Humboldt-Universität zu Berlin

Mensch-Technik-Interaktion

Arbeitsgruppe
Softwaretechnik
(Institut für Informatik)

Arbeitsgruppe
Ingeneurpsychologie
(Institut für Psychologie)

Unit Testing, SUnit & You

M. Hildebrandt

Agenda

- **Unit Testing**
  - Begriffsklärung
  - Guidelines
- **SUnit**
  - Geschichte
  - Framework
  - Testschritte

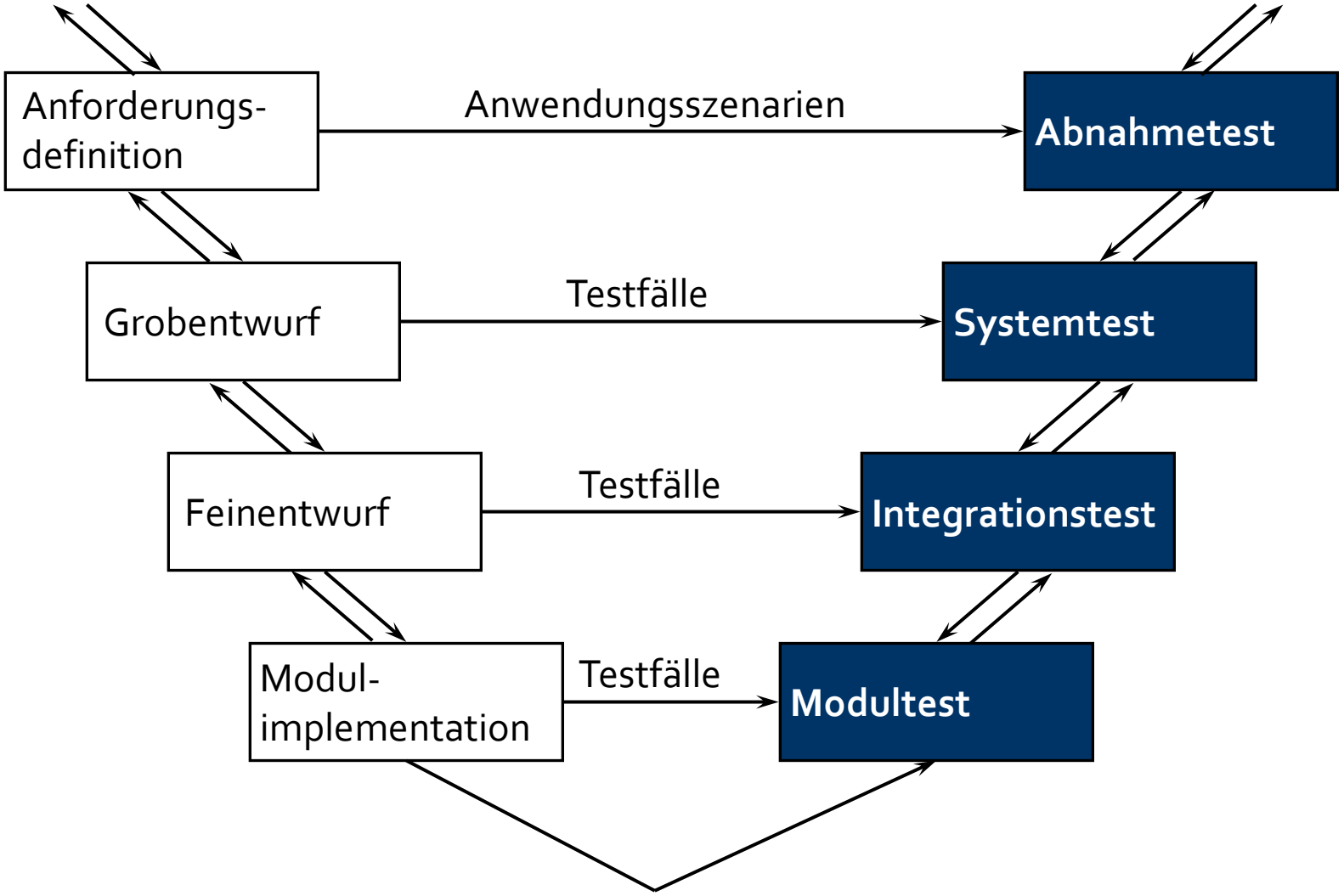
Begriff: Testen

- **Vorgehen**
  - Anwenden des Testlings auf definierte Eingabewerte
  - Vergleich der Ergebnisse mit Erwartungswerten

- **Ziel**
  - Finden von Fehlern
  - Nicht: Nachweis von Korrektheit

Absence of proof is NOT proof of absence.

