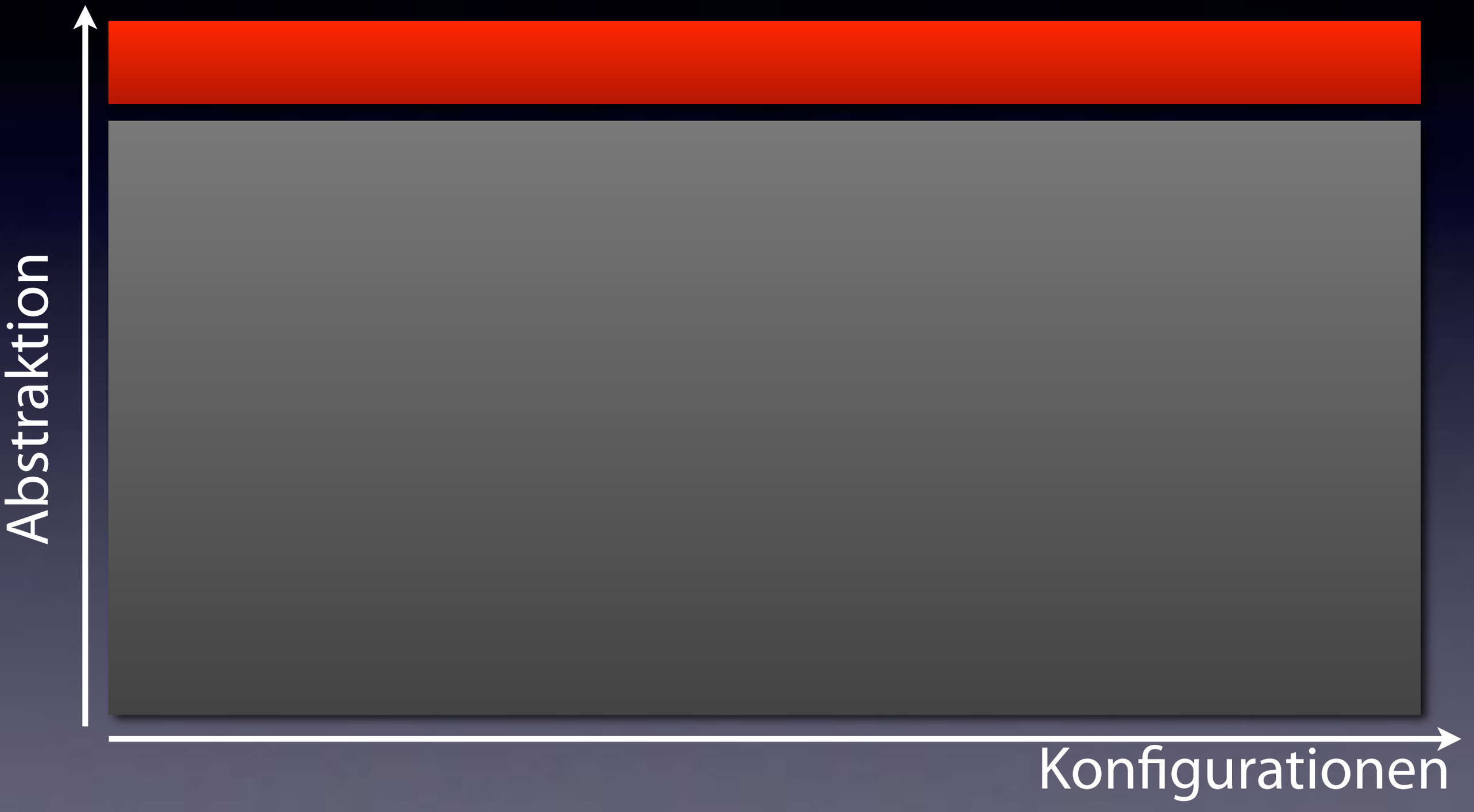
Formale Verifikation



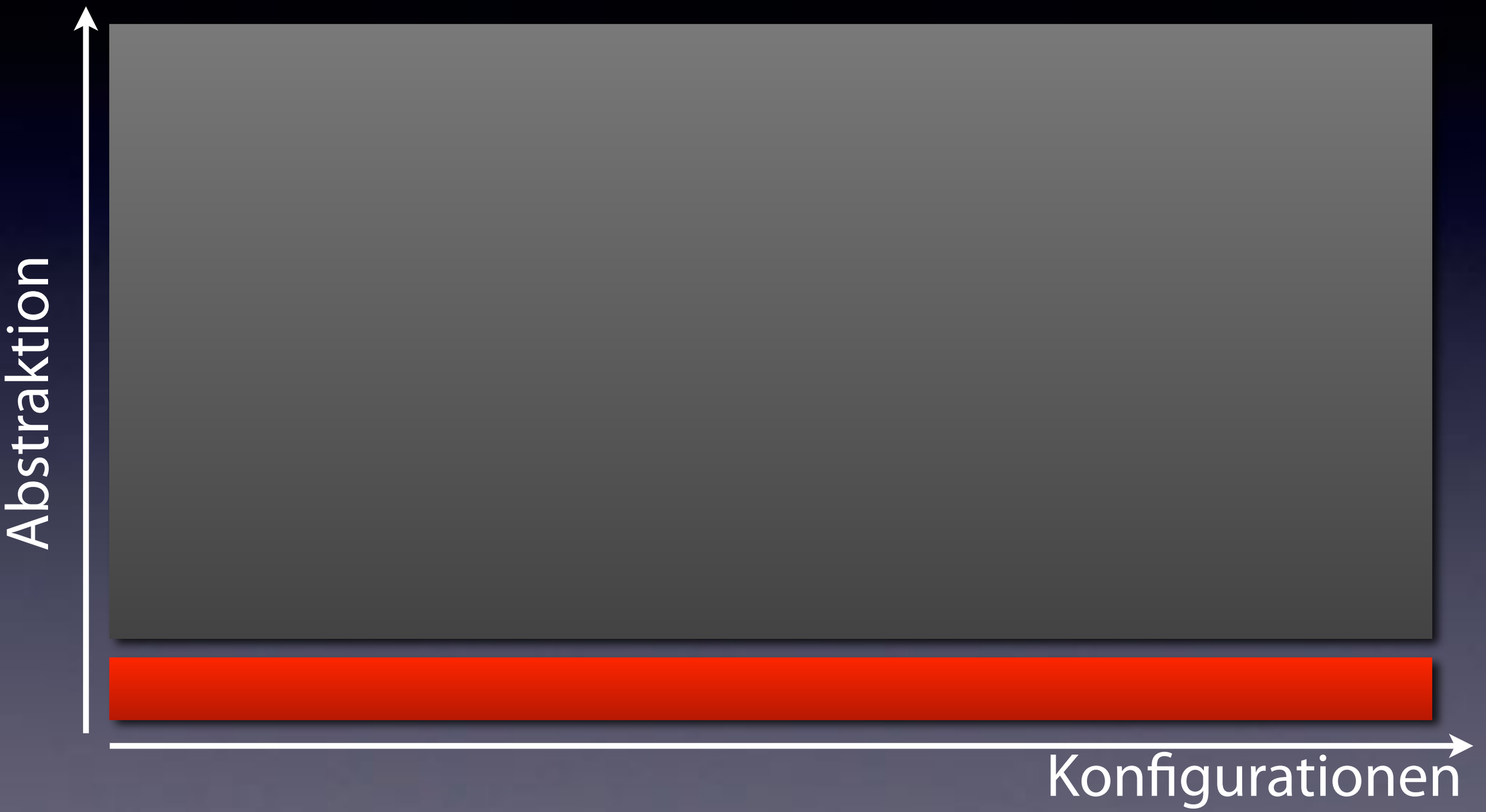
Formale Verifikation



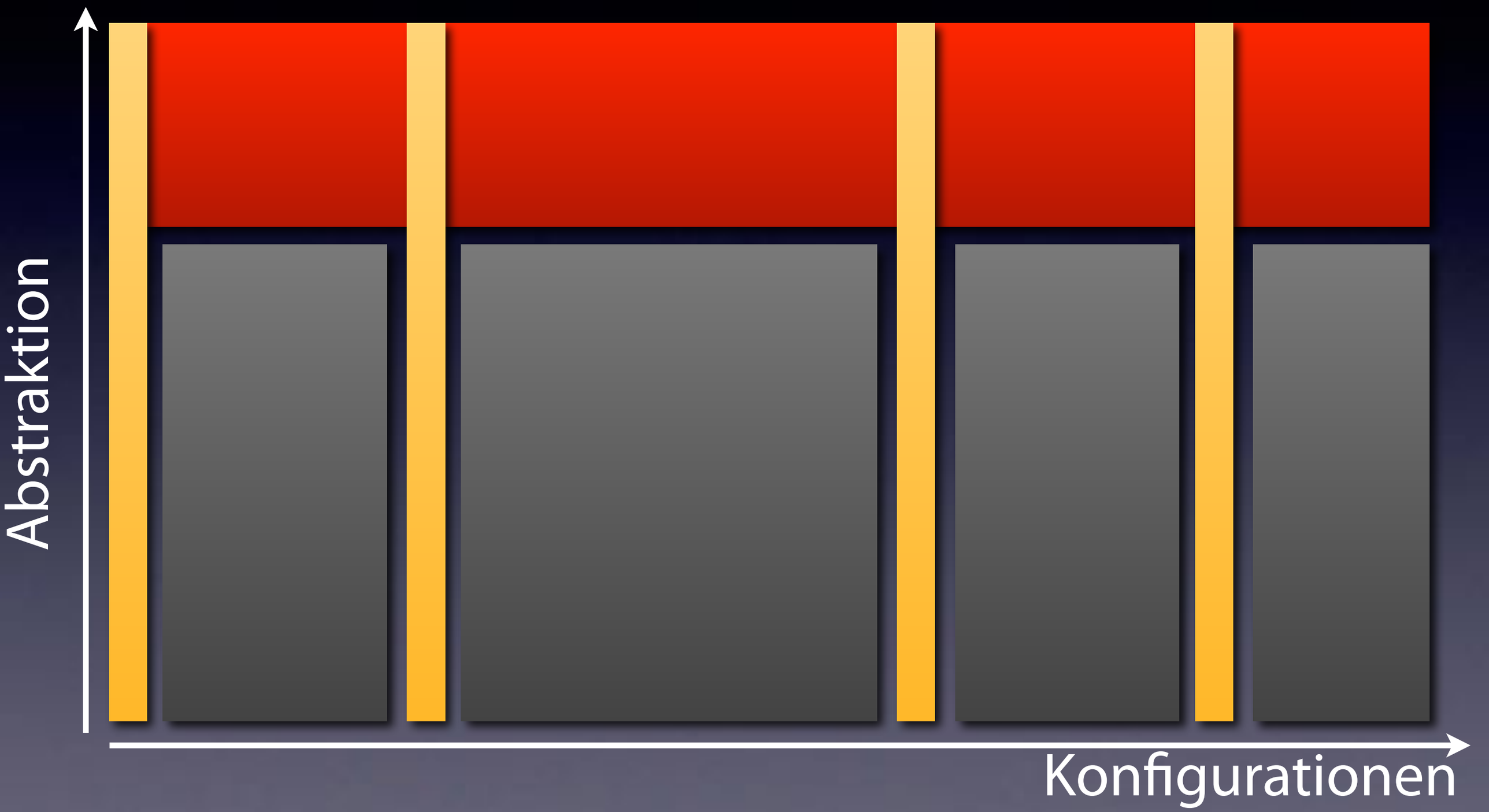
Formale Verifikation



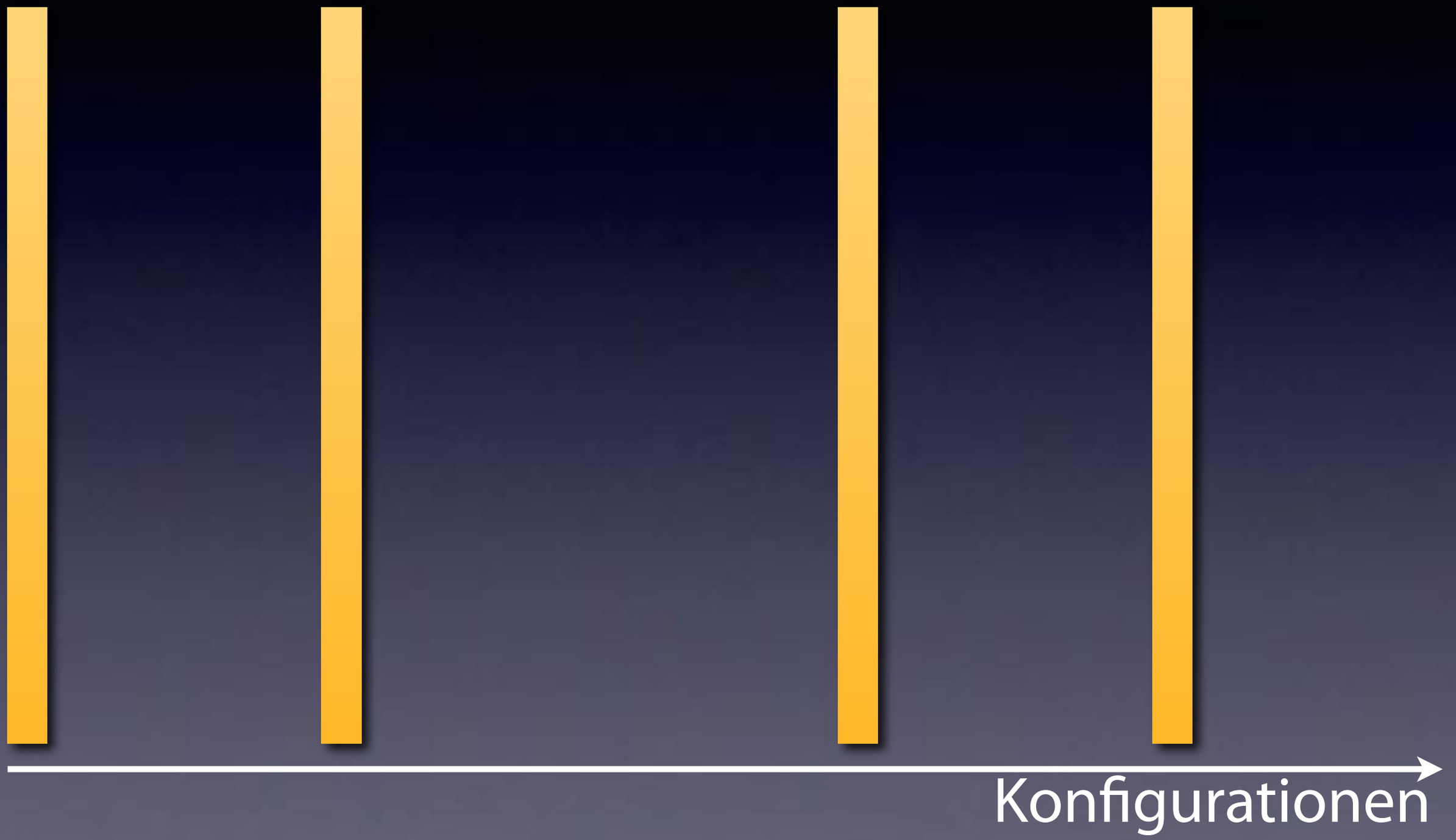
Formale Verifikation



Das Beste zweier Welten



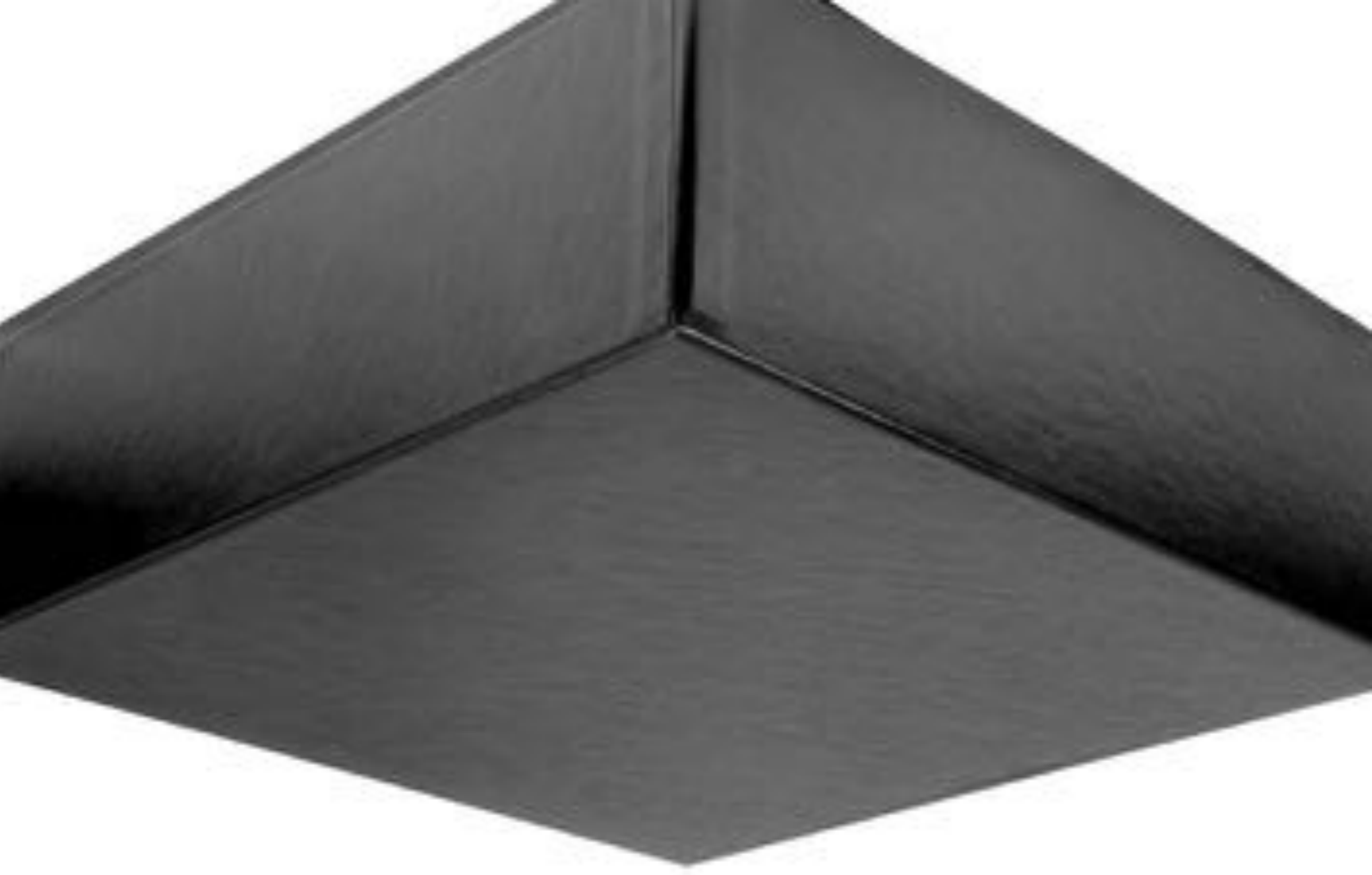
Was testen?

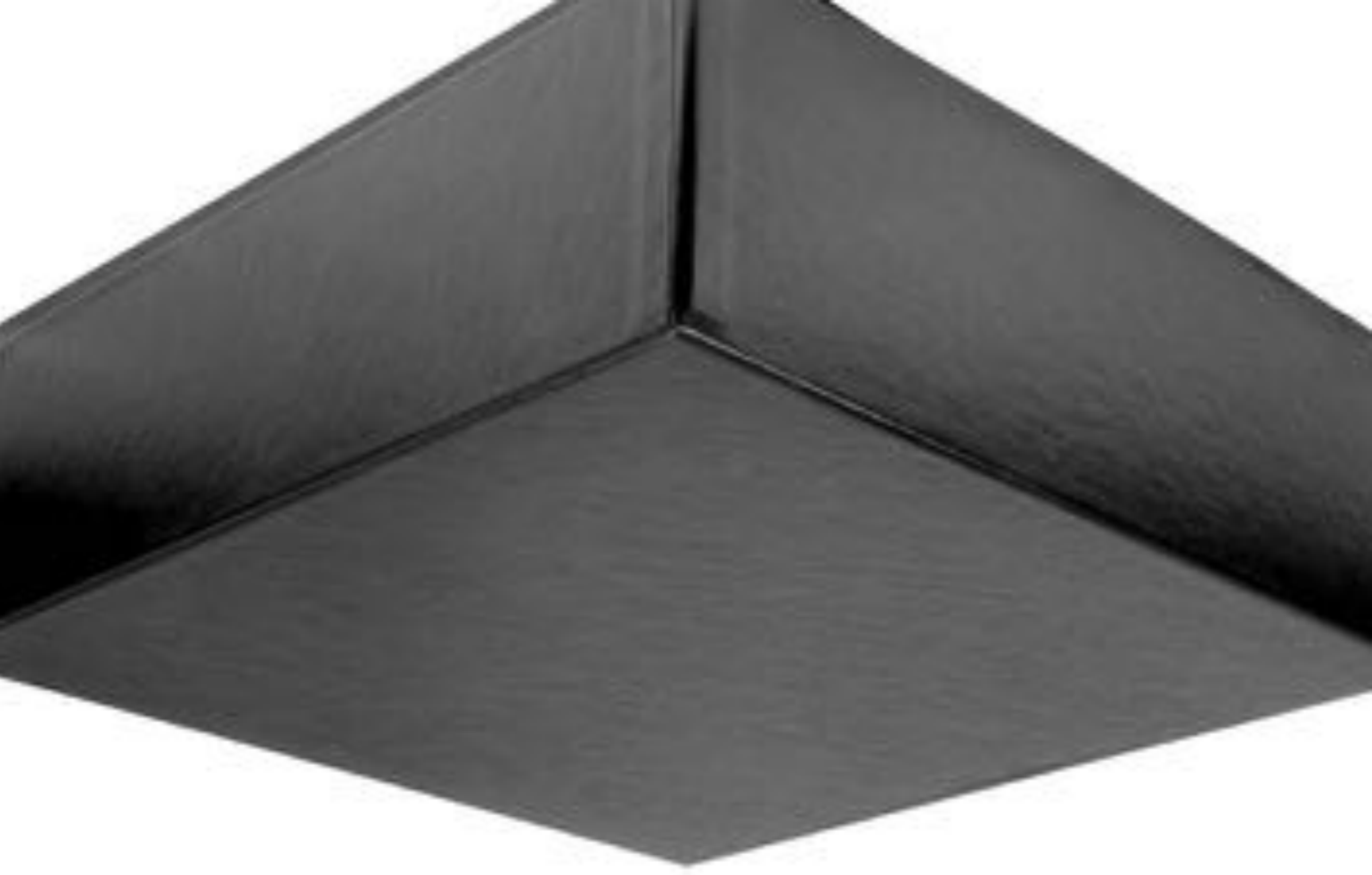


Funktionales Testen

Software-Praktikum

Andreas Zeller • Universität des Saarlandes



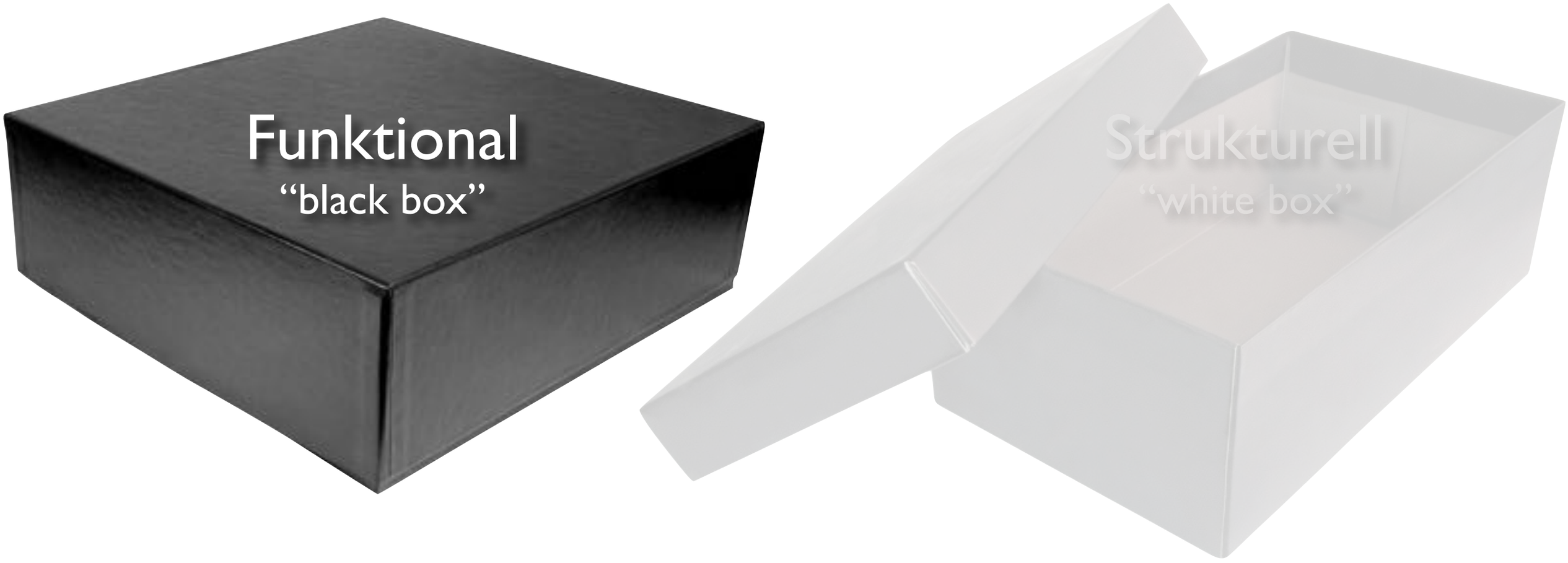


Test-Taktiken

- Tests basieren auf *Spezifikation*
- Test deckt soviel *spezifiziertes* Verhalten wie möglich ab

- Tests basieren auf *Code*
- Test deckt soviel *implementiertes* Verhalten wie möglich ab

Warum Funktional?



- Programm-Code wird nicht benötigt
- Frühes funktionales Testen hat Vorteile
  zeigt Probleme mit Spezifikation auf • prüft *Testbarkeit* • reichert Spezifikation mit weiteren Details an • kann sogar als Spezifikation dienen

Warum Funktional?

- Am besten für Probleme *fehlender Logik*
  Häufiges Problem: Details wurden einfach nicht implementiert
  Strukturelles Testen erkennt fehlenden Code

- Anwendbar auf allen Testebenen
  Unit-Tests • Integrationstests • Systemtests • Regressionstests

Zufallstesten

- Wähle mögliche Eingaben gleichmäßig aus
- Vermeidet *Voreingenommenheit* des Testers
  Ein echtes Problem: Der Tester kann die gleichen logischen Fehler und falsche Annahmen machen wie der Programmierer (besonders, wenn es sich um dieselbe Person handelt)
- Alle Eingaben sind gleichermaßen wertvoll

