51. Testwerkzeuge

Prof. Dr. rer. nat. Uwe Aßmann
Institut für Software- und Multimediatechnik
Lehrstuhl Softwaretechnologie
Fakultät für Informatik
TU Dresden
http://st.inf.tu-dresden.de
Version 11-0.1, 29.12.11

1) Aufgaben und Arten
2) Einzelne Funktionalitäten
    1) Klassifikationsbaum-Methode
    2) Coverage
3) Ausgewählte Testumgebungen
4) Simulation
    1) Debugger

Obligatorische Literatur

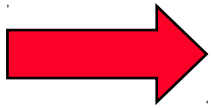
- IMBUS testing http://www.imbus.de
- Englischer Glossar: ISTQB Glossary 1.3
  - http://www.istqb.org/download.htm
  - http://www.imbus.de/engl/download/ct/glossary-current.pdf
- Deutscher Glossar:
- http://www.imbus.de/glossary/glossary.pl?filter=&show_Deutsch=on&pagetype=&Display=
- Zusätzliche Literatur:
  - Peter Liggesmeyer: Software-Qualität - Testen, Analysieren und Verifizieren von Software. Spektrum Akademischer Verlag, Heidelberg/Berlin 2002, S.34. ISBN 3-8274-1118-1
- http://de.wikipedia.org/wiki/Dynamisches_Software-Testverfahren
- http://www.testingstandards.co.uk/Component%20Testing.pdf Britischer Standard mit schönen Definitionen

51.1 Aufgaben und Arten von Testwerkzeugen

Fehlerhafte Software produziert Kosten

- SW-Fehler pro Jahr in Deutschland verursachen Kosten in Höhe von 80 Milliarden EUR (Studie von Lot-Consulting bei 922 deutschen Unternehmen)
- Produktivitätsverlust aufgrund stillstehender Computer kostet 70 Milliarden EUR (Handelsblatt)
- Großteil der Fehler tritt bereits in der

  SW-Entwicklungsphase auf!



Wir brauchen zukünftig noch höhere Qualität - und das in immer kürzerer Zeit!

Quelle: Kugel, Thomas: Qualitätssicherung in der Praxis der Softwareerstellung; Vortrag der GI-Regionalgruppe Dresden am 18.10.2001; URL: http://www.gi-dresden.de/f les/181001.pdf

Aufgaben von Testwerkzeugen

Testwerkzeuge sind Softwaresysteme, die die Analyse von Programmen hinsichtlich ihrer Qualitätsmerkmale (Korrektheit, Performance, Wartbarkeit, Usability, Kosten, ...) oft auf **Basis mehrerer Test-Methoden unterstützen**.

- **Statische Programmanalyse**, ohne dass das Programm ausgeführt wird.
- **Dynamische Programmanalyse** durch Ausführung (oder Simulation) in einer geeigneten Testumgebung mit ausgesuchten Testdaten.

“Black-Box” Test	**“Grey-Box” Test**	**“White-Box” Test**
Funktionsabdeckung Äquivalenzklassenanalyse Grenzwertanalyse intuitive Testfallermittlung Zufallstest Fehlererwartung	“Back-to-Back”-Test Test spezieller Werte zustandsbasierter Test	**Struktur**abdeckung Codeüberdeckung Anweisungsüberdeckg. Zweigüberdeckung Entscheidungsüberd. Weg-Überdeckung