Unit Testing im Entwicklungsprozess



Unit & Integration Testing

- **Unit Testing**
  - bezieht sich **auf einzelne Komponenten** eines Systems
    - **Komponenten:** Module, Prozeduren, Klassen, Methoden
  - **idealerweise:** unabhängiger Test der Komponenten

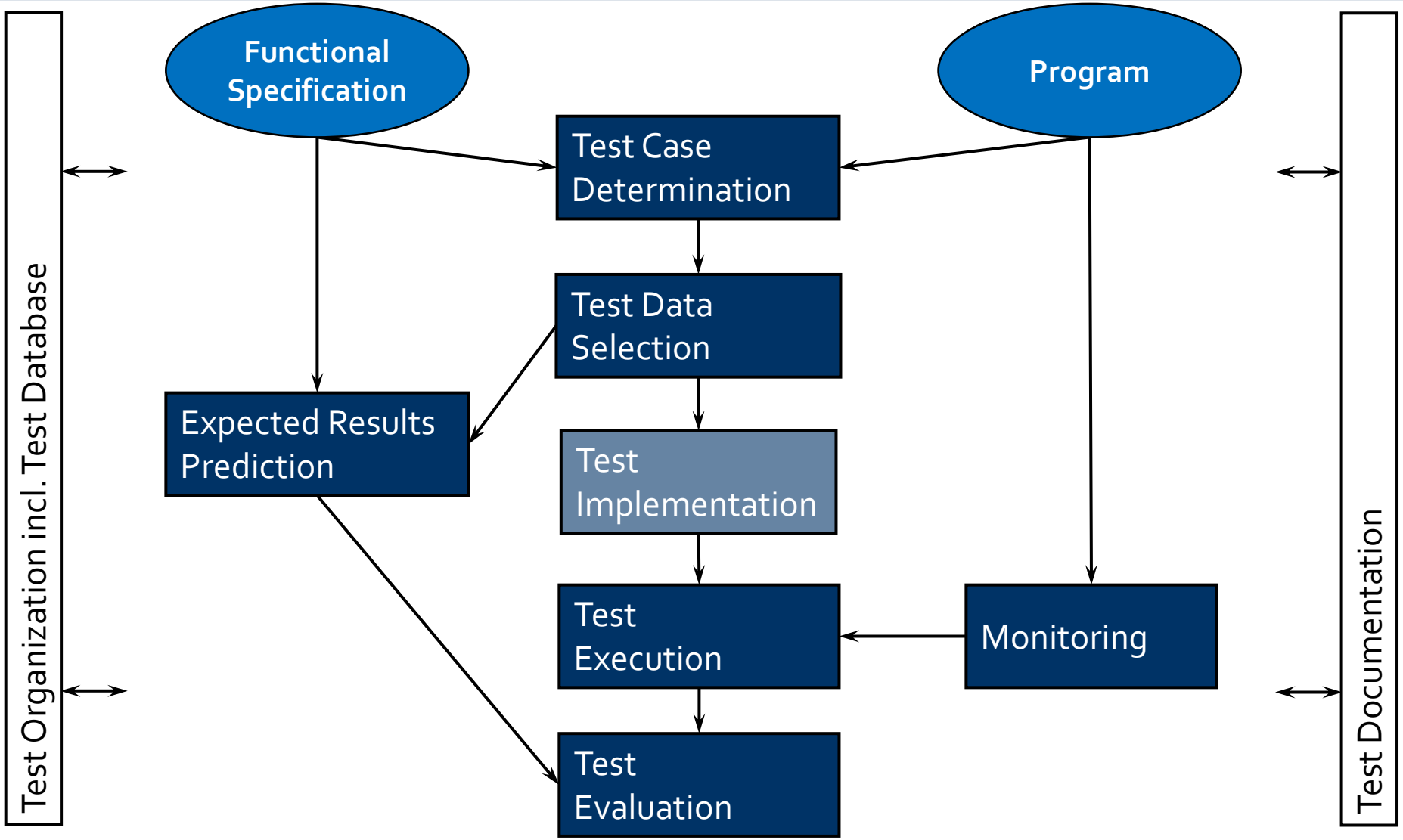
- **Integration Testing**
  - bezieht sich auf das **Zusammenspiel** mehrerer Komponenten bezieht
  - **idealerweise:** Komponenten bereits einzeln getestet

Unit Testing: Unabhängigkeit

- **Problem: Abhängigkeiten**
  - Komponenten verwenden oft Dienste von anderen Komponenten
  - Unit Testing erfordert aufwendiges Setup
  - Unabhängigkeit nicht gegeben

- **Lösung: Substitution**
  - **Stub:**
    - Platzhalter für geplante aber noch nicht implementierte Funktionalität
  - **Dummy:**
    - Ersatz für die „echte" Implementierung für Testzwecke
    - simuliert die echte Implementierung, meist stark reduziert
  - **Mock:**
    - Dummy, welcher zusätzlich Testcode enthält
    - Benutzt für endoskopisches Testen

Test-Aktivitäten (erweitert nach J. Wegener)