PUNKTE
0

Winkel
Kraft

PUNKTE
0

Infinite Monkey Theorem

Youtube

Winkel

$2^{32} = 4.294.967.296$
verschiedene Werte

Kraft

$2^{32} = 4.294.967.296$
verschiedene Werte

$2^{32} = 4.294.967.296$
verschiedene Werte

×

$2^{32} = 4.294.967.296$
verschiedene Werte

=

$2^{64} = 18.446.744.073.709.551.616$
verschiedene Läufe

18.446.744.073.709.551.616 Minuten

gadgets-club.com

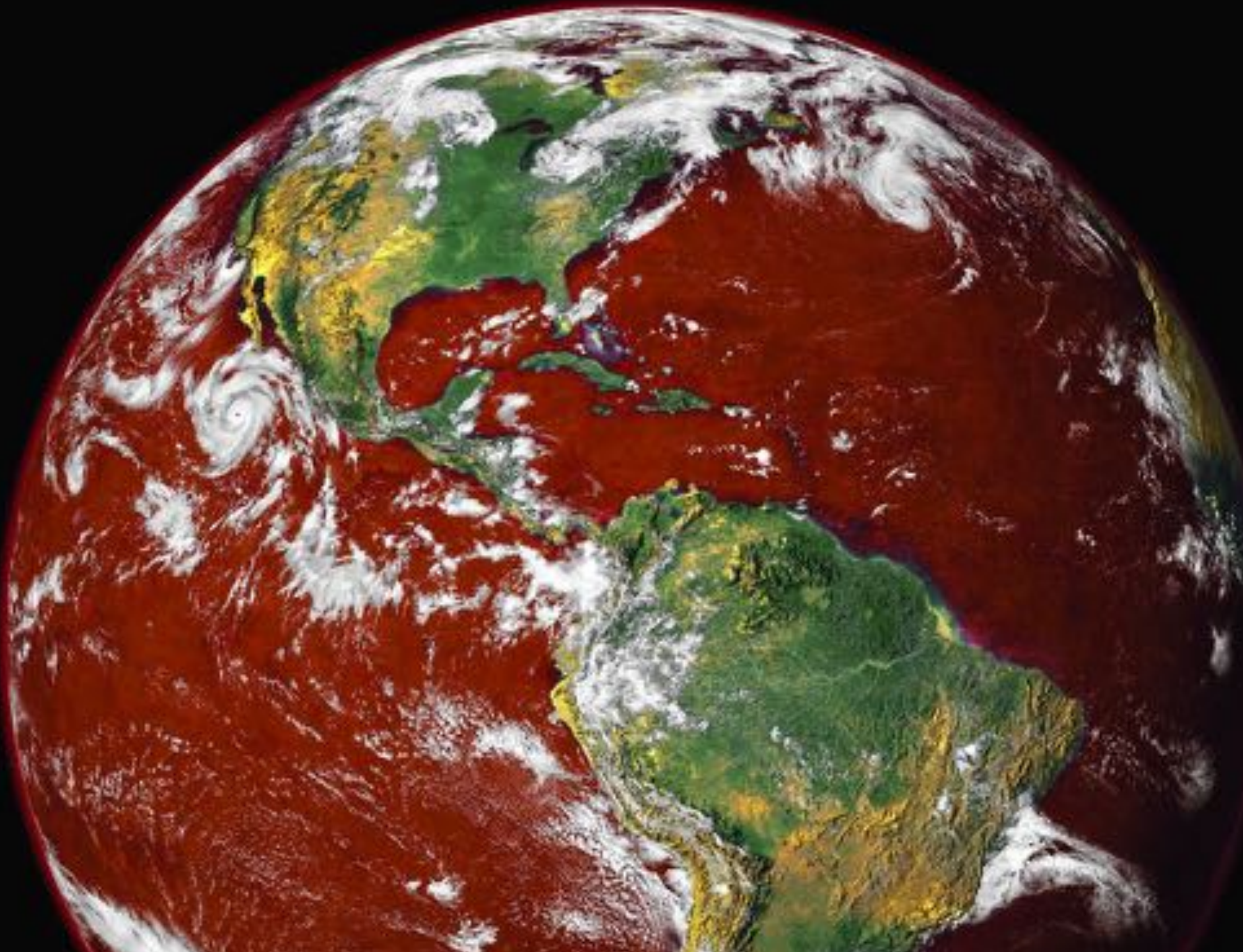
9.223.372.036.854.775.808 Minuten

4.611.686.018.427.387.904 Minuten

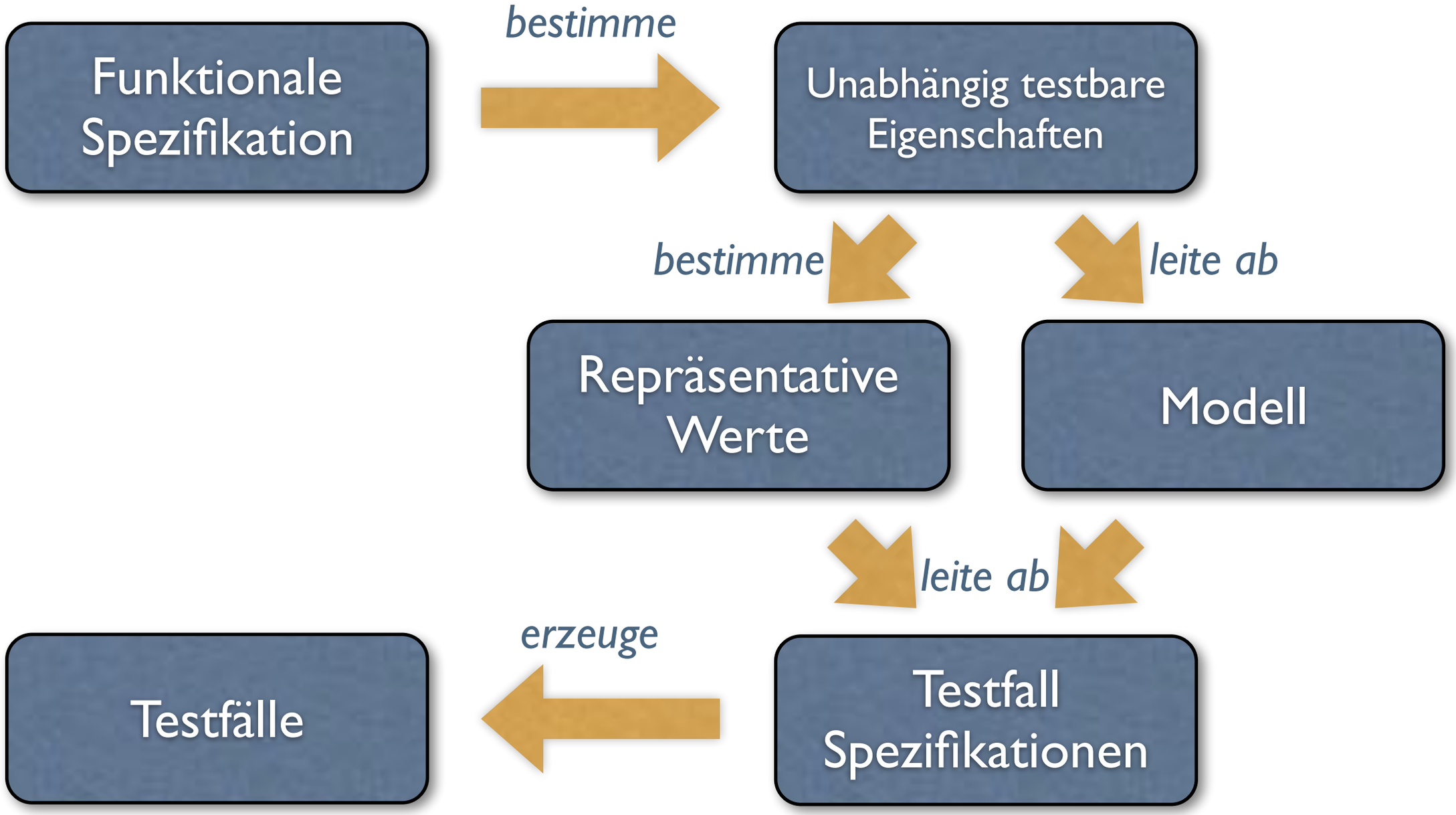
18.446.744.073.709.551.616

×

1 Minute



Systematisches funktionales Testen



Testbare Eigenschaften

- Teile System in *unabhängig testbare Eigenschaften* (independently testable features, ITF) auf
- Ein ITF muss keine Unit oder Subsystem der Software sein
- Beim Systemtesten sind ITFs durch Benutzeroberflächen oder APIs gegeben

Testbare Eigenschaften

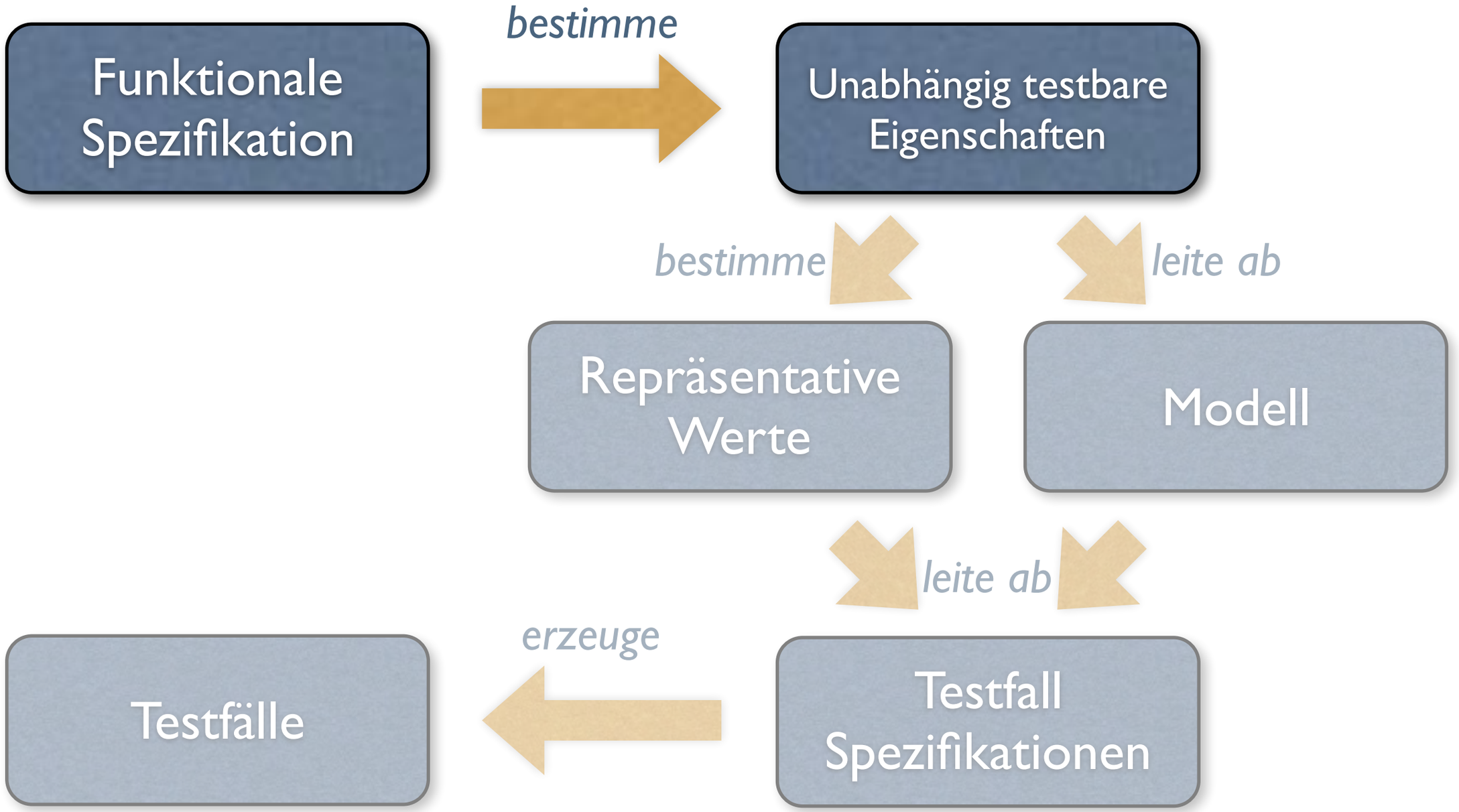
```
class Roots {
    // Löse ax² + bx + c = 0
    public roots(double a, double b, double c)
    { … }

    // Ergebnis: Werte für x
    double root_one, root_two;
}
```

- Was sind die unabhängig testbaren Eigenschaften?

Testbare Eigenschaften

- Beispiel: Multifunktionaler Taschenrechner
- Was sind die unabhängig testbaren Eigenschaften?

Testbare Eigenschaften



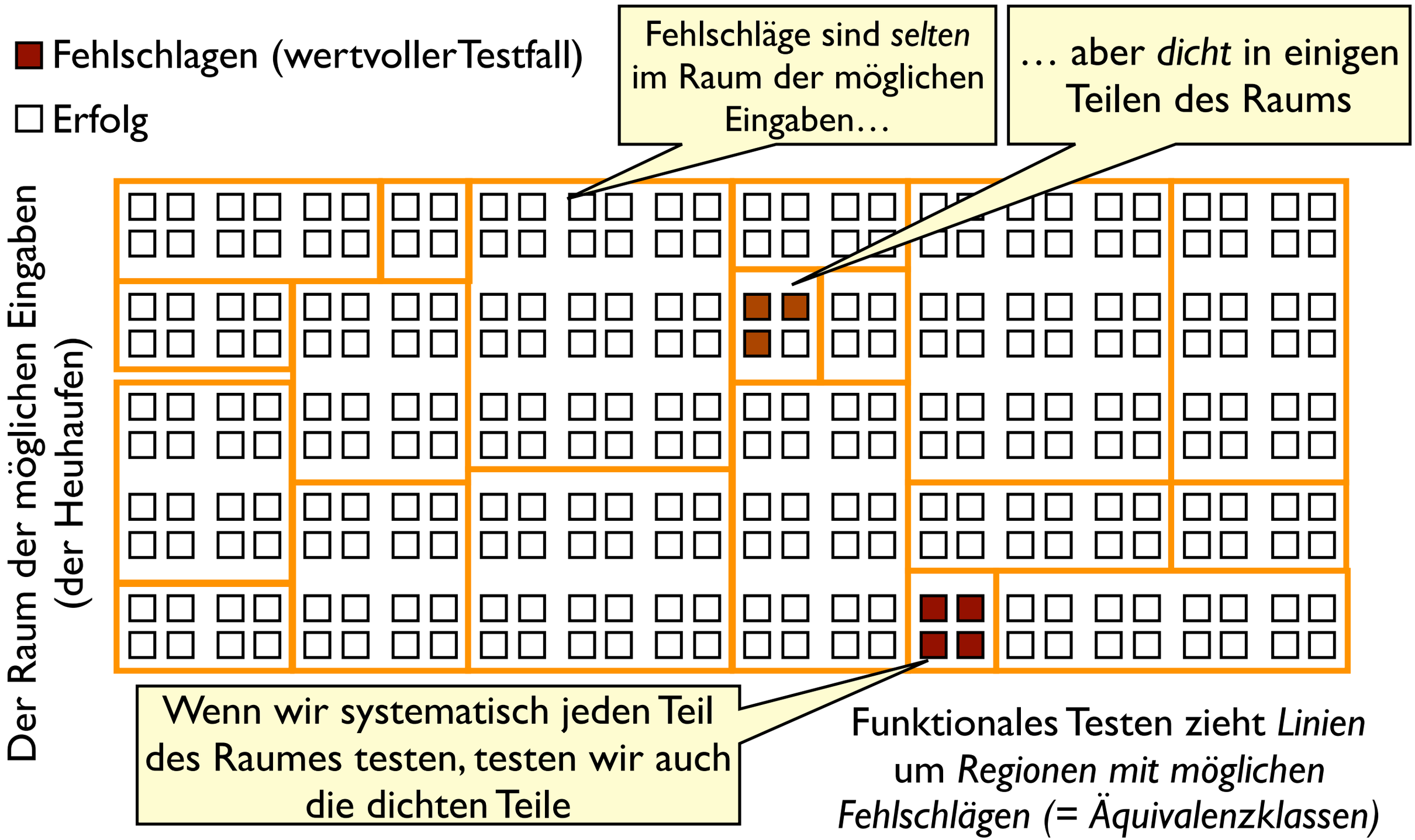
Repräsentative Werte

- Suche Eingaben, die *besonders wertvoll* sind
- Typischer-weise durch Wahl von *Repräsentanten* von *Äquivalenzklassen,* die womöglich oft oder gar nicht fehlschlagen