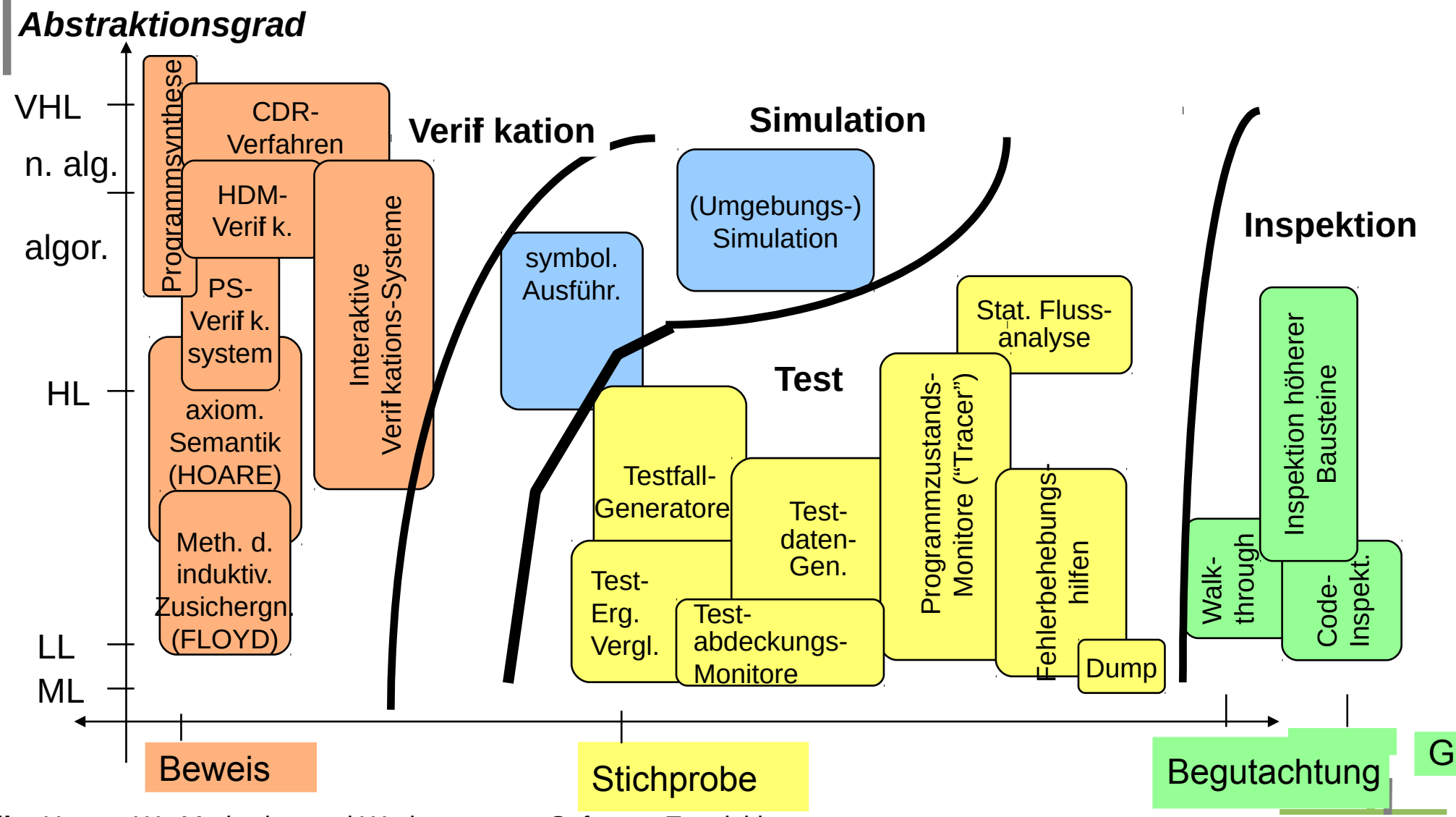
Prüftechniken im Dynamischen Test

- Strukturorientierter Test
  - Kontrollflussorientiert (Maß für die Überdeckung des Kontrollflusses)
    - Anweisungs-, Zweig-, Bedingungs- und Pfadüberdeckungstests
  - Datenflussorientiert (Maß für die Überdeckung des Datenflusses)
    - Defs-/Uses Kriterien, Required k-Tupels-Test, Datenkontext-Überdeckung
- Funktionsorientierter Test (Test gegen eine Spezifikation)
  - Äquivalenzklassenbildung, Zustandsbasierter Test, Ursache-Wirkung-Analyse z. B. mittels Ursache-Wirkungs-Diagramm, Transaktionsflussbasierter Test, Test auf Basis von Entscheidungstabellen
- Diversifizierender Test (Vergleich der Testergebnisse mehrerer Versionen)
  - Regressionstest, Back-To-Back-Test, Mutationen-Test
- Sonstige Mischformen
  - Bereichstest bzw. Domain Testing (Verallgemeinerung der Äquivalenzklassenbildung), Error guessing, Grenzwertanalyse, Zusicherungstechniken

- [Liggesmeyer]

http://de.wikipedia.org/wiki/Softwaretest

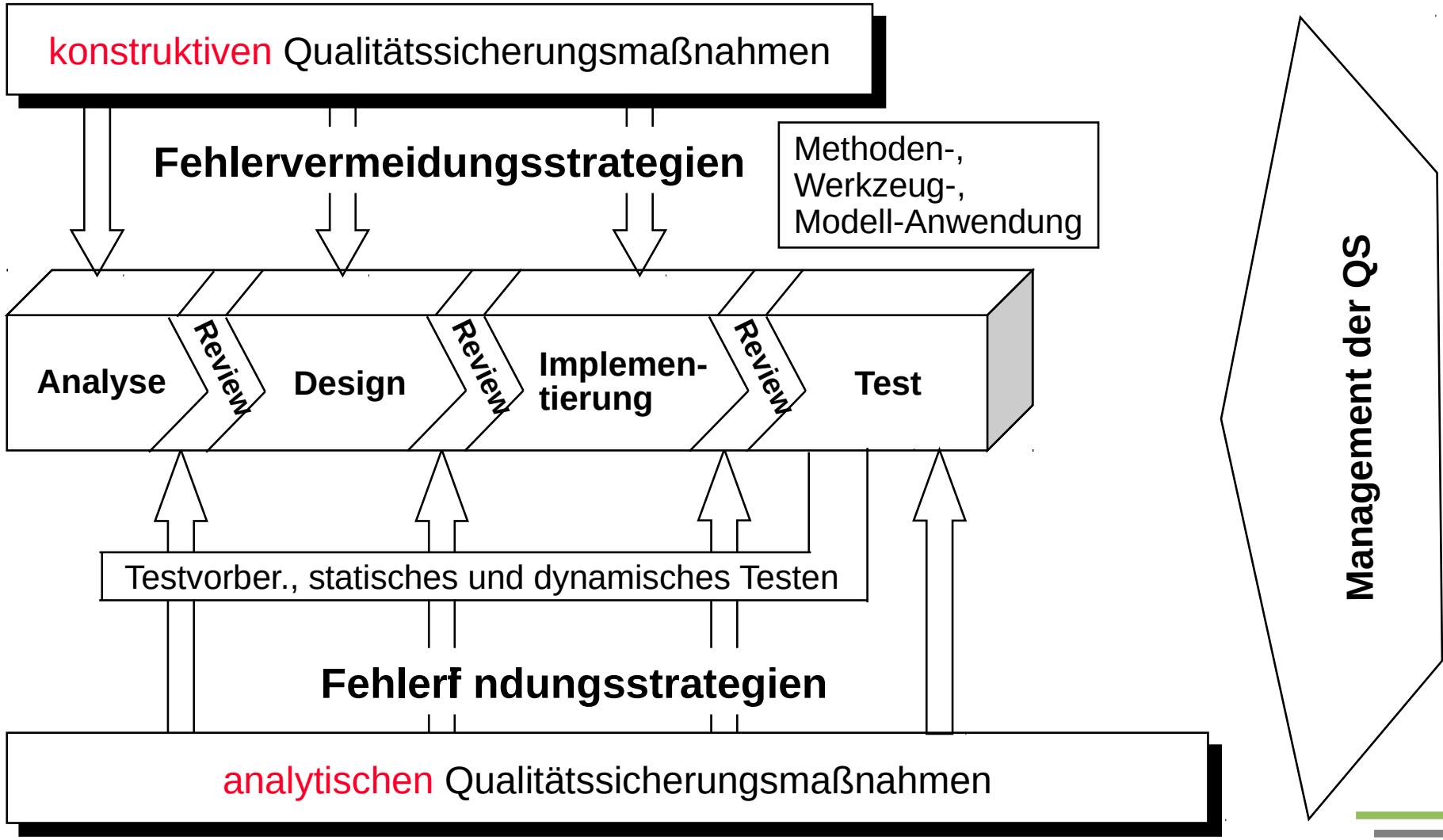
Verifikations- und Validations-Techniken