Unit Testing: Guidelines

- **wiederholt** und **automatisiert** ausführbar
  - häufiges Wiederholen möglich
- der Test läuft **schnell** durch
  - häufiges Wiederholen möglich
- ist **unabhängig** von anderen Tests
  - kann immer wieder unter denselben Bedingungen ausgeführt werden
- **einfach** zu implementieren
  - kompliziertes Setup weißt auf Integrationstests hin

Osherove, Roy (2007). *The Art Of Unit Testing* Manning Publications

Unit Test: Guidelines (con)

- dienen der **Beschreibung** von Funktionalität
  - demonstrieren Benutzung und erwartetes Verhalten
- sollte **nicht öfter** geändert werden als der getestete Code
- auch für andere Entwickler zugänglich

Osherove, Roy (2007). *The Art Of Unit Testing* Manning Publications

Agenda

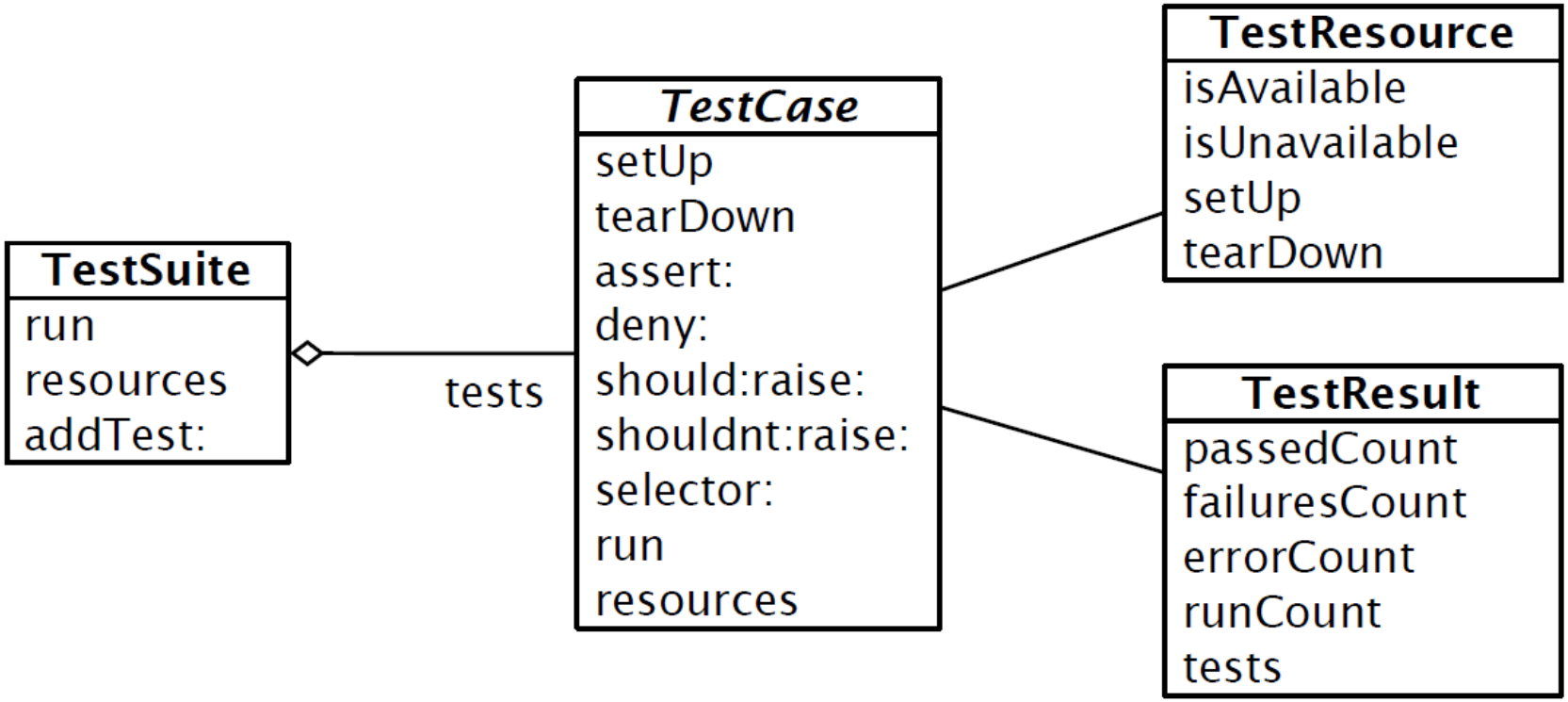
- **Unit Testing**
  - Begriffsklärung
  - Guidelines
- **SUnit**
  - Geschichte
  - Framework
  - Testschritte

SUnit

- **Testing Framework** für Smalltalk

- **Kent Back**:
  - Simple Smalltalk Testing With Patterns (1989)
  - (ursprünglicher) Entwickler von Sunit
  - Schöpfer vom eXtrem Programming