Nadeln im Heuhaufen

- Um Nadeln zu finden,
  müssen wir systematisch suchen
- Wir müssen herausfinden,
  was *Nadeln besonders macht*

Systematisches Partitionstesten

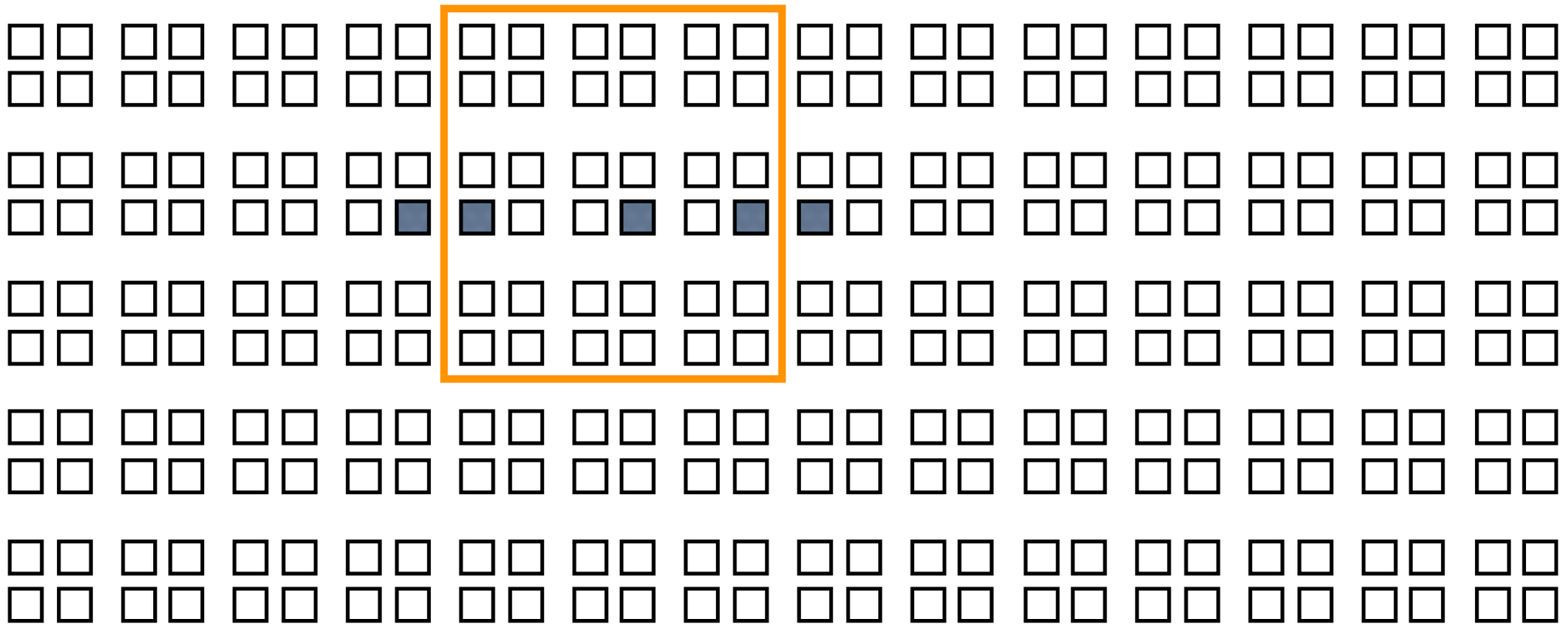


Äquivalenzklassen bilden

Eingabebedingung	Äquivalenzklassen
Bereich	eine gültig, zwei ungültig (größer und kleiner)
gegebener Wert	eine gültig, zwei ungültig (größer und kleiner)
Element einer Menge	eine gültig, eine ungültig
Boolean	eine gültig, eine ungültig

Grenzanalyse

- Möglicher Testfall



- Test am *unteren Ende* (1x gültig, 1x ungültig),
  am *oberen Ende* (1x gültig, 1x ungültig),
  und in der *Mitte*

Beispiel: Postleitzahl

- Eingabe:
  5-stellige Postleitzahl
- Ausgabe:
  Liste der Städte
- Welche repräsentativen Werte nutzen wir zum Testen?

Gültige Postleitzahlen

1. Mit 0 Städten als Ausgabe
   (0 ist Grenzwert)
2. Mit 1 Stadt als Ausgabe
3. Mit ≥2 Städten als Ausgabe

Ungültige Postleitzahlen

4. Leere Eingabe
5. 1–4 Zeichen
   (4 ist Grenzwert)
6. 6 Zeichen
   (6 ist Grenzwert)
7. sehr lange Eingabe
8. keine Ziffern
9. ungültige Zeichen

“Besondere” Postleitzahlen

- Wie wäre es mit einer Postleitzahl, die da lautet

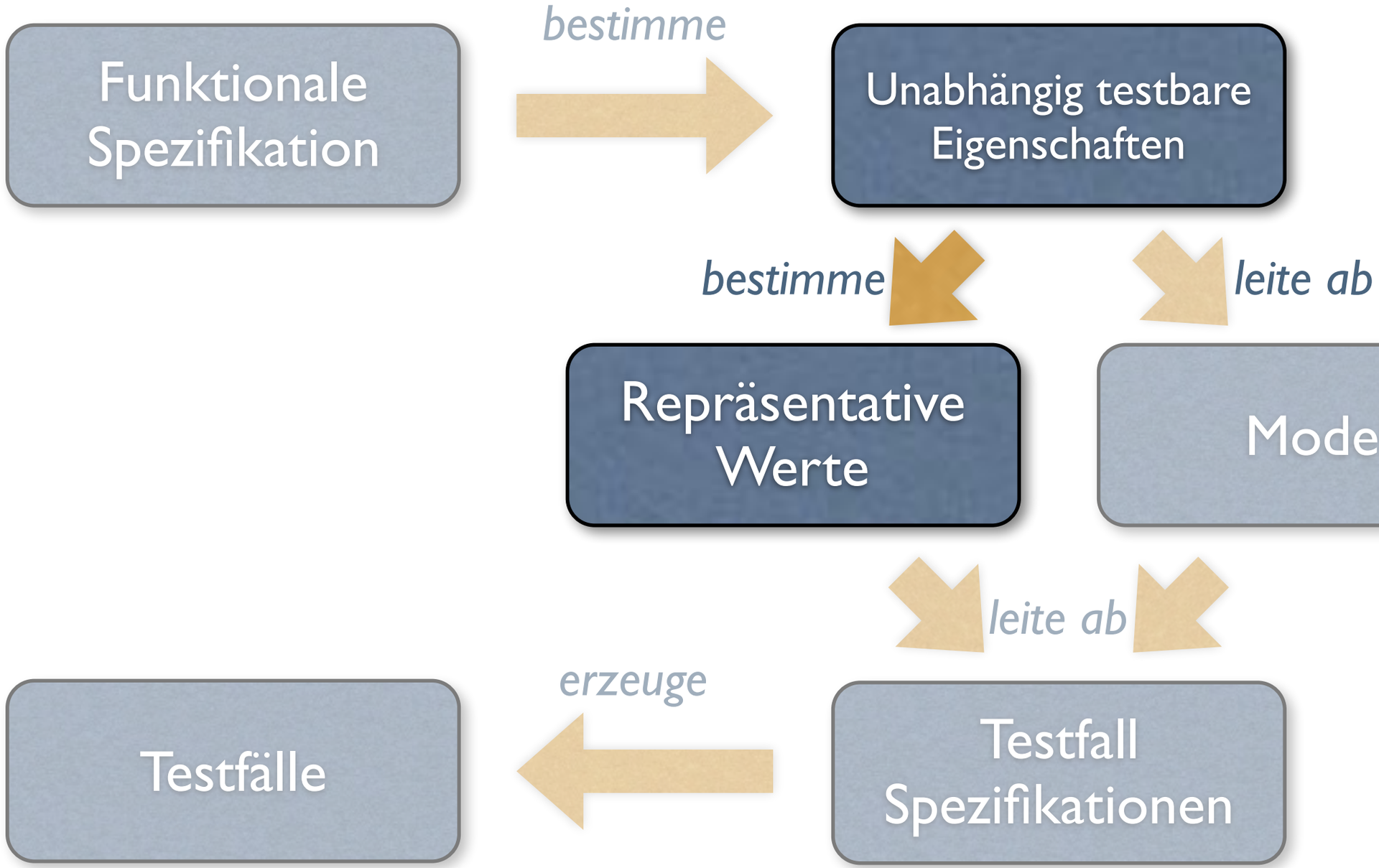
```
12345'; DROP TABLE orders; SELECT *
FROM zipcodes WHERE 'zip' = '
```

- Oder eine Postleitzahl aus 65536 Zeichen…
- Dies ist Sicherheitstesten!

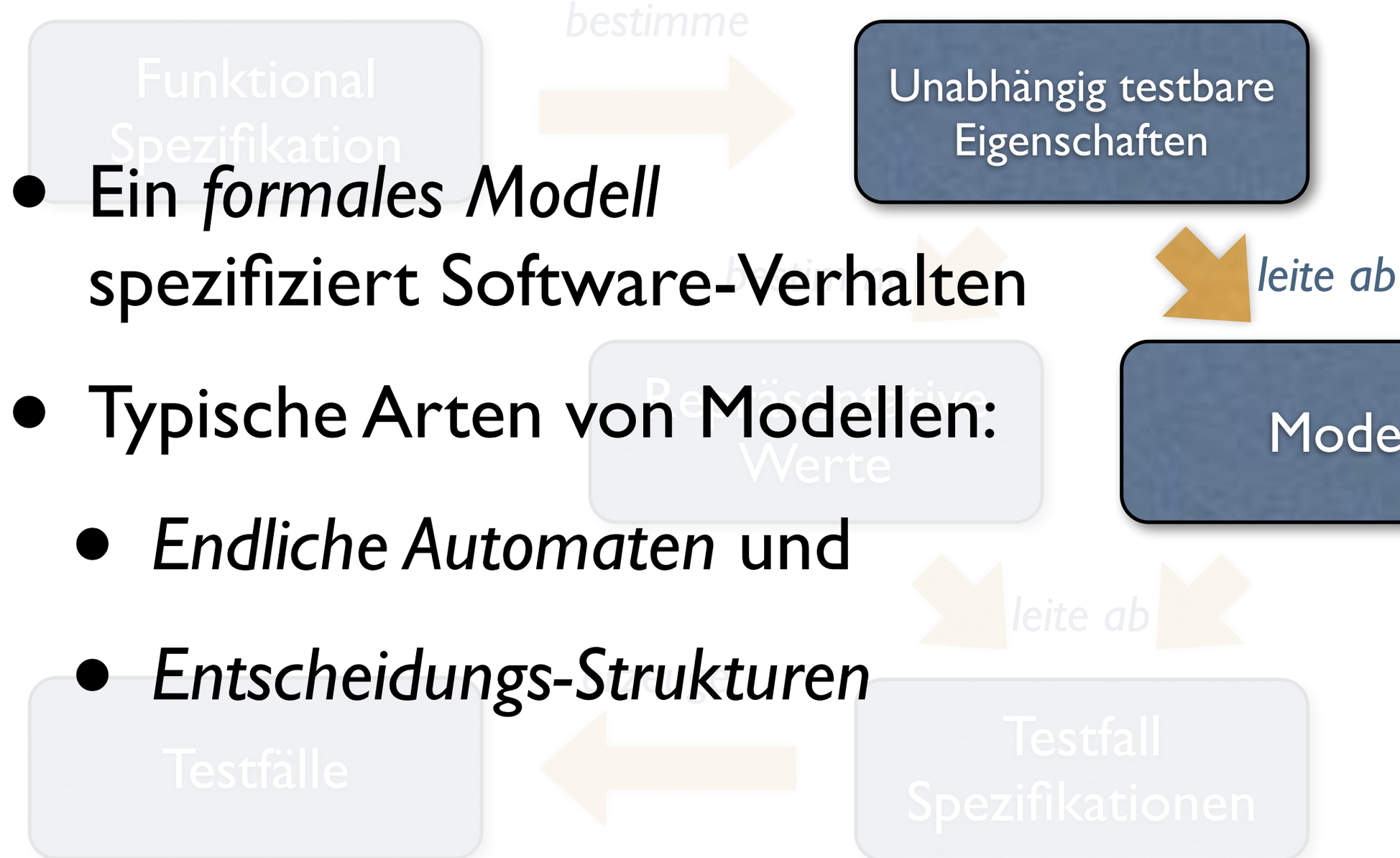
Gutjahrs Hypothese

Partitionstesten
ist effektiver
als Zufallstesten.