Quelle: Hesse, W.: Methoden und Werkzeuge zur Software-Entwicklung: Einordnung und Überblick; Informatik-Fachberichte Bd. 43 Springer Verlag 1981

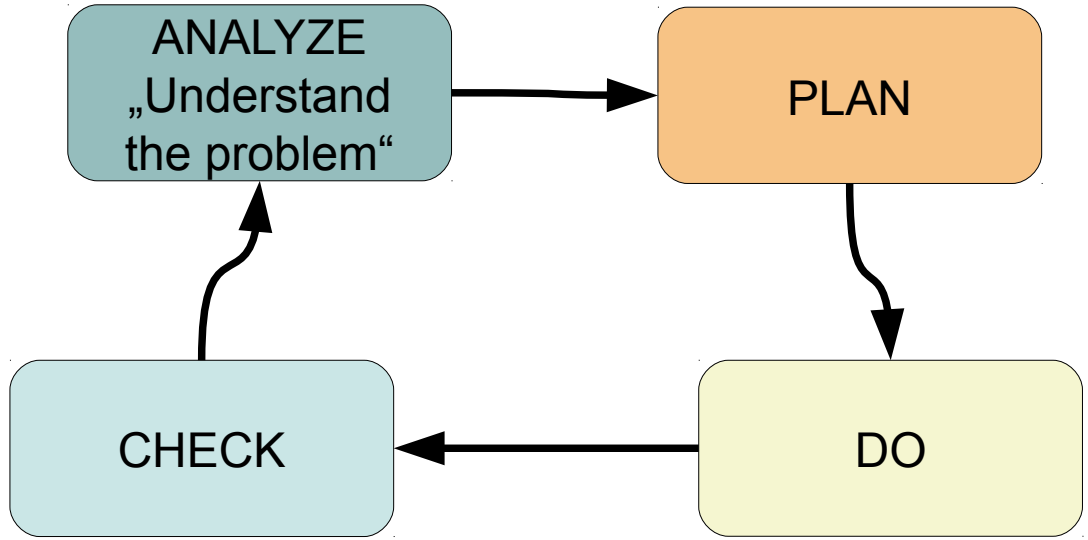
Kategorisierung der QS-Maßnahmen

Maßnahmen der Software- Qualitätssicherung werden differenziert nach:

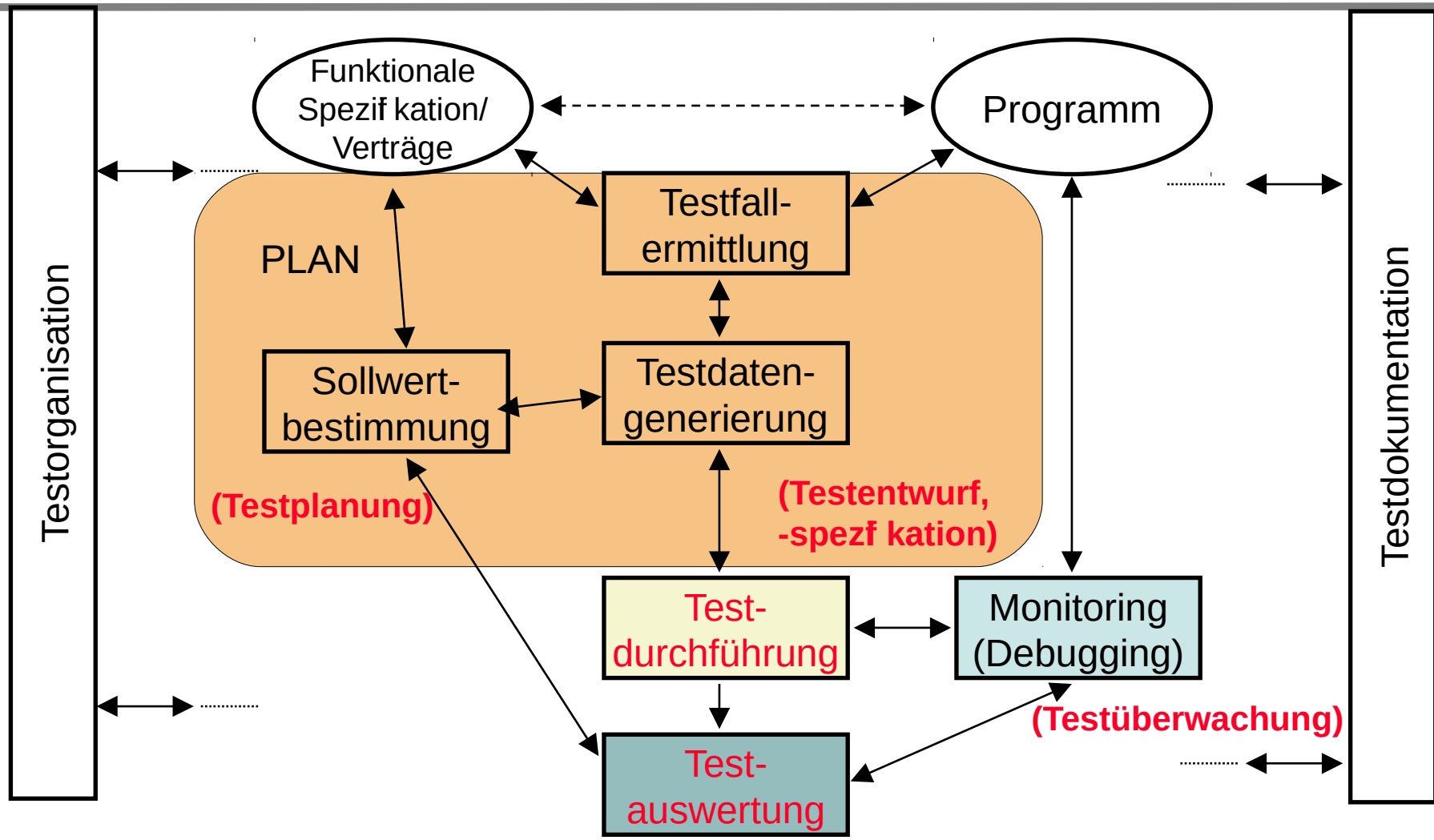


Testing wendet den “Polya Cycle” an

- George Polya. How to Solve It (1945).