SUnit Framework



Black, A.P., Ducasse, P., Nierstrasz, O., Pollet, D. (2008). *Squeak By Example*

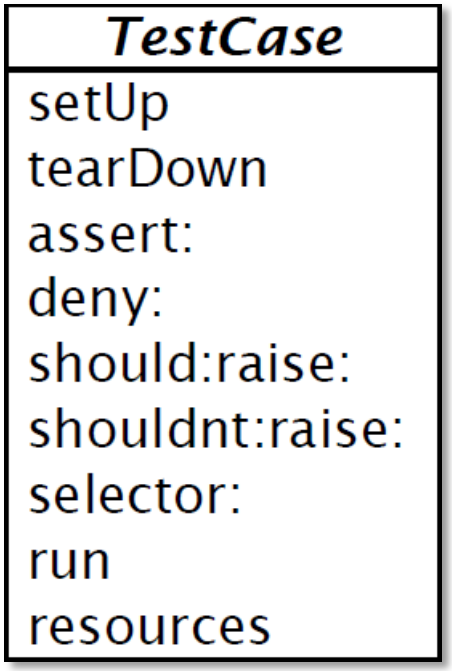
SUnit Framework: TestCase

- **abstrakte Klasse**, wird für eigene Tests abgeleitet

- **Gruppe von Tests** mit gemeinsamen Kontext
  - Tests: Methoden
  - Kontext: Instanzvariablen

- **Testdurchlauf**
  - erzeuge neue `TestCase` Instanz
  - führe `setUp` aus (erzeuge Kontext)
  - führe Testmethode aus
  - führe `tearDown` aus (aufräumen)

- Testdurchlauf wird für **jede Methode** ausgeführt

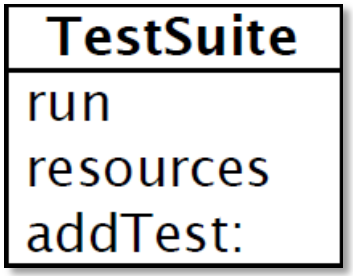


SUnit Framework (cont): TestSuite

» gruppiert TestCases und TestSuites

» ebenfalls ausführbar

» sinnvoll bei zwei und mehr Testfällen, die automatisiert hintereinander ausgeführt werden sollen

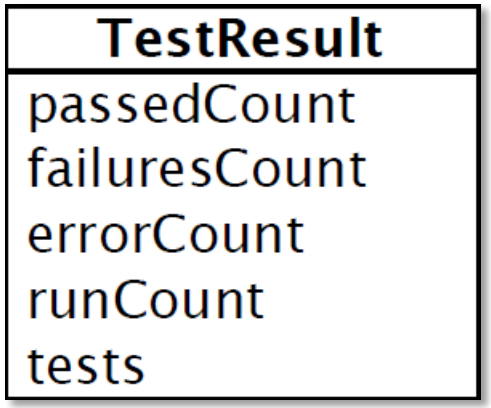


SUnit Framework (cont): TestResult

» Ergebnis einer **TestSuite-Ausführung**

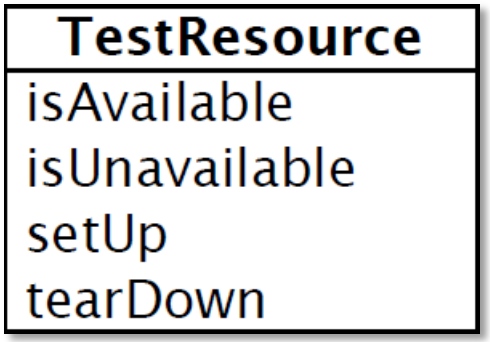
» **beinhaltet**

- » Anzahl bestandener Tests
- » Anzahl nicht bestandener Tests (failures)
- » Anzahl an aufgetretene Fehler (errors)
- » Anzahl ausgeführter Tests



SUnit Framework: TestResource

- **Singleton**
- sinnvoll, wenn eine **Ressource**
  - aufwendig zu initialisieren ist
  - von mehreren Testfällen (einer TestSuite) benutzt wird
  - es genügt die Ressource nur einmal für eine Testsuite zu initialisieren
  - Testfälle, die Ressource nicht zerstören können



- muss mit einer TestCase Instanz **assoziiert** werden
- **Beispiel**: Datenbankverbindung

Agenda

- **Unit Testing**
  - Begriffsklärung
  - Guidelines
- **SUnit**
  - Geschichte
  - Framework
  - Testschritte