Repräsentative Werte

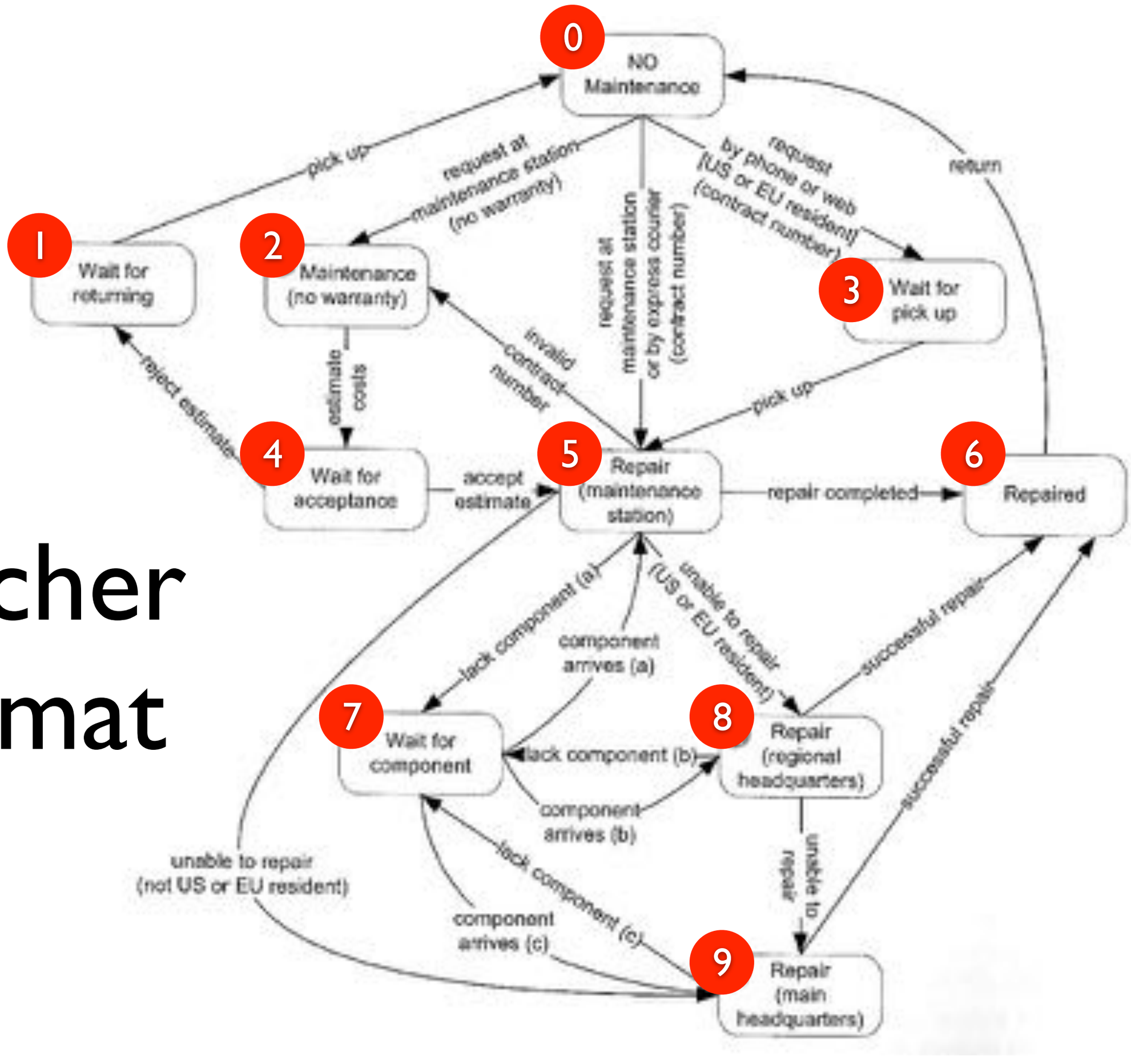


Modellbasiertes Testen



- Ein *formales Modell* spezifiziert Software-Verhalten
- Typische Arten von Modellen:
  - *Endliche Automaten* und
  - *Entscheidungs-Strukturen*

Endlicher Automat



Maintenance: The *Maintenance* function records the history of items undergoing maintenance.

If the product is covered by warranty or maintenance contract, maintenance can be requested either by calling the maintenance toll free number, or through the Web site, or by bringing the item to a designated maintenance station.

If the maintenance is requested by phone or Web site and the customer is a US or EU resident, the item is picked up at the customer site, otherwise, the customer shall ship the item with an express courier.

If the maintenance contract number provided by the customer is not valid, the item follows the procedure for items not covered by warranty.

If the product is not covered by warranty or maintenance contract, maintenance can be requested only by bringing the item to a maintenance station. The maintenance station informs the customer of the estimated costs for repair. Maintenance starts only when the customer accepts the estimate. If the customer does not accept the estimate, the product is returned to the customer.

Small problems can be repaired directly at the maintenance station. If the maintenance station cannot solve the problem, the product is sent to the maintenance regional headquarters (if in US or EU) or to the maintenance main headquarters (otherwise).

If the maintenance regional headquarters cannot solve the problem, the product is sent to the maintenance main headquarters.

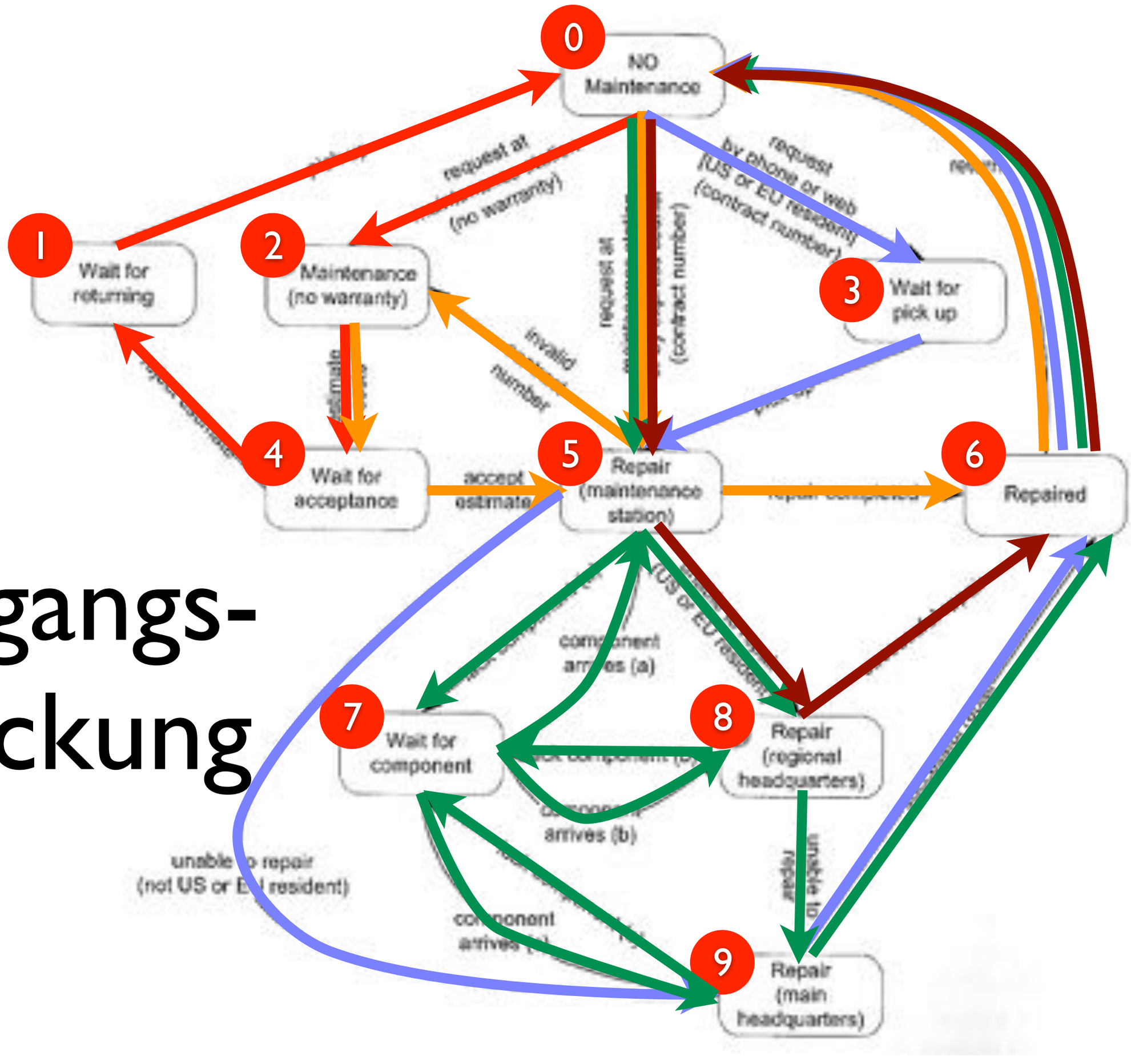
Maintenance is suspended if some components are not available.

Once repaired, the product is returned to the customer.

Abdeckungskriterien

- *Pfad-Abdeckung:* Tests decken jeden Pfad ab
  In der Praxis nicht möglich, da unendlich viele Pfade
- *Zustands-Abdeckung:* Jeder *Knoten* wird erreicht
  Mindest-Kriterium zum Testen
- *Übergangs-Abdeckung:* Jede *Kante* wird erreicht
  Sinnvolles und gewöhnlich erfüllbares Abdeckungs-Kriterium

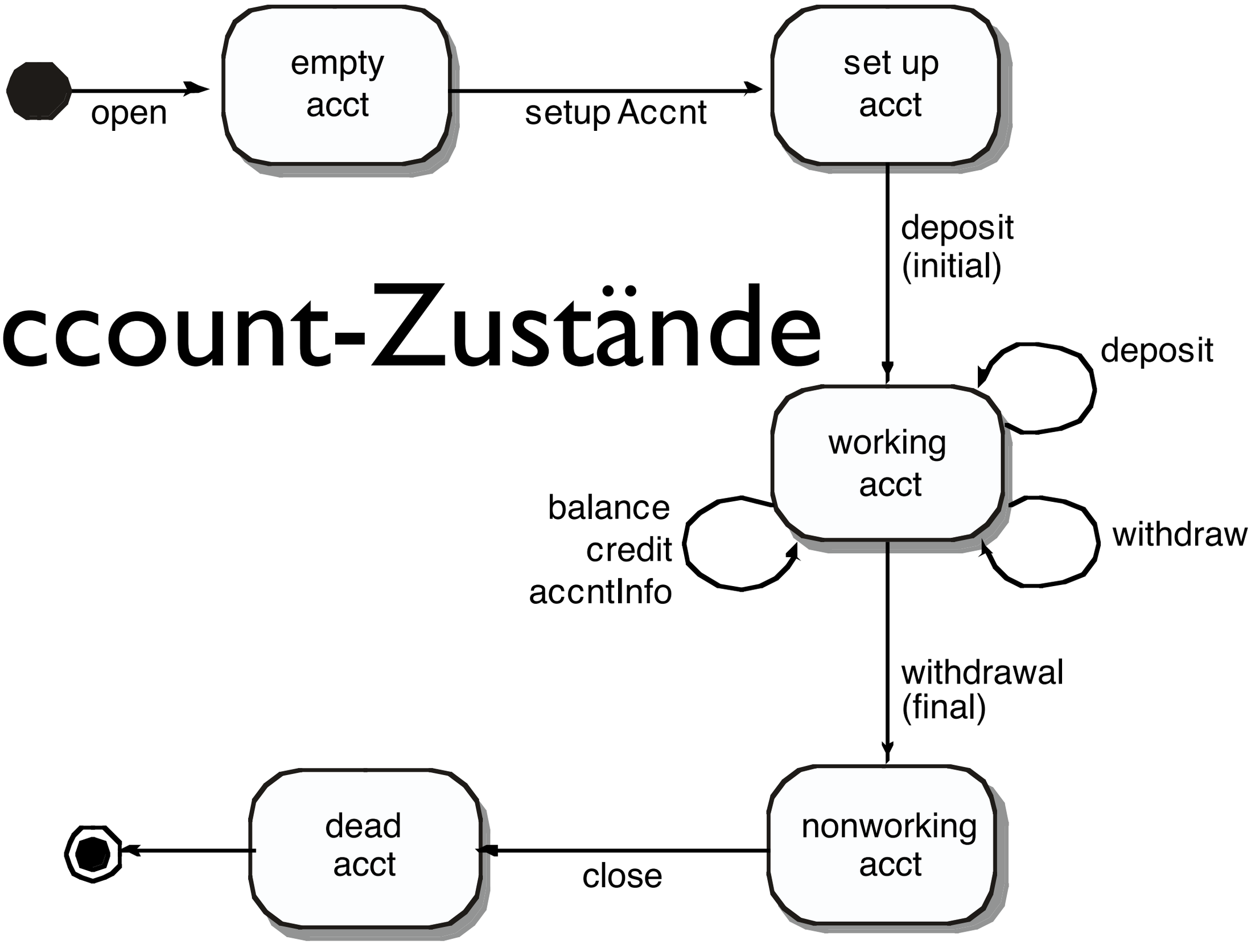
Übergangs-Abdeckung



Zustandsbasiertes Testen

- Protokolle (z.B. Netzwerke)
- GUIs (Folgen von Benutzeraktionen)
- Objekte (Methoden und Zustände)

Account-Zustände



Entscheidungstabelle

	Studenten		Andere Kunden					
Studentenkonto	T	T	F	F	F	F	F	F
Aktueller Kauf > Rabattgrenze 1	–	–	F	F	T	T	–	–
Aktueller Kauf > Rabattgrenze 2	–	–	–	–	F	F	T	T
Sonderpreis < Rabattierter Preis	F	T	F	T	–	–	–	–
Sonderpreis < Stufe 1	–	–	–	–	F	T	–	–
Sonderpreis < Stufe 2	–	–	–	–	–	–	F	T
Ergebnis	Edu Rabatt	Sonderpreis	Kein Rabatt	Sonder Preis	Stufe 1 Rabatt	Sonderpreis	Stufe 2 Rabatt	Sonderpreis

Bedingungsabdeckung

- *Grundkriterium:* Jede Spalte testen
  "Don't care"-Einträge (–) können beliebige Werte annehmen
- *Kombinationskriterium:* Jede Kombination testen
  Benötigt 2^n Tests bei n Bedingungen und ist unrealistisch
- *Abgeschwächtes Kombinationskriterium (MCDC):*
  Wie Grundkriterium, aber jeder T/F-Wert muss mindestens einmal geändert werden, so dass sich das Ergebnis ändert
  Gutes und erreichbares Abdeckungs-Kriterium

MCDC Kriterium

	Studenten		Andere Kunden					
Studentenkonto	F	T	F	F	F	F	F	F
Aktueller Kauf > Rabattgrenze 1	–	–	F	F	T	T	–	–
Aktueller Kauf > Rabattgrenze 2	–	–	–	–	F	F	T	T
Sonderpreis < Rabattierter Preis	F	T	F	T	–	–	–	–
Sonderpreis < Stufe 1	–	–	–	–	F	T	–	–
Sonderpreis < Stufe 2	–	–	–	–	–	–	F	T
Ergebnis	Edu Rabatt	Sonderpreis	Kein Rabatt	Sonder Preis	Stufe 1 Rabatt	Sonderpreis	Stufe 2 Rabatt	Sonderpreis

MCDC Kriterium

	Studenten		Andere Kunden					
Studentenkonto	T	T	F	F	F	F	F	F
Aktueller Kauf > Rabattgrenze 1	–	–	F	F	T	T	–	–
Aktueller Kauf > Rabattgrenze 2	–	–	–	–	F	F	T	T
Sonderpreis < Rabattierter Preis	T	T	F	T	–	–	–	–
Sonderpreis < Stufe 1	–	–	–	–	F	T	–	–
Sonderpreis < Stufe 2	–	–	–	–	–	–	F	T
Ergebnis	Edu Rabatt	Sonderpreis	Kein Rabatt	Sonder Preis	Stufe 1 Rabatt	Sonderpreis	Stufe 2 Rabatt	Sonderpreis

MCDC Kriterium

	Studenten		Andere Kunden					
Studentenkonto	T	F	F	F	F	F	F	F
Aktueller Kauf > Rabattgrenze 1	–	–	F	F	T	T	–	–
Aktueller Kauf > Rabattgrenze 2	–	–	–	–	F	F	T	T
Sonderpreis < Rabattierter Preis	F	T	F	T	–	–	–	–
Sonderpreis < Stufe 1	–	–	–	–	F	T	–	–
Sonderpreis < Stufe 2	–	–	–	–	–	–	F	T
Ergebnis	Edu Rabatt	Sonderpreis	Kein Rabatt	Sonder Preis	Stufe 1 Rabatt	Sonderpreis	Stufe 2 Rabatt	Sonderpreis