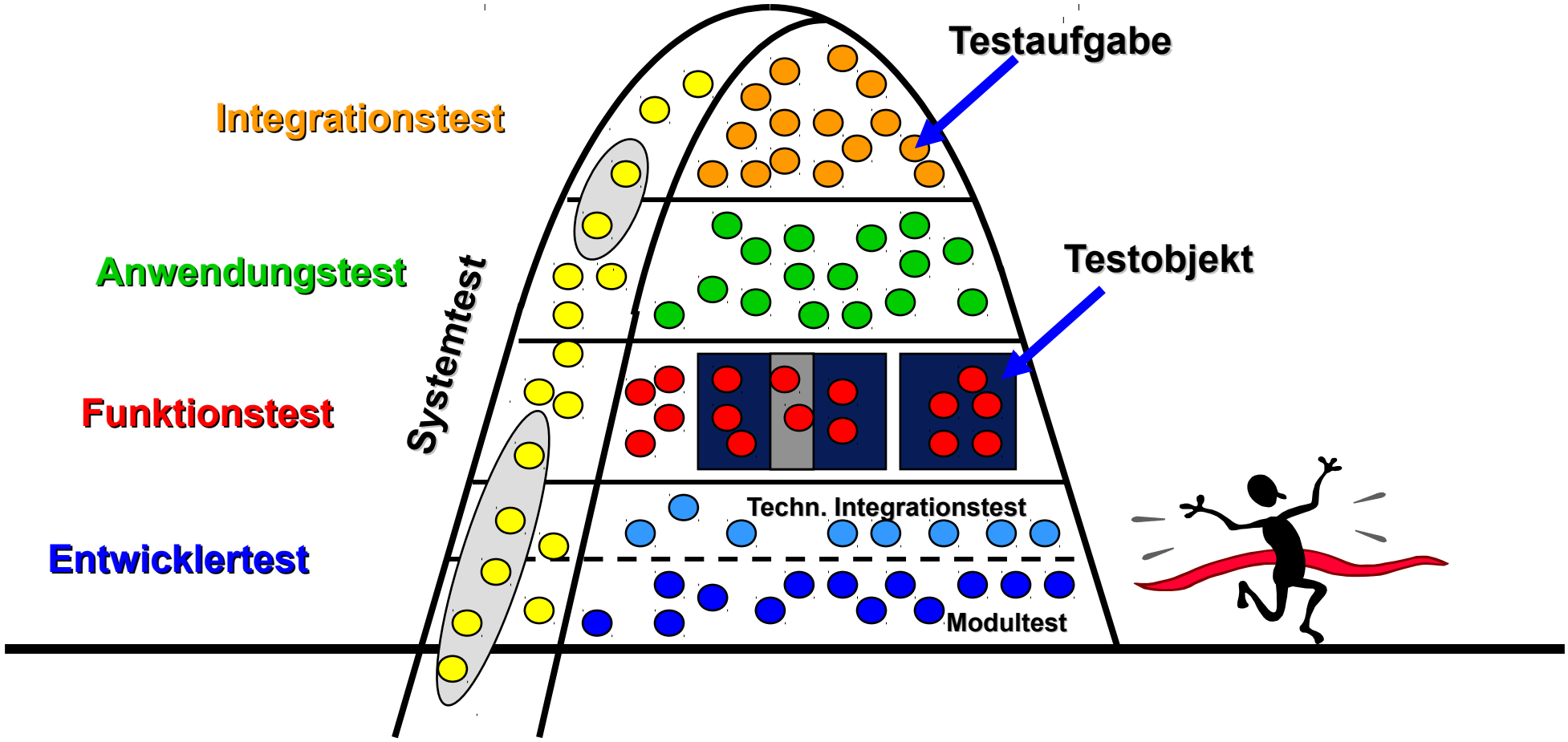


Test-Management



Quelle: Müllerburg, M. u.a.(Hrsg.): Test, Analyse und Verif kation von Software; GMD-Bericht Nr. 260, R. Oldenbourg Verlag 1996 S.115