Unit Testing: Vorgehen

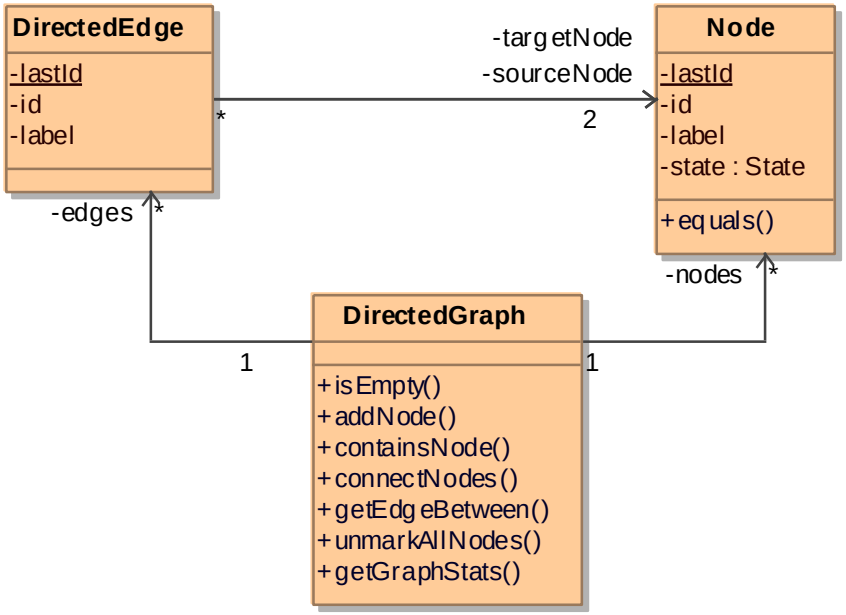
1. **Test-Spezifikation**
   - » Auswahl der Funktionalität und der Testfälle inkl. Testkontext (z.B. via CTE)
   - » Auswahl der Testdaten
   - » Ermittlung des erwarteten Verhaltens

2. **Test-Implementation**
   - » Erzeugung der Testklasse / Testmethoden
     - » Erzeugung des Testkontextes / Testanweisungen
     - » Erzeugung der Test-Orakel (asserts)
   - » Refactoring (z.B. gemeinsame Testkontexterzeugung)

3. **Test-Ausführung**
   - » Testsuite ausführen, evtl. Monitoring (z.B. Coverage)

4. **Test-Auswertung**
   - » Failures, Passes & Errors
   - » erreichte Überdeckungsmaße

Beispiel: Gerichteter Graph



» **Ziel:**

» Modellierung von Aufrufabhängigkeiten

» Zyklensuche, Statistiken ermöglichen

1. Test-Spezifikation

- **Funktionalität**
  - DirectedGraph.connectNodes

- **Testkontext**
  - Ziel- und Quellknoten (Node-Instanzen)
  - Gerichtete Kante (DirectedEdge-Instanz)

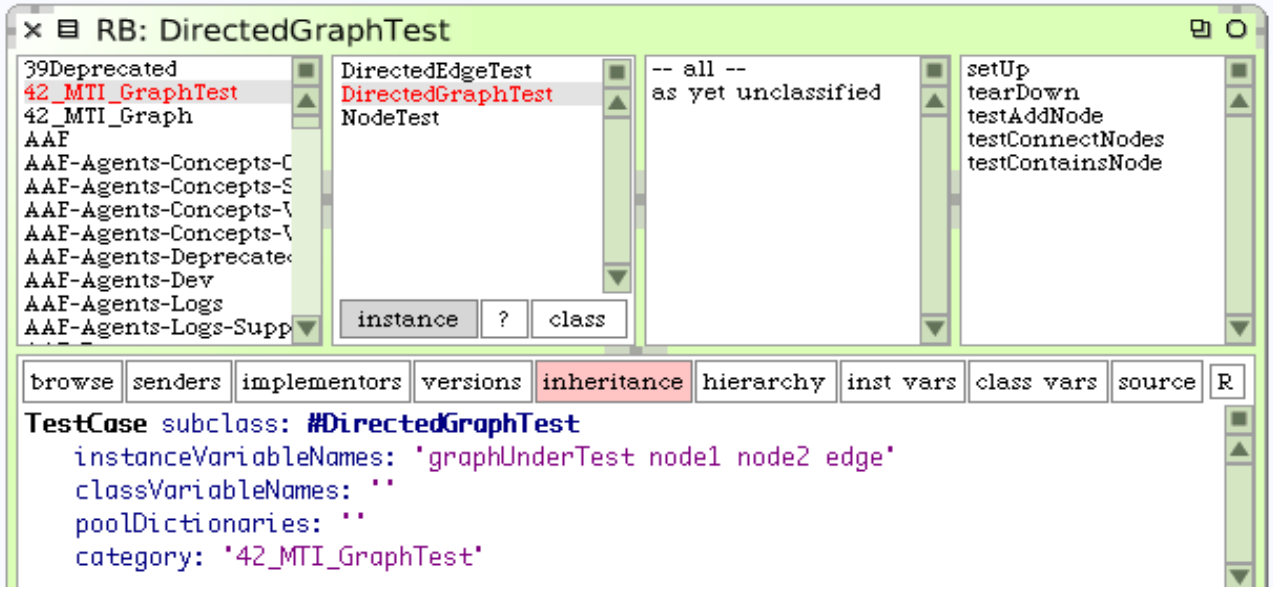
- **Testfälle & Orakel**
  1. TF: Kein Knoten nicht enthalten
     - keine Kante hinzugefügt, *False* geliefert
  2. TF: Quellknoten nicht enthalten
     - keine Kante hinzugefügt, *False* geliefert
  3. TF: Zielknoten nicht enthalten
     - keine Kante hinzugefügt, *False* geliefert
  4. TF: Quell- und Zielknoten im Graph enthalten
     - Kante (Quelle, Ziel) hinzugefügt, *True* geliefert