MCDC Kriterium

	Studenten		Andere Kunden					
Studentenkonto	T	T	F	F	F	F	F	F
Aktueller Kauf > Rabattgrenze 1	–	–	F	F	T	T	–	–
Aktueller Kauf > Rabattgrenze 2	–	–	–	–	F	F	T	T
Sonderpreis < Rabattierter Preis	F	F	F	T	–	–	–	–
Sonderpreis < Stufe 1	–	–	–	–	F	T	–	–
Sonderpreis < Stufe 2	–	–	–	–	–	–	F	T
Ergebnis	Edu Rabatt	Sonderpreis	Kein Rabatt	Sonder Preis	Stufe 1 Rabatt	Sonderpreis	Stufe 2 Rabatt	Sonderpreis

Weyukers Hypothese

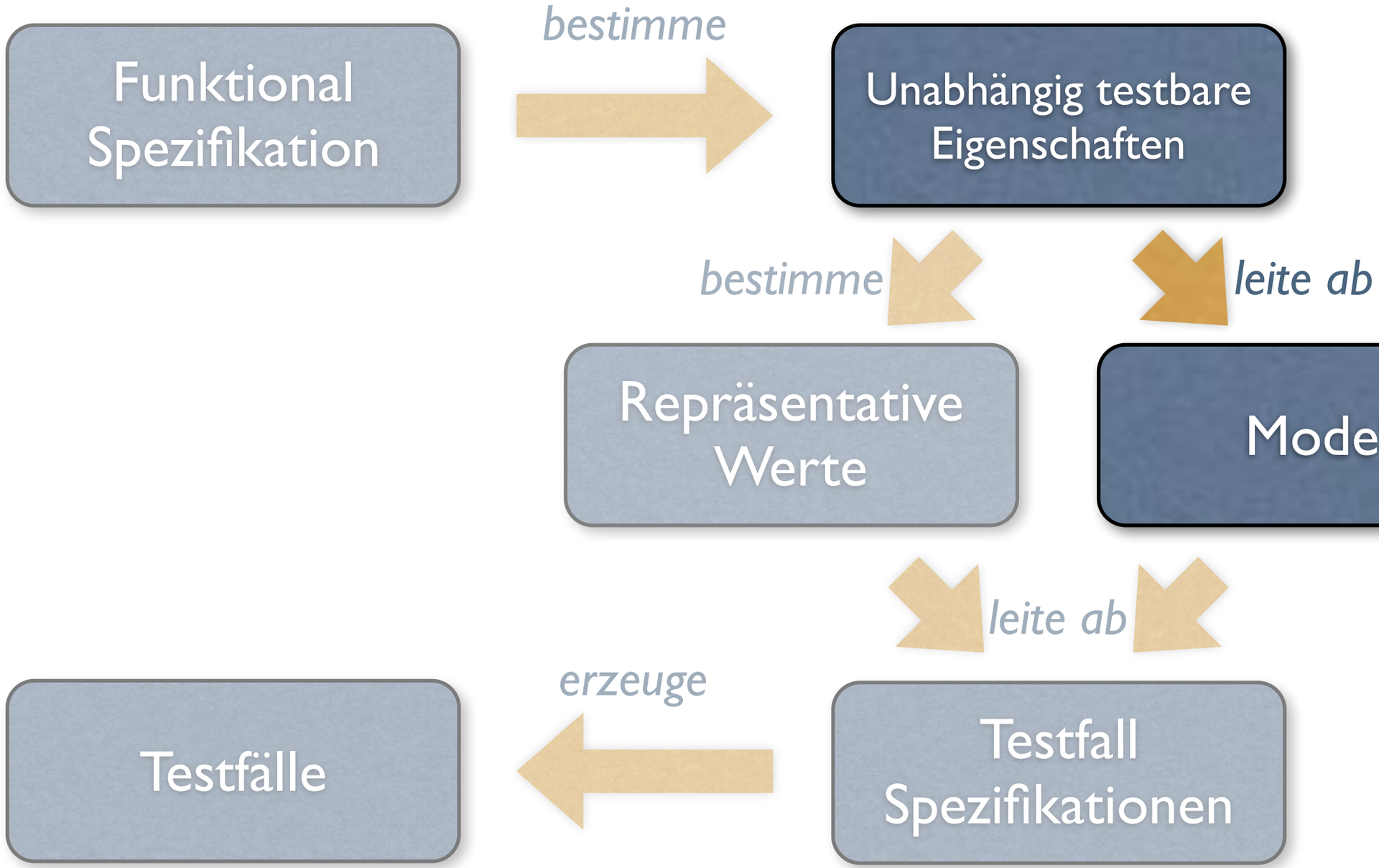
Die Angemessenheit eines
Abdeckungs-Kriteriums
kann nur intuitiv definiert werden.

Aus der Geschichte lernen

Paretos Gesetz

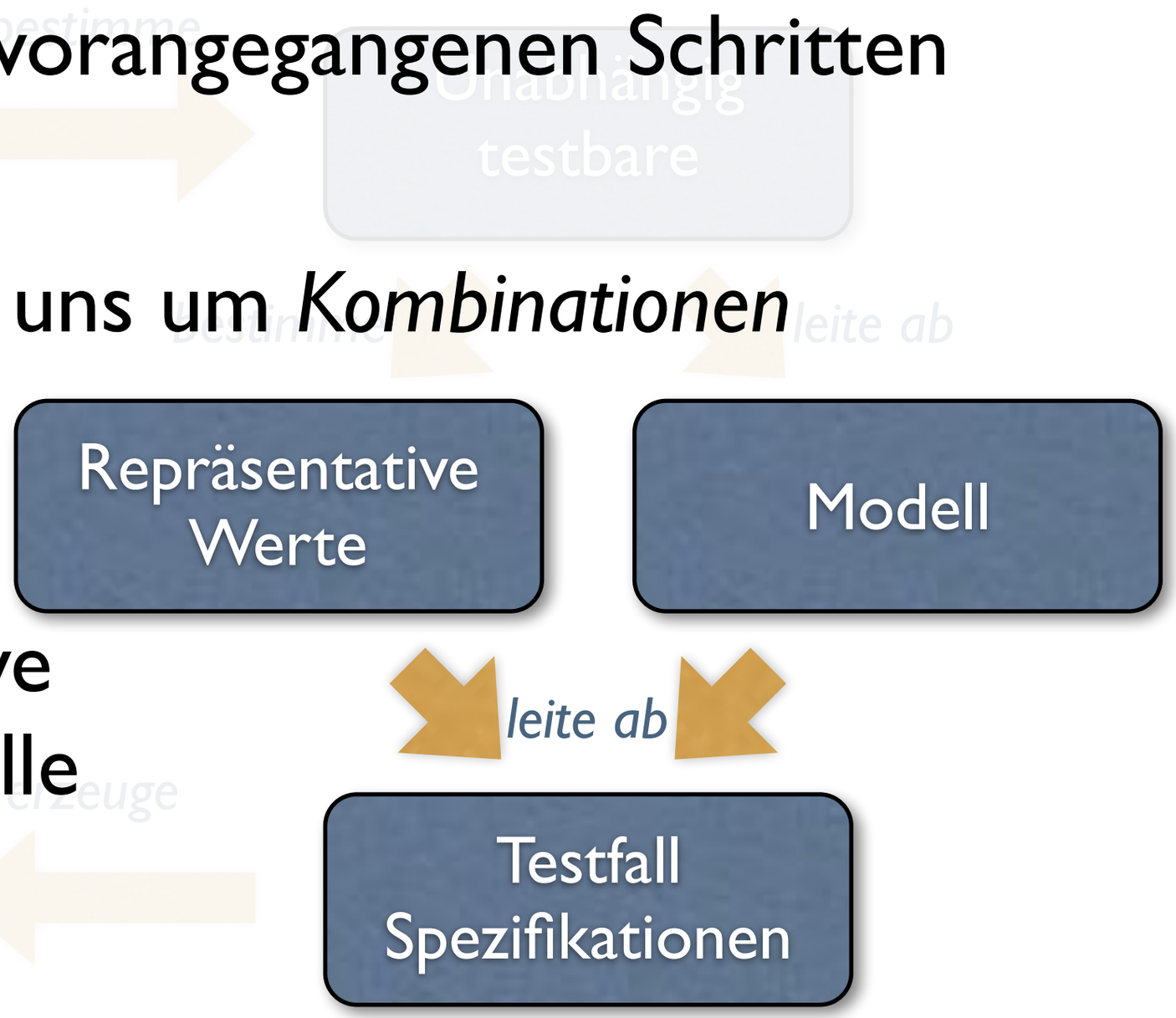
Etwa 80% der Fehler
stecken in 20% der Module

Modellbasiertes Testen

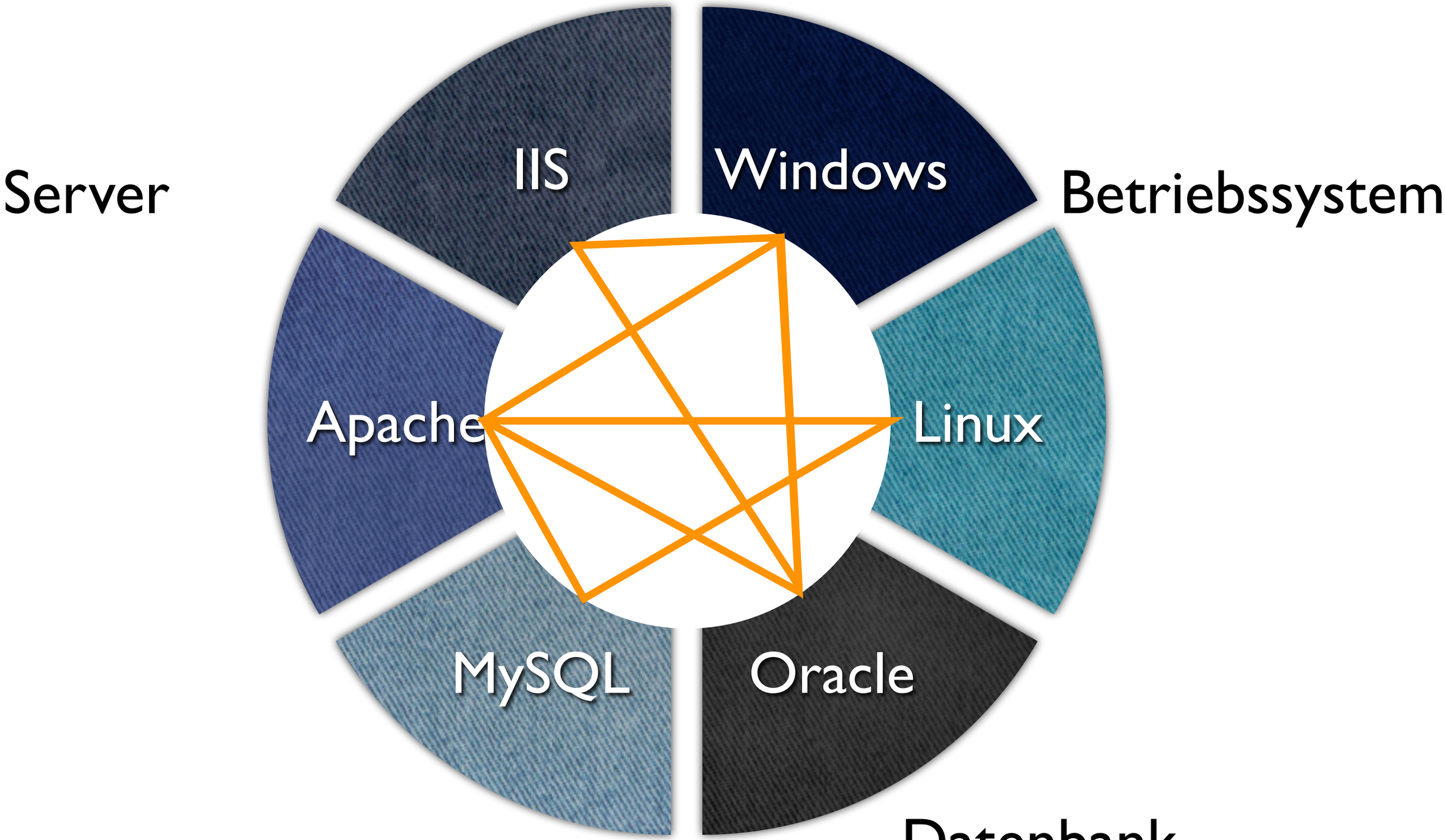


Testfall-Spezifikationen ableiten

- Eingabewerte in vorangegangenen Schritten aufgezählt
- Jetzt müssen wir uns um *Kombinationen* kümmern
- Nutze Modelle *und* repräsentative Werte, um Testfälle abzuleiten