Testorganisation / Testberg



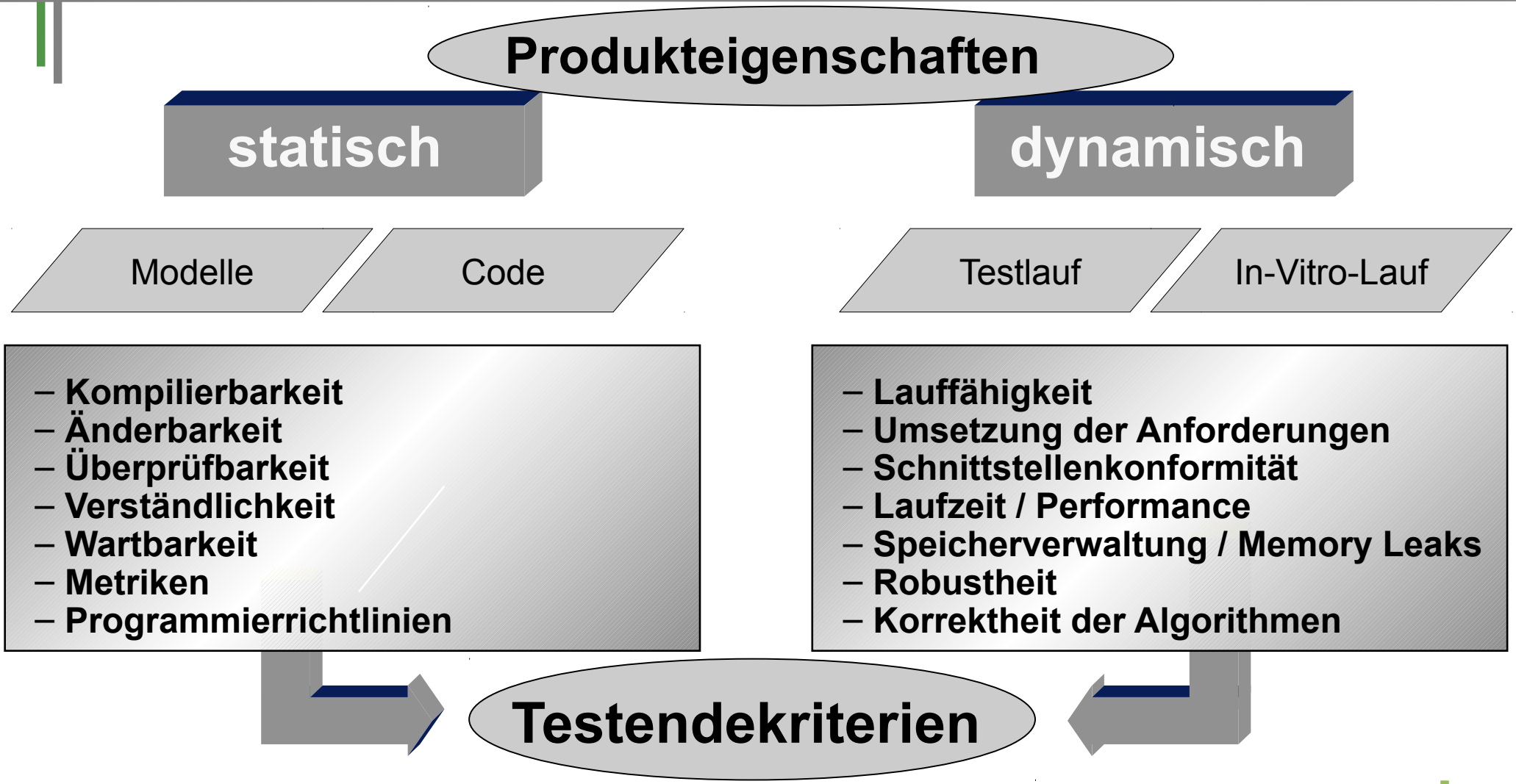
Quelle: Kugel, Thomas: Qualitätssicherung in der Praxis der Softwareerstellung; Vortrag der GI-Regionalgruppe Dresden am 18.10.2001; URL: http://www.gi-dresden.de/f les/181001.pdf

Teststufen im Entwicklungsprozess

- **Entwurfstest:**
  Testobjekt sind alle Dokumente (und Modelle), in denen die Fachlichkeit der Anwendung beschrieben ist. Sie werden durch die Prüf-Werkzeuge der CASE auf Korrektheit, Konsistenz, Kohärenz und Abgeschlossen-heit getestet.
- **Entwicklertest (Einheitentest, unit test):**
  - **Klassentest** ermittelt korrekte Objektzustände, die möglichen Methodenaufrufe und Parameterzustände (vgl. Modultest)
  - **Clustertest** überprüft eine Gruppe von kohärenten, stark voneinander abhängigen Klassen(Package, techn. Integrationstest)
- **Testfallermittlung:**
  Vollständige, systematische Abdeckung des Zustandsraumes der **Test-objekte** über alle möglichen Verkettungen von Methodenaufrufen und Ketten für Sequenzen von Testfällen.
- **Anwendungstest:**
  Betrachtet die Anwendung aus fachlicher Black-Box-Sichtweise. Getestet werden Testfälle aus dem Fachwissen der Anwender heraus.
- **Systemtest:**
  Prüfung des Einf usses verteilter Objekte(Komponenten), die über verschiedene logische oder physische Knoten gemeinsam verwendet werden.

Quelle: Meyerhoff, D.B., Timpe, M., Westheide, J.T.:Teststufen und Testwerkzeuge im objektorientierten Software-Entwicklungsprozess; Softwaretechnik-Trends 18(1998) H. 2, S. 16-19

Entwicklertest: Qualitätsziele



Quelle: Kugel, Thomas: Qualitätssicherung in der Praxis der Softwareerstellung; Vortrag der GI-Regionalgruppe Dresden am 18.10.2001; URL: http://www.gi-dresden.de/f les/181001.pdf

Testen - was?

http://www.imbus.de/testservices/testspektrum.shtml

Dynamischer Test (Test mit Programmausführung):

In-Vitro-Lauf:

- Debugging
- Dynamisches Slicing

Testlauf:

- Funktionaler Test
- Installationstest
- Lizenzierungstests
- Test der Dokumentation (Online Hilfe)
- Migrationstest
- Plattformtest
- Last- und Performanztest
- Stresstest
- Robustheit und Recovery
- Internationalisierungstest (I18N)
- Lokalisierungstest (L10N)
- Security Test
- Usability Test
- Web Test
- Embedded Test
- Interoperabilitätstest
- Koexistenztest

Statischer Test: (Test ohne Programmausführung)

- Statische Analysen
- Statische Vertragsprüfung

Testen – was?