2. Test-Implementation

TestCase
setUp
tearDown
assert:
deny:
should:raise:
shouldnt:raise:
selector:
run
resources

- **Test-Klasse & Methoden** erstellen
  - als Subclass von TestCase
  - Klasse endet auf **Test**
  - Methoden beginnen mit **test,** besitzen **keine Parameter**

- **Beispiel:** DirectedGraphTest
  - gruppiert Tests für DirectedGraph

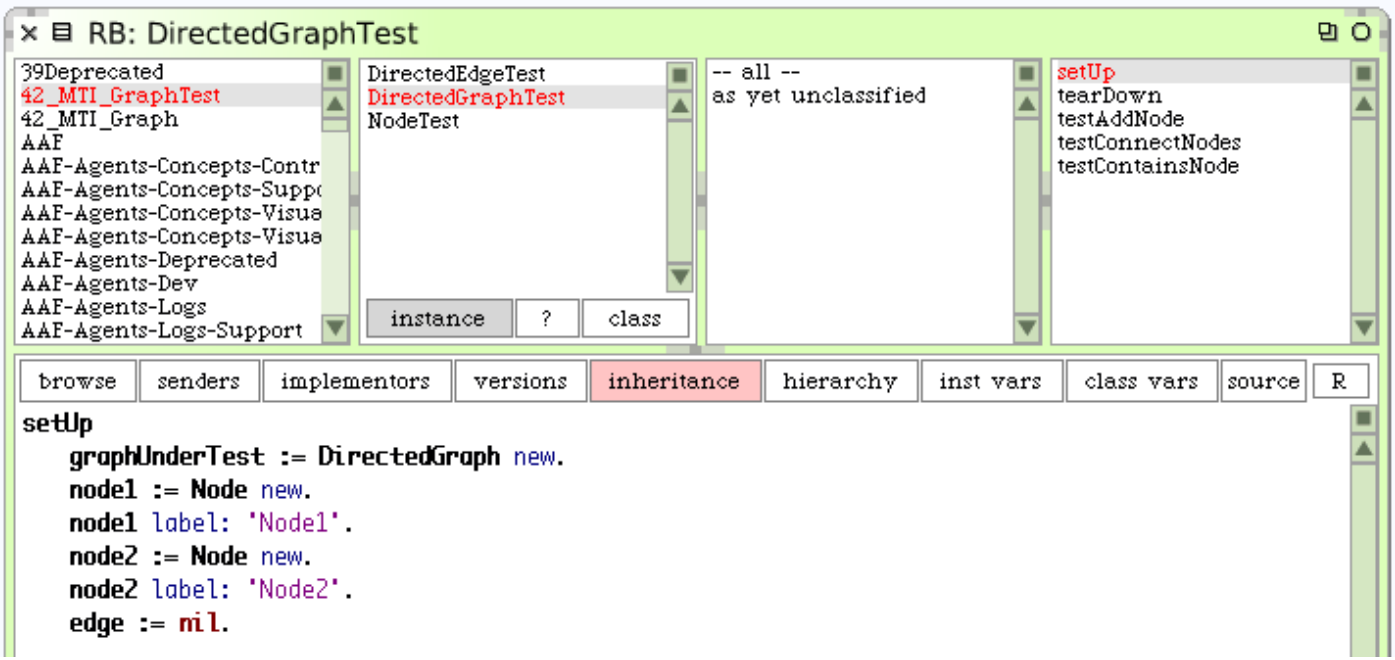


2. Test-Implementation (con)

TestCase
setUp
tearDown
assert:
deny:
should:raise:
shouldnt:raise:
selector:
run
resources

- **Test-Kontext** erstellen
  - **node1, node2, edge**

- **Beispiel:** DirectedGraphTest.setUp
  - erstellt (gemeinsamen) Kontext



2. Test-Implementation (con)

- **Hilfsmethoden**
  - `assert: aBooleanOrBlock`
    - `aBooleanOrBlock` enthält den eigentlichen Test oder auszuwertenden Ausdruck
    - evaluiert zu `true` oder `false`
    - `aBooleanOrBlock` = `true` → Test bestanden
      `aBooleanOrBlock` = `false` → Test **nicht** bestanden
  - `deny: aBooleanOrBlock`
    - Negation von `assert`:

      ```
      assert: aBooleanOrBlock value not
      ```

2. Test-Implementation (con)

- **Hilfsmethoden**
  - `should: aBlock raise: anExceptionalEvent`
    - `aBlock` enthält Testcode und soll die Ausnahme `anExceptionalEvent` auslösen
    - Ausnahme tritt auf → Test bestanden
    - Ausnahme tritt nicht auf → Test nicht bestanden
  - `shouldnt:raise:`
    - Negation von `should`