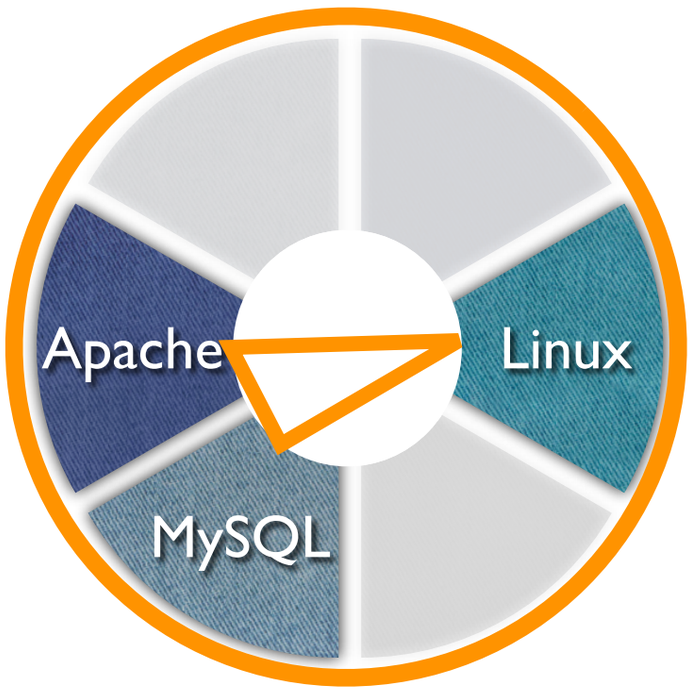
Kombinatorisches Testen

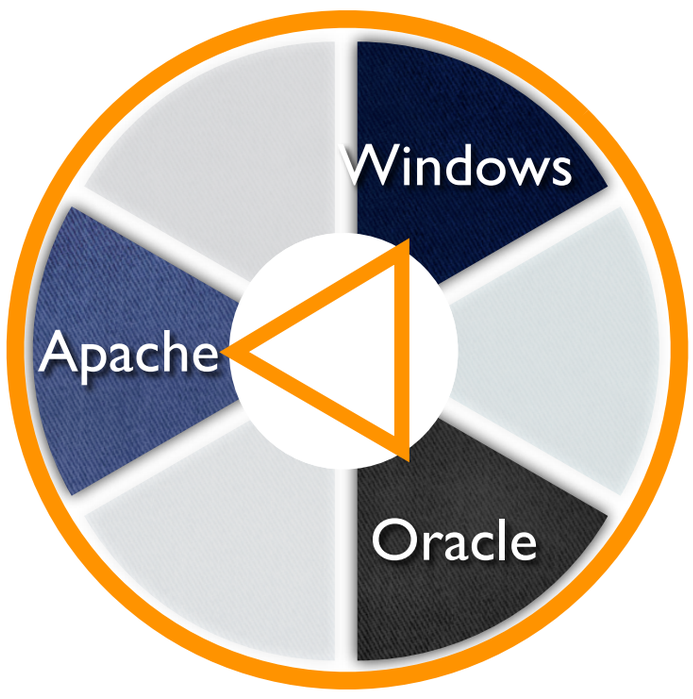


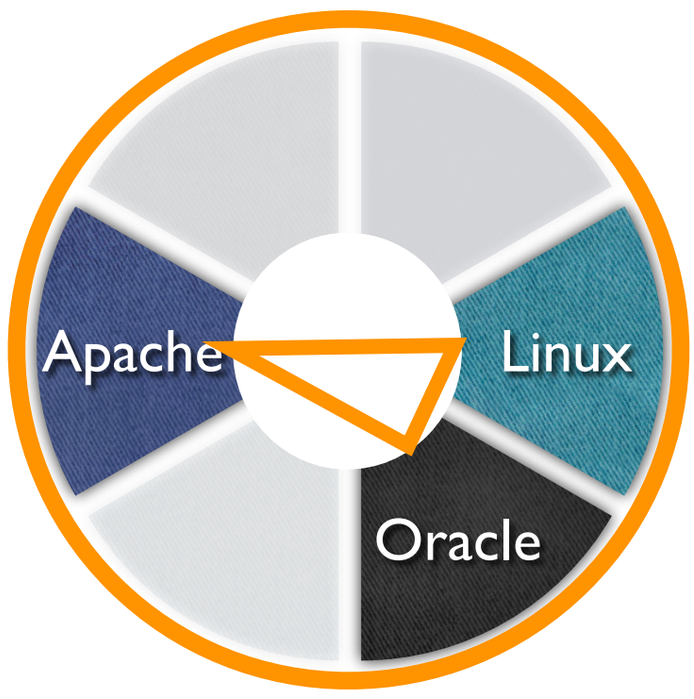
Kombinationstesten

- *Ungültige* Kombinationen ausschließen
  IIS etwa läuft nur auf Windows
- *Alle Paare* aus Kombinationen abdecken
  wie etwa MySQL auf Windows und Linux
- Kombinationen gewöhnlich *automatisch generiert*
  und – hoffentlich – auch automatisch getestet

Paarweises Testen



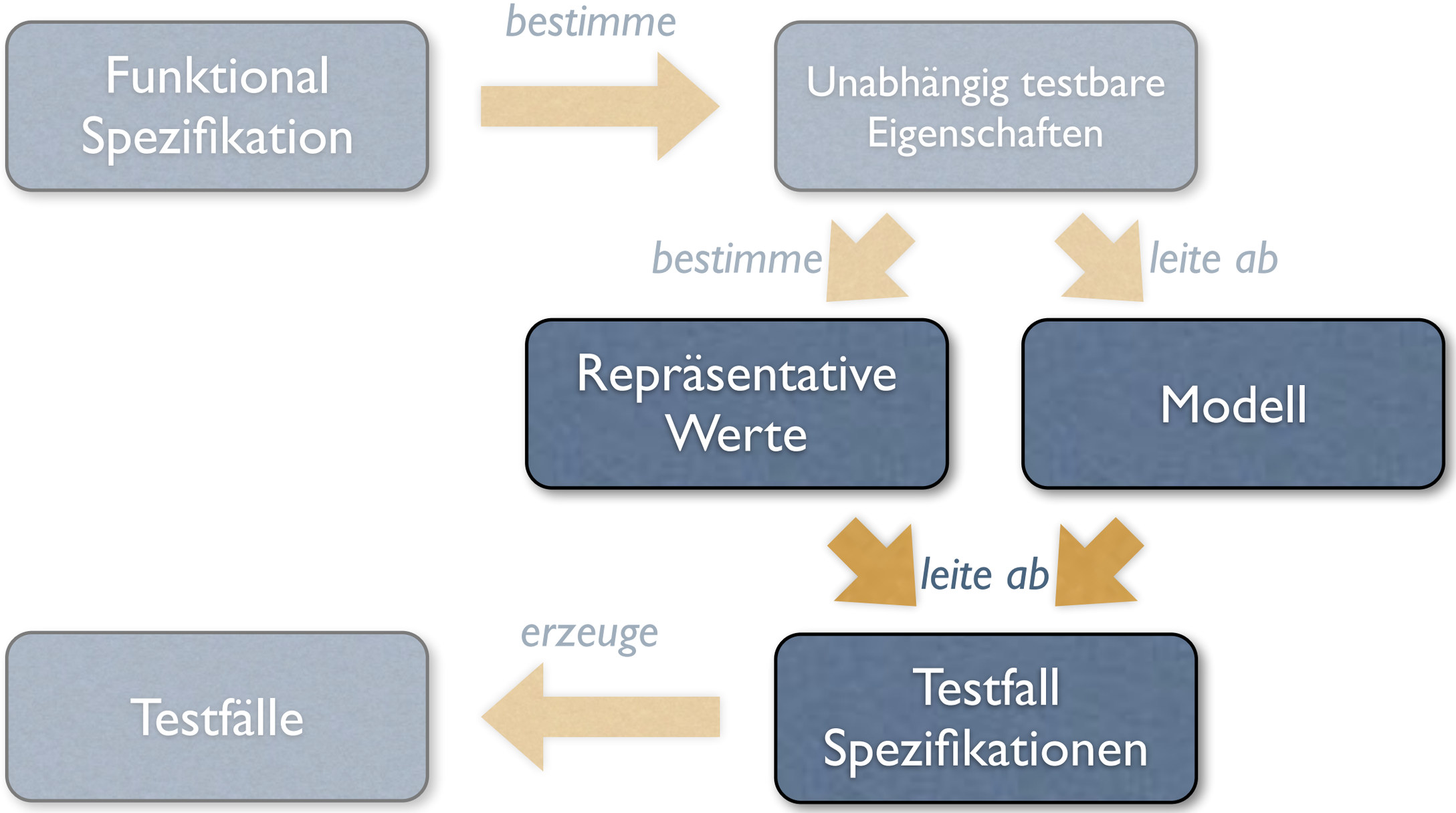




Testumgebung

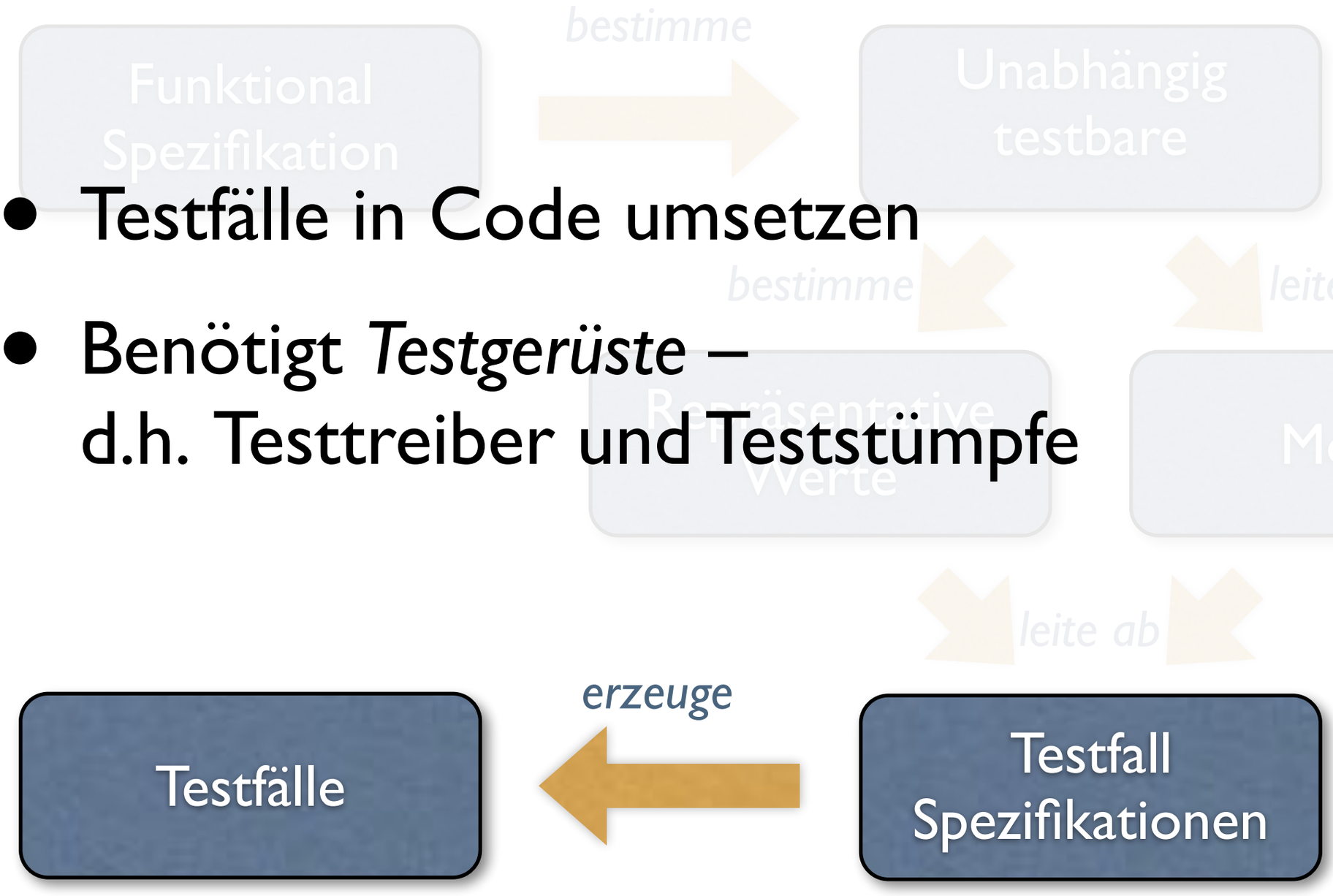
- Millionen von Konfigurationen
- Getestet auf Dutzenden verschiedener Maschinen
- Alles, um Fehler zu finden und zu reproduzieren