Testmethoden:

- Anforderungsbasiertes Testen
- Geschäftsprozessbasiertes Testen
- Lebenszyklusbasiertes Testen
- Anwendungsfallbasiertes Testen
- Risikobasiertes Testen
- Spezifikationsbasiertes Testen
- Agiles Testen
- Exploratives Testen

Teststufen:

- Komponententest/Unit-Test
- Integrationstest
- Systemtest
- Abnahmetest

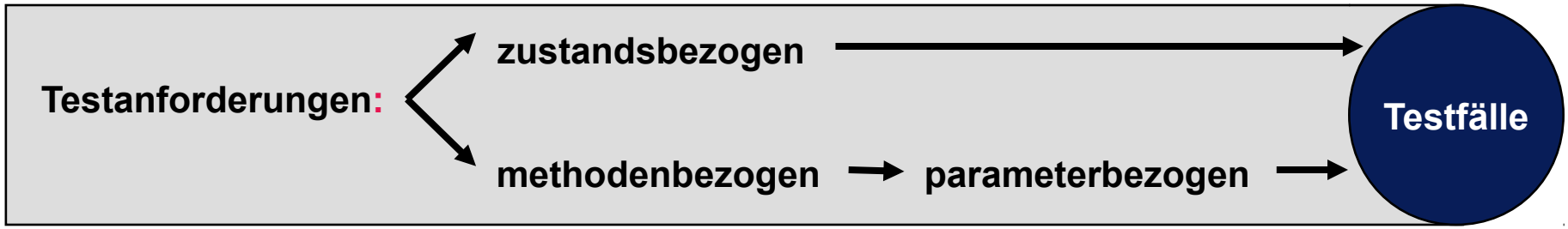
http://www.imbus.de/testservices/testspektrum.shtml

Dynamischer Test: Testfall

- Grundlage für einen Testfall
  - ein (mehrere) Test-*Objekt* in einem gewünschten Ausgangszustand
  - ein (mehrere) *Parameter* für den Aufruf einer *Methode* des Objektes
- Durch den Methodenaufruf ändert sich der Zustand des Objektes
- Prüfung, ob das veränderte Objekt dem erwarteten *Endzustand* entspricht

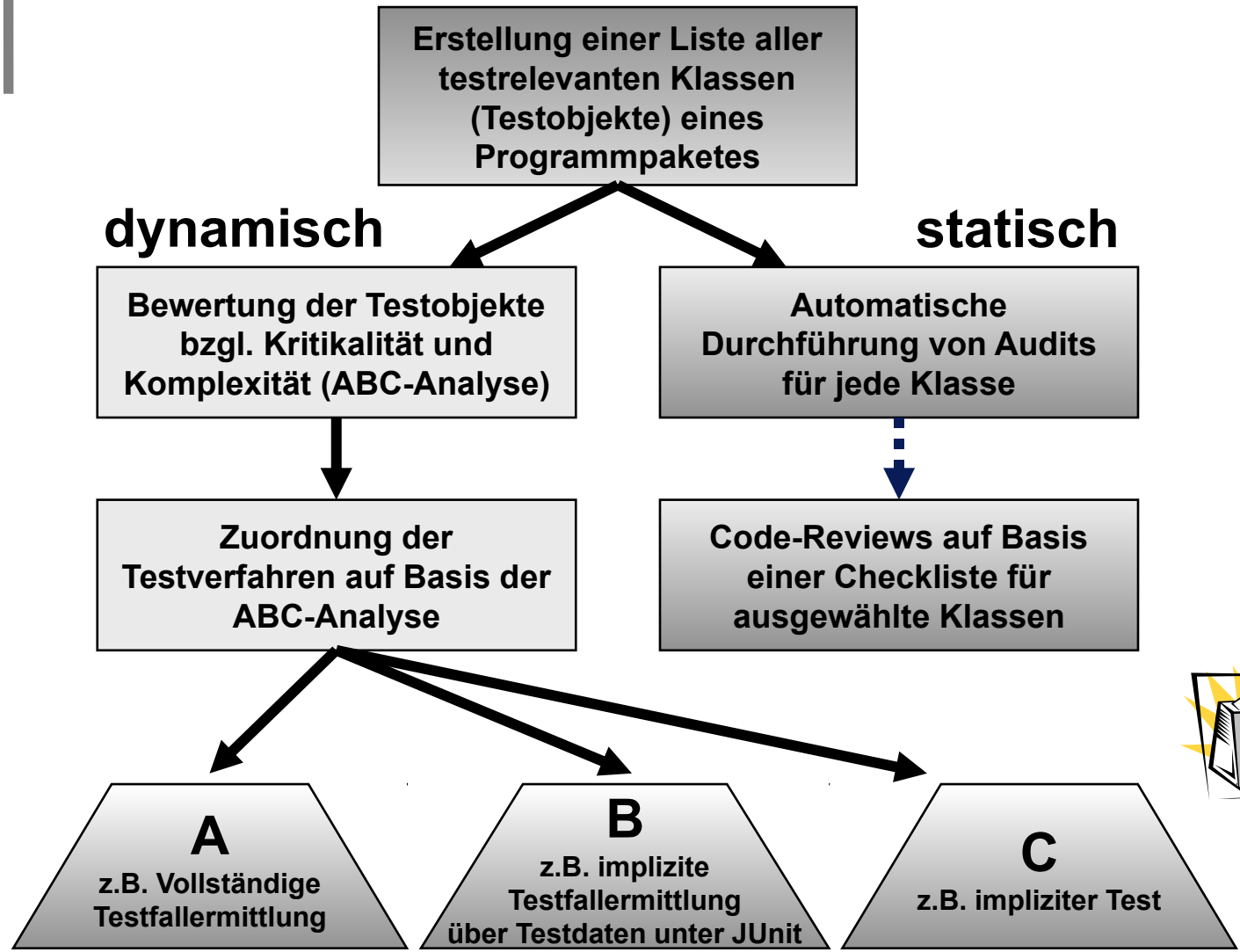
Was wird getestet ? — **Die Klasse (*Objekte, Methoden*)**