2. Test-Implementation (con)

TestCase
setUp
tearDown
assert:
deny:
should:raise:
shouldnt:raise:
selector:
run
resources

- **Test-Methode** erstellen
  - testet korrespondierende Methode

- **Beispiel:** DirectedGraphTest.testConnectNodes

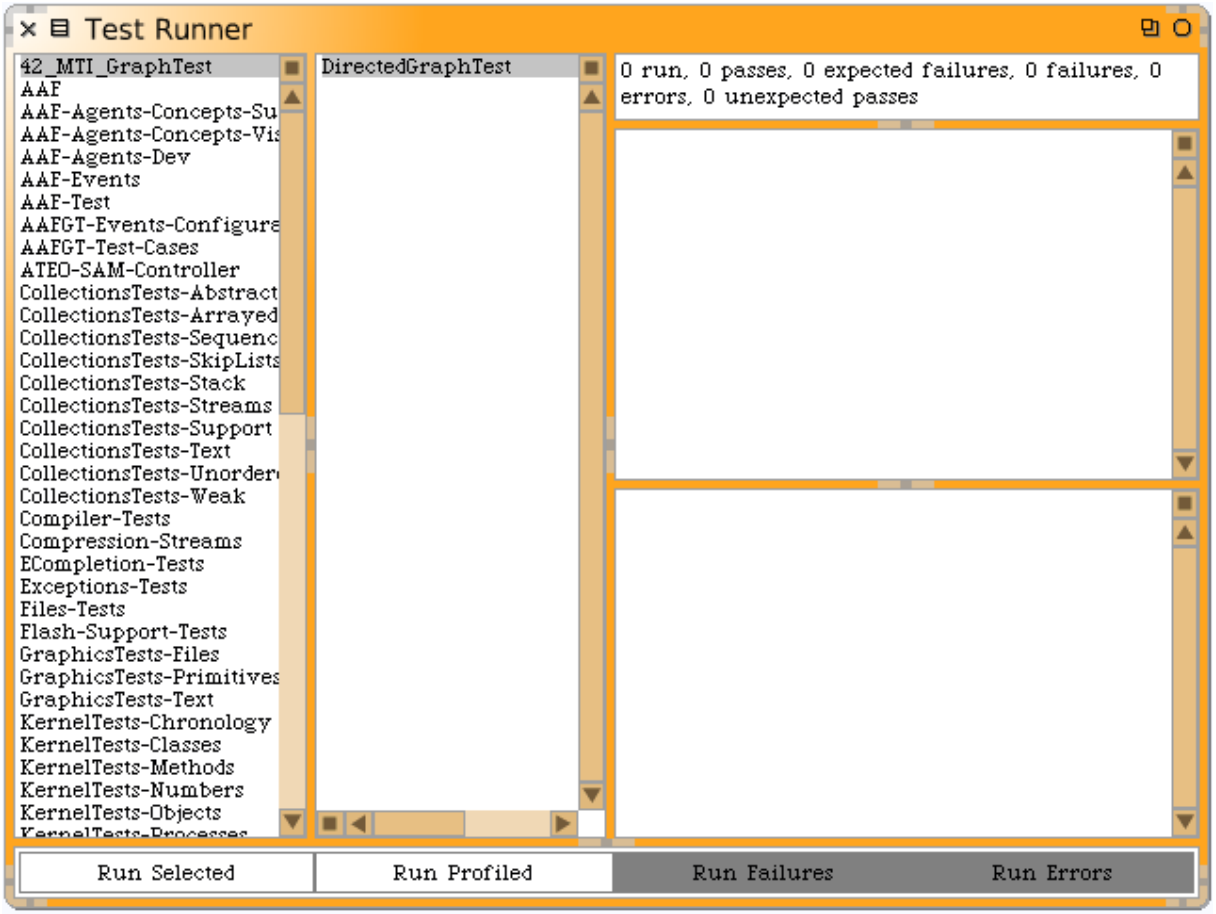
```
testConnectNodes
    "1. empty graph"
    self deny: [(graphUnderTest connectNodes: (node1 label) label: (node2 label)) ].
    self assert: [graphUnderTest edges isEmpty].

    "changing context"
    graphUnderTest addNode: node1.
    graphUnderTest addNode: node2.

    "2. no source node"
    self deny: [graphUnderTest connectNodes: 'notContained' label: (node2 label)].
    self assert: [graphUnderTest edges isEmpty].

    "3. no target node"
    self deny: [graphUnderTest connectNodes: (node1 label) label: 'notContained'].
    self assert: [graphUnderTest edges isEmpty].

    "4. nodes contained"
    self assert: [graphUnderTest connectNodes: (node1 label) label: (node2 label)].
    edge := graphUnderTest edges at: 1.
    self assert: [ (edge sourceNode equals: node1) & (edge targetNode equals: node2) ].
```