Testfall-Spezifikationen ableiten

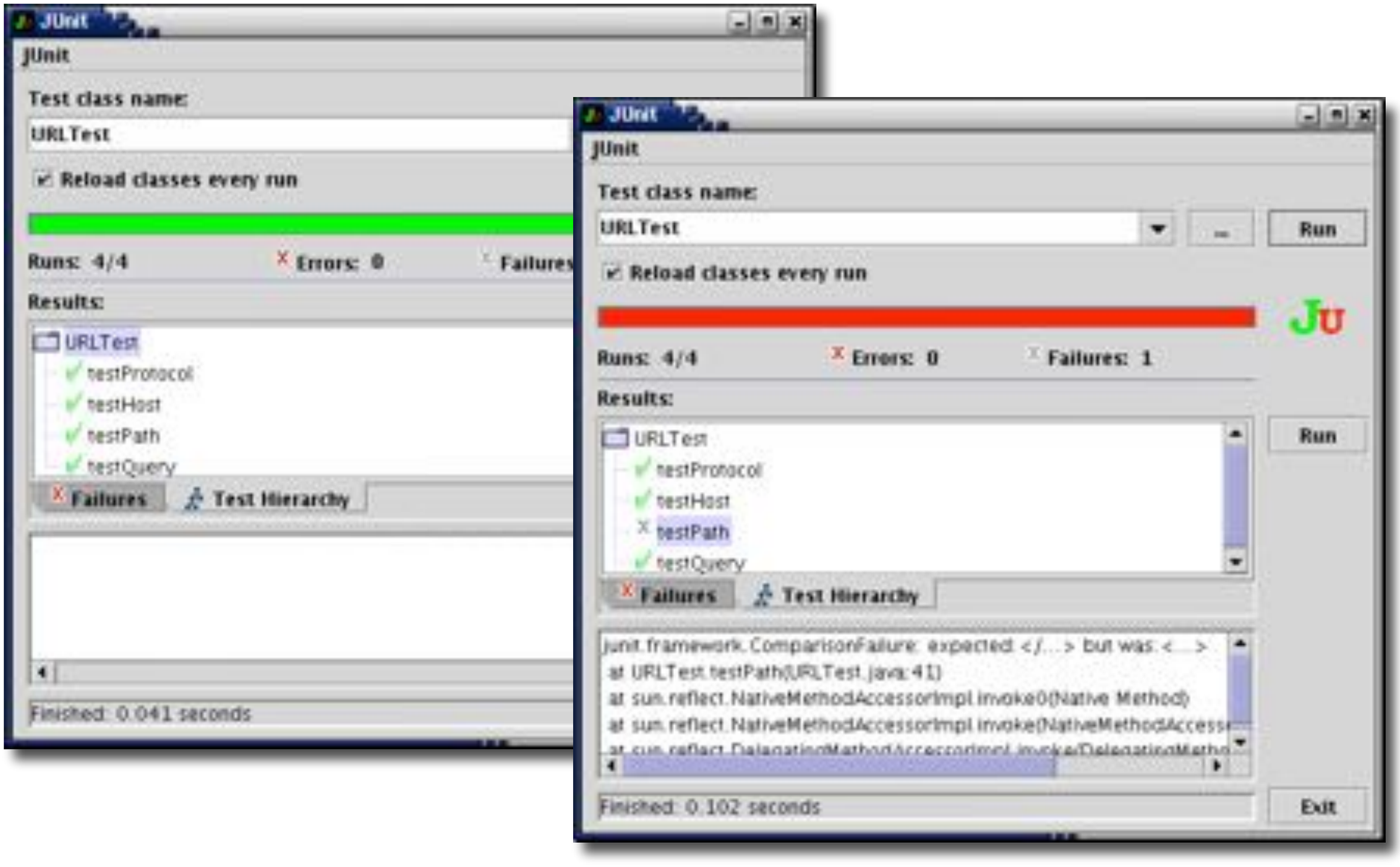


Testfälle ableiten

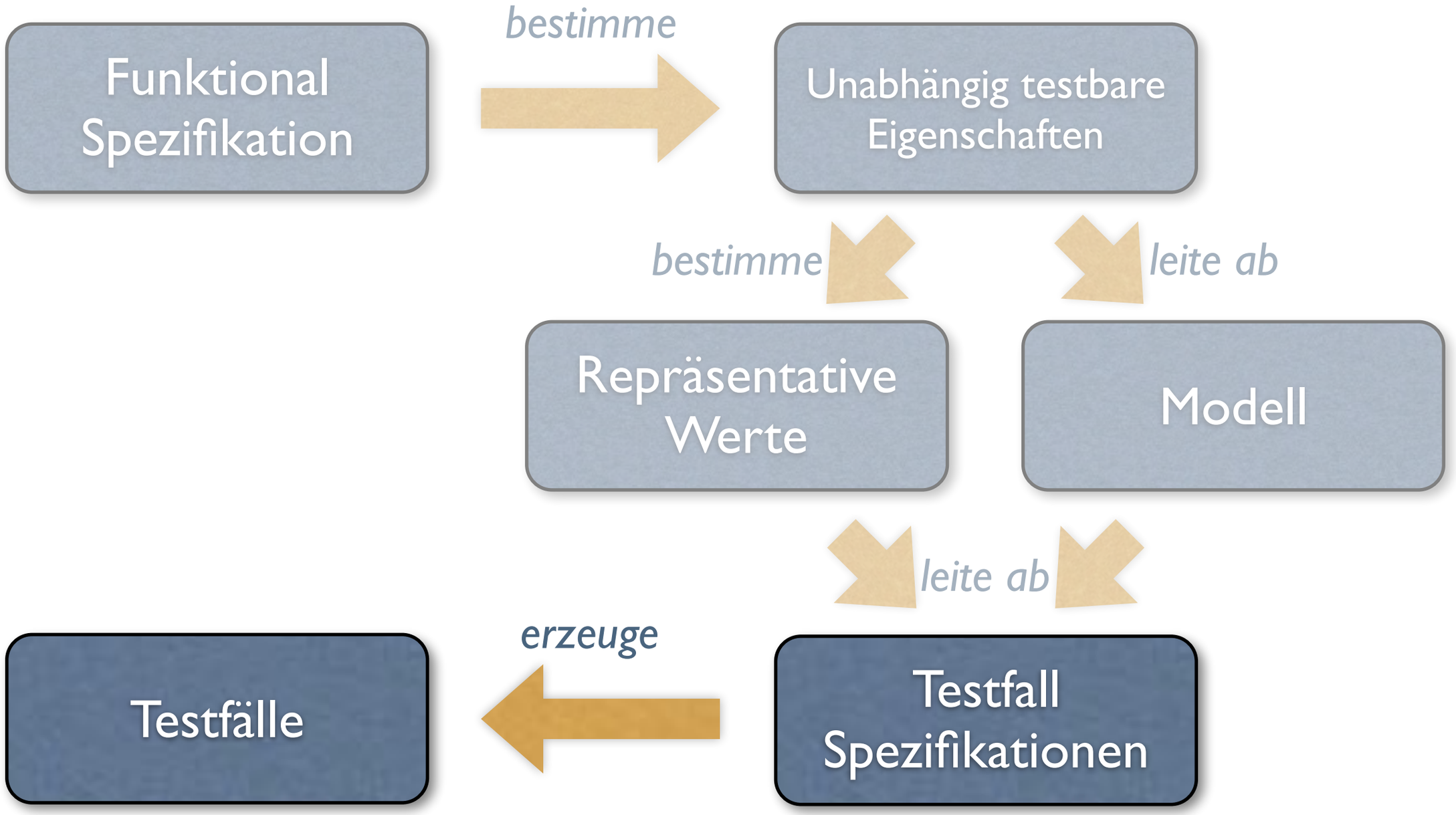
- Testfälle in Code umsetzen
- Benötigt *Testgerüste* –
  d.h. Testtreiber und Teststümpfe



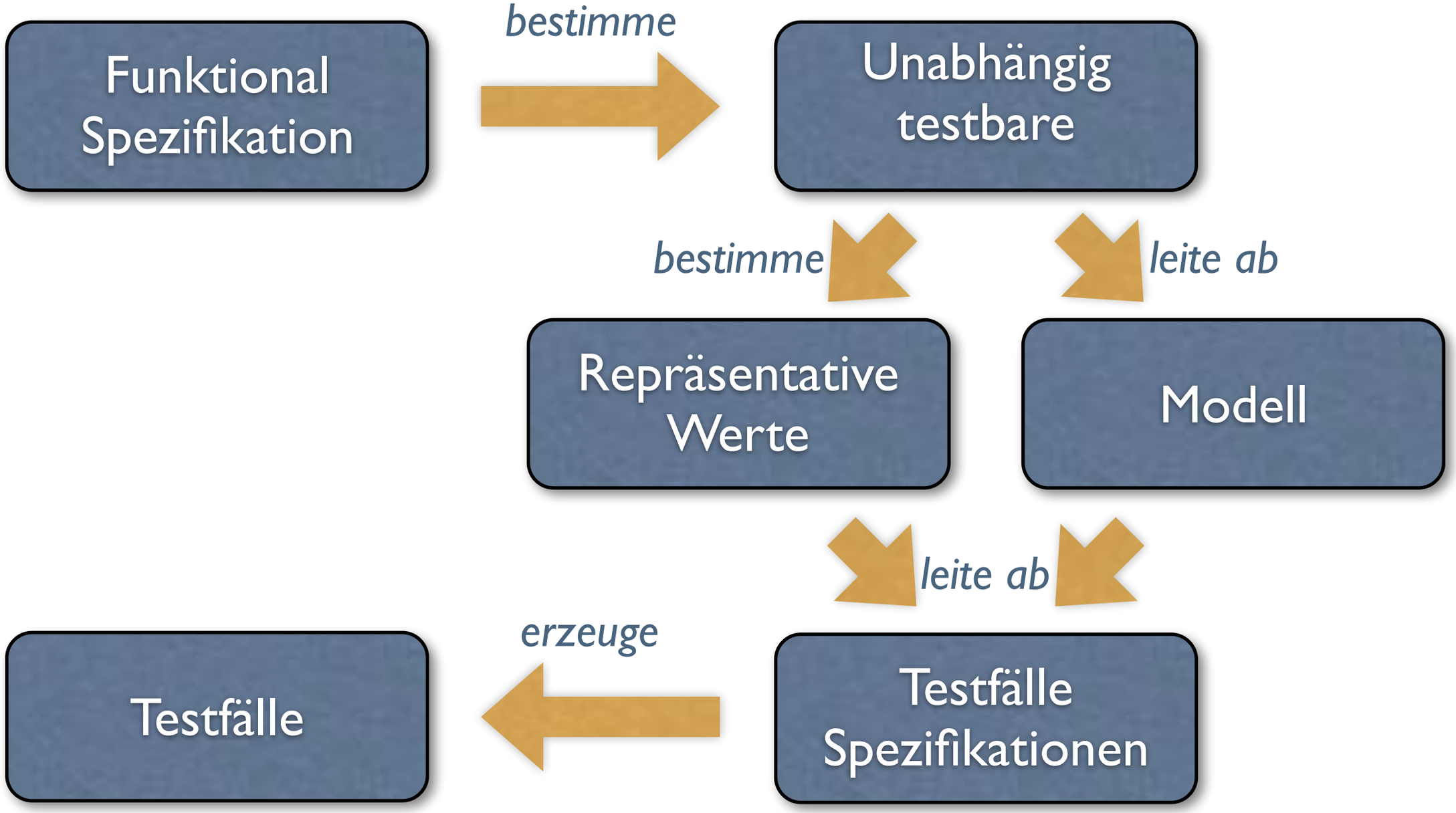
JUnit


JUnit
JUnit
Test class name:
URLTest
Reload classes every run
Runs: 4/4
Errors: 0
Failures
Results:
URLTest
testProtocol
testHost
testPath
testQuery
Failures
Test Hierarchy
Finished: 0.041 seconds
JUnit
JUnit
Test class name:
URLTest
...
Run
Reload classes every run
Runs: 4/4
Errors: 0
Failures: 1
Results:
URLTest
testProtocol
testHost
testPath
testQuery
Run
Failures
Test Hierarchy
junit.framework.ComparisonFailure: expected:</...> but was:<...>
at URLTest.testPath(URLTest.java:41)
at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccess
Finished: 0.102 seconds
Exit

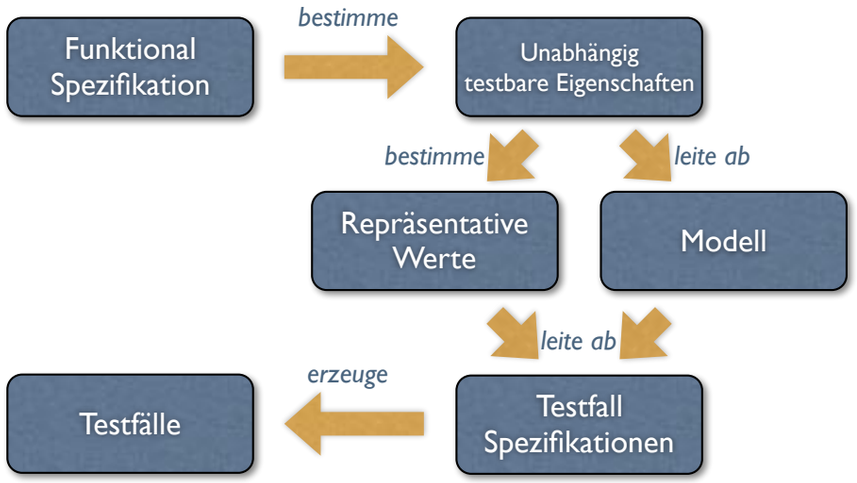
Testfälle ableiten



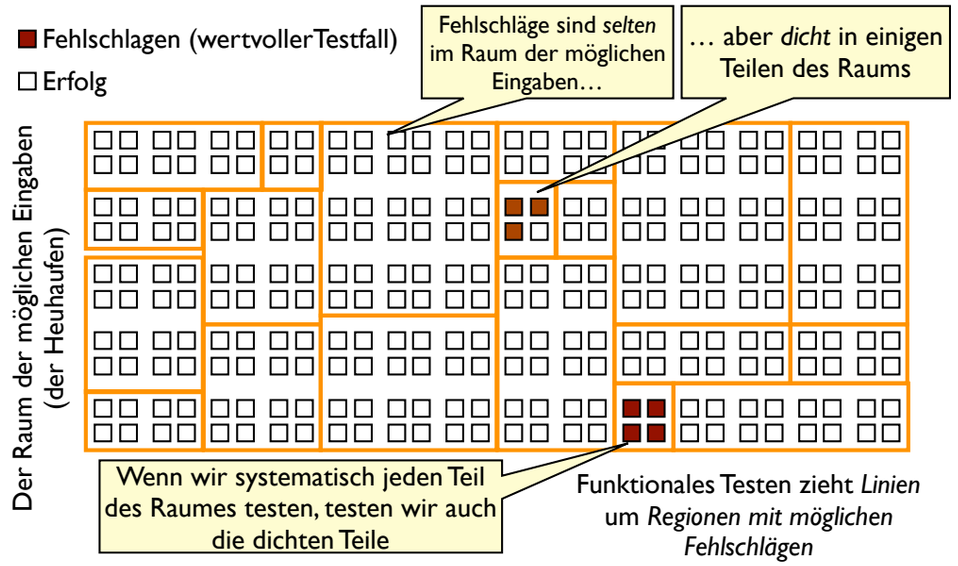
Systematisches funktionales Testen



Systematisches funktionales Testen



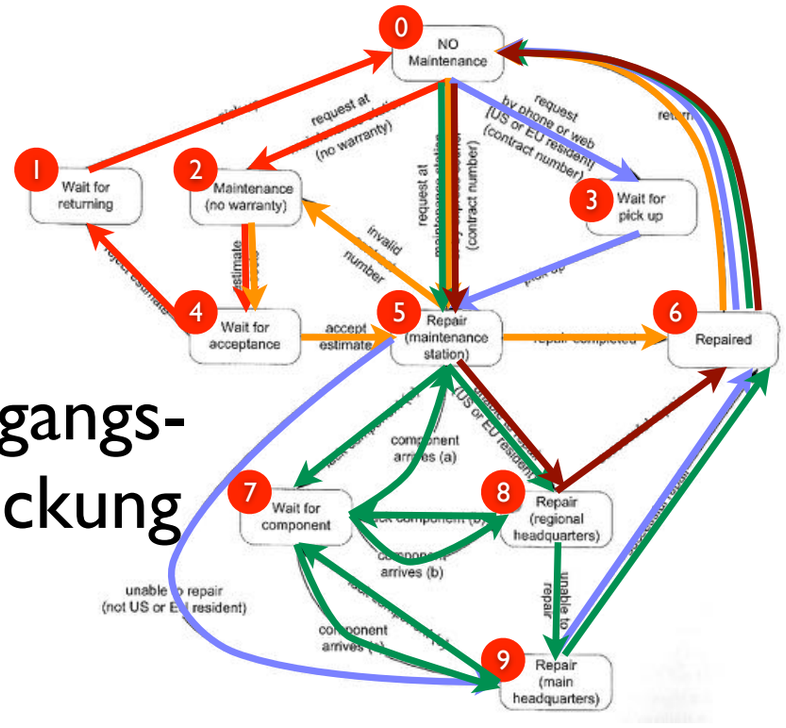
Systematisches Partitionstesten



Test-Taktiken

- auf *Spezifikation*
- el
- rhalten
- Tests basie
- Test deckt *implementie* wie möglic

Übergangs-Abdeckung



MCDC Kriterium

	Studenten		Andere Kunden					
Studentenkonto	F	T	F	F	F	F	F	F
Aktueller Kauf > Rabattgrenze 1	–	–	F	F	T	T	–	–
Aktueller Kauf > Rabattgrenze 2	–	–	–	–	F	F	T	T
Sonderpreis < Rabattierter Preis	F	T	F	T	–	–	–	–
Sonderpreis < Stufe 1	–	–	–	–	F	T	–	–
Sonderpreis < Stufe 2	–	–	–	–	–	–	F	T
Ergebnis	Edu Rabatt	Sonderpreis	Kein Rabatt	Sonder Preis	Stufe 1 Rabatt	Sonderpreis	Stufe 2 Rabatt	Sonderpreis