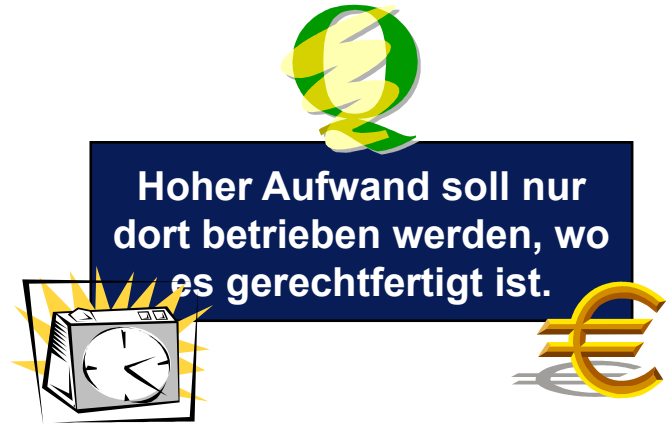
Wo sind die Testdaten ? — **In den Variablen, die *Zustände* beschreiben**
In den *Methodenparametern*



Quelle: Kugel, Thomas: Qualitätssicherung in der Praxis der Softwareerstellung; Vortrag der GI-Regionalgruppe Dresden am 18.10.2001; URL: http://www.gi-dresden.de/f les/181001.pdf

Roadmap für den Entwicklertest





Quelle: Kugel, Thomas: Qualitätssicherung in der Praxis der Softwareerstellung; Vortrag der GI-Regionalgruppe Dresden am 18.10.2001; URL: http://www.gi-dresden.de/files/181001.pdf

Anforderungen an Werkzeuge für den Entwicklertest

- **Modulare Skripte (Testfälle) für maximale Wiederverwendbarkeit**
  - Die Testskripte, die die möglichen Sequenzen von Methodenaufrufen enthalten, sollen modular aufgebaut sein
  - Wiederverwendung über Komponentenkonf gurationen hinweg, zwischen Produkten, sollten möglich sein
- **Schnelle und f exible Sequenzbildung von Skripten** (Testfällen)
  - Ein Testfall sollte aus Input-, dem Call- und dem Output-Block bestehen.
    - Die zu testende Methode wird im Call-Block mit den im Input-Block aufbereiteten Parametern gerufen.
    - Der Output-Block enthält die geforderten Folgezustände, die gegen das Logf le geprüft werden.
- **Einfache Kontrolle der Abdeckung aller Methodenaufrufe und Zustände** (Matrix für alle Zustandsübergänge mit Test-Endekriterium)
- **Automatisierte regressionsfähige Testausführung**
  - unter der Bedingung, das alle möglichen Methodenaufrufe und Zustände in den Testskripten beschrieben sind.
  - Bei Änderung der Testskripte muss gewährleistet sein, daß die ursprünglichen Tests weiterhin als bestanden gelten.
- **Handling und Verwaltung der Skripten für Klassen und Clustertest**

51.2 Werkzeugeinsatz in einzelnen Testaktivitäten

Auswahl-Liste von Test-Frameworks

- Basierend auf dem Framework jUnit

	Unternehmen	URL
JUnit	(K. Beck, E. Gamma) Open Source	www.JUnit.org
JUnitDoclet	Objectfab Dresden	www.objectfab.org
Coverlipse	Sourceforge	coverlipse.sourceforge.net
DDTunit	Sourceforge	http://ddtunit.sourceforge.net/

Weitere Nachweise: http://www.cetus-links.org/oo_testing.html#oo_testing_major_anchor_testing_utilities_tools
http://www.testingfaqs.org/
http://www.imbus.de/tool-list.shtml

Auswahl-Liste von Test-Umgebungen

	Unternehmen	URL
SilkTest	Segue Software	www.segue.com
TestBench	Imbus	www.imbus.de
Visual 2000	McCabe & Associates, USA	www.mccabe.com
Cantata++	IPL, Bath, UK	www.iplbath.com
ClickTracks	ClickTracks Analytics, Inc.CA	www.clicktracks.com

Weitere Nachweise: http://www.cetus-links.org/oo_testing.html#oo_testing_major_anchor_testing_utilities_tools
http://www.testingfaqs.org/
http://www.imbus.de/tool-list.shtml