3. Test-Ausführung

TestSuite
run
resources
addTest:

TestCase
setUp
tearDown
assert:
deny:
should:raise:
shouldnt:raise:
selector:
run
resources

» **TestRunner** ausführen



4. Test-Auswertung

» **Mögliche Ausgänge eines Tests**

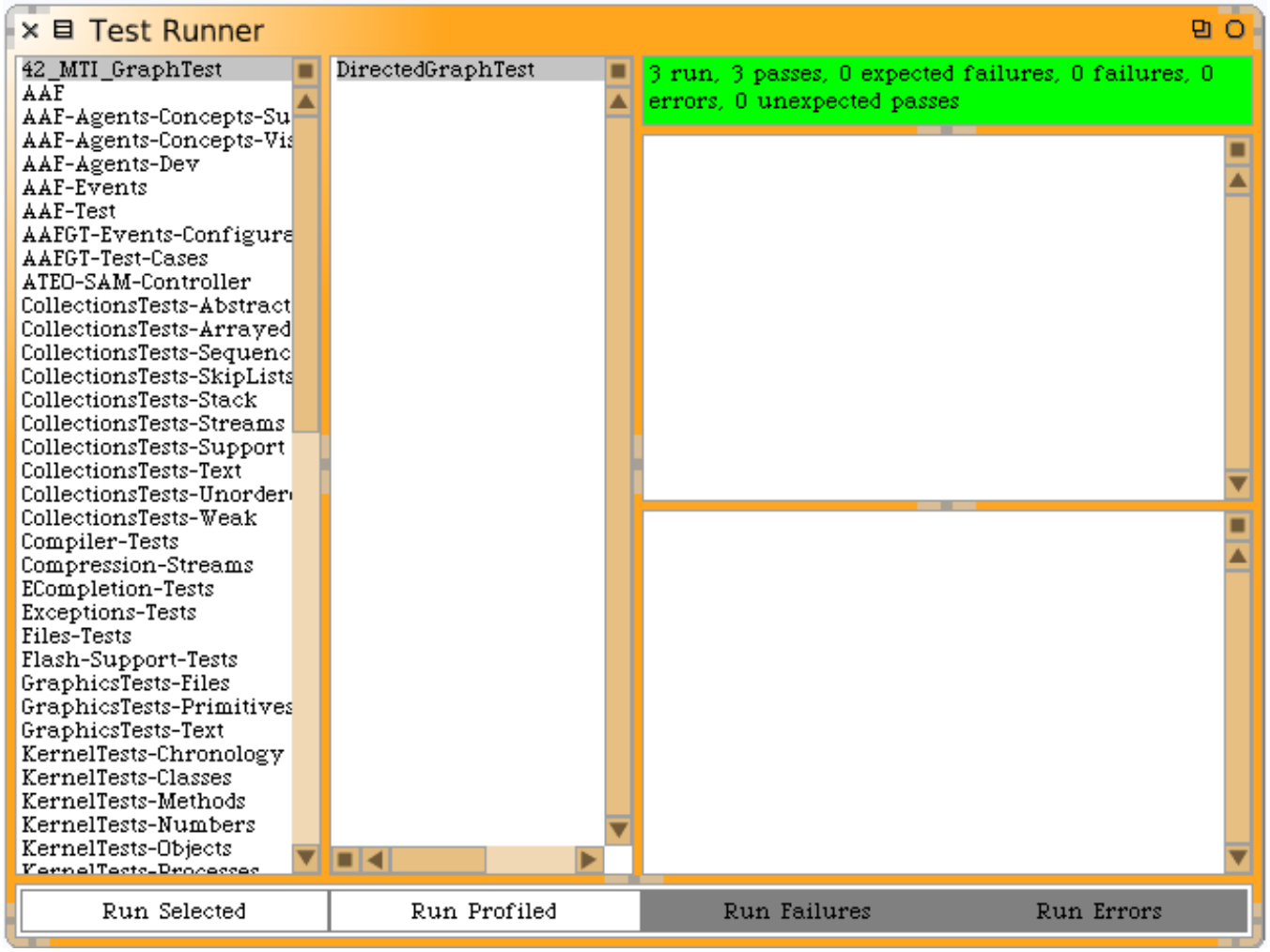
1. **Test bestanden**
   z.B. Block bei assert wertet zu true aus

2. **Test nicht bestanden** (failure)
   z.B. Block bei assert wertet zu false aus

3. **Test abgebrochen** wegen einem Fehler (error)
   z.B. MessageNotUnderstood, IndexOutOfBounds

TestResult
passedCount
failuresCount
errorCount
runCount
tests

4. Test-Auswertung (con)



TestResult
passedCount
failuresCount
errorCount
runCount
tests

DANKE FÜR DIE AUFMERKSAMKEIT.

Fragen?
Hinweise?
Ergänzungen?