51.2.1 Die Klassifikationsbaum-Methode

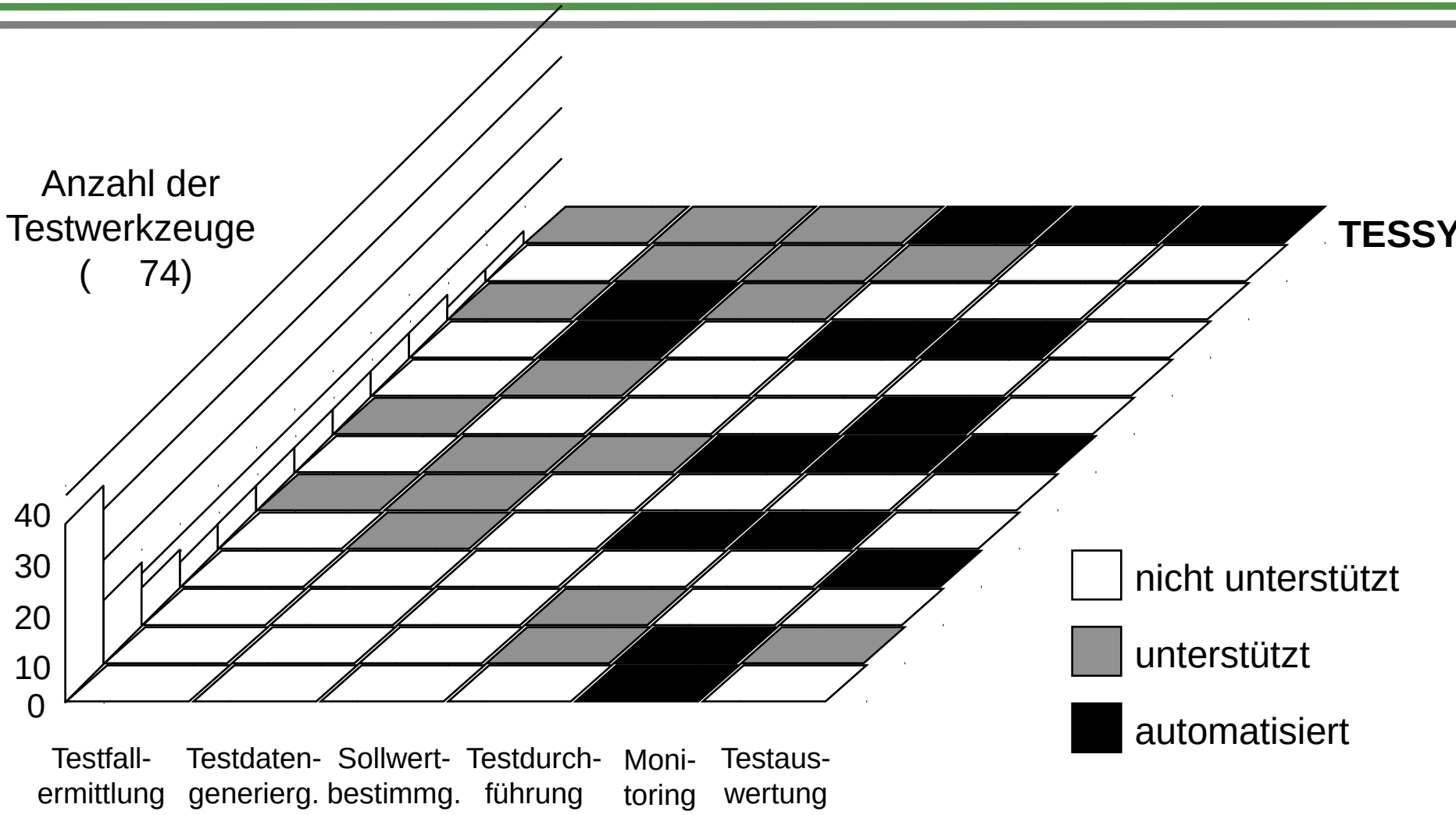
Beispiel-Werkzeugsystem TESSY von HITEX

Aufgaben und Merkmale:

- **Durchgängige** Unterstützung und **kontextsensitive Steuerung** für alle Test-Aktivitäten
- **Unterstützung der Testfallermittlung** für die Bildung zentraler Testobjekte
- Testfallermittlung auf der Grundlage einer **Kombination von funktions- und strukturorientierten Testverfahren** sowie der **Klassif kationsbaum-Methode**
- **Automatisierung von Regressionstests**
- **Prinzipielles Test-Vorgehen:**
  - Ausgangspunkt **funktionsorientierte** Testfälle
  - **strukturorientierte** Testfallermittlung nach der Klassif kationsbaum-Methode
  - Bestimmung der **Programmüberdeckung** nach Auswertung der Durchlaufhäuf gkeiten
  - **Wiederholung** funktionsorientierter(1) oder strukturorientierter(2) Testfälle bis **maximale** Überdeckung der Programmzweige erreicht.

http://www.hitex.com/index.php?id=module-unit-test&L=2

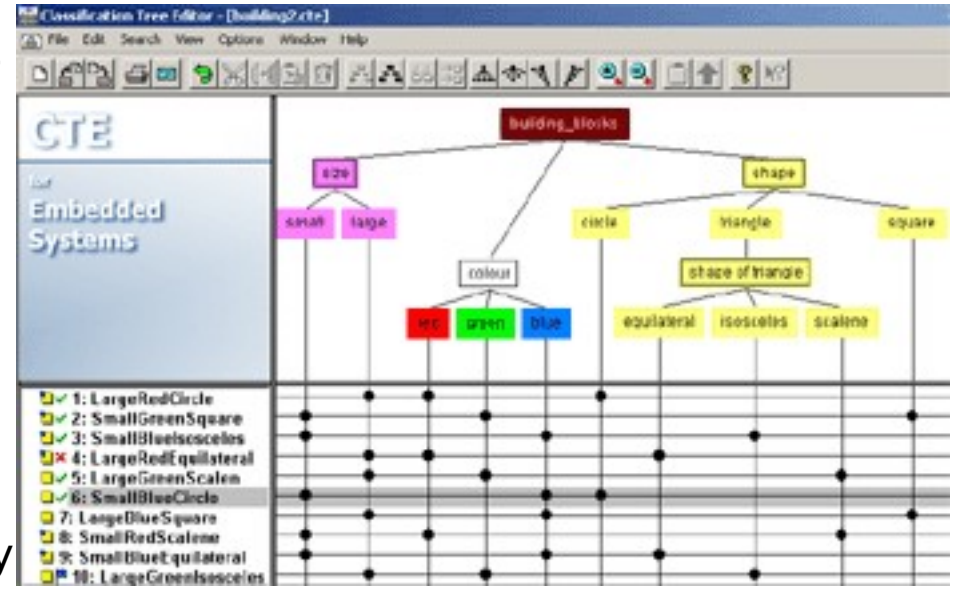
Automatisierungsgrad von Werkzeugen für den Unit-Test (Beispiel TESSY)



Quelle: Wegener, J., Pitschinetz, R.: Testsystem TESSY zur Unterstützung von Software-Tests; in Müllerburg, M. u.a.(Hrsg.): Test, Analyse und Verif kation von Software; GMD-Bericht Nr. 260, R. Oldenbourg Verlag 1996

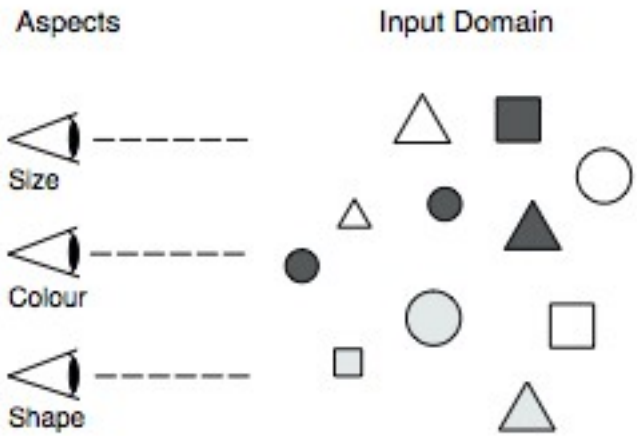
Klassifikationsbaum (Classification Tree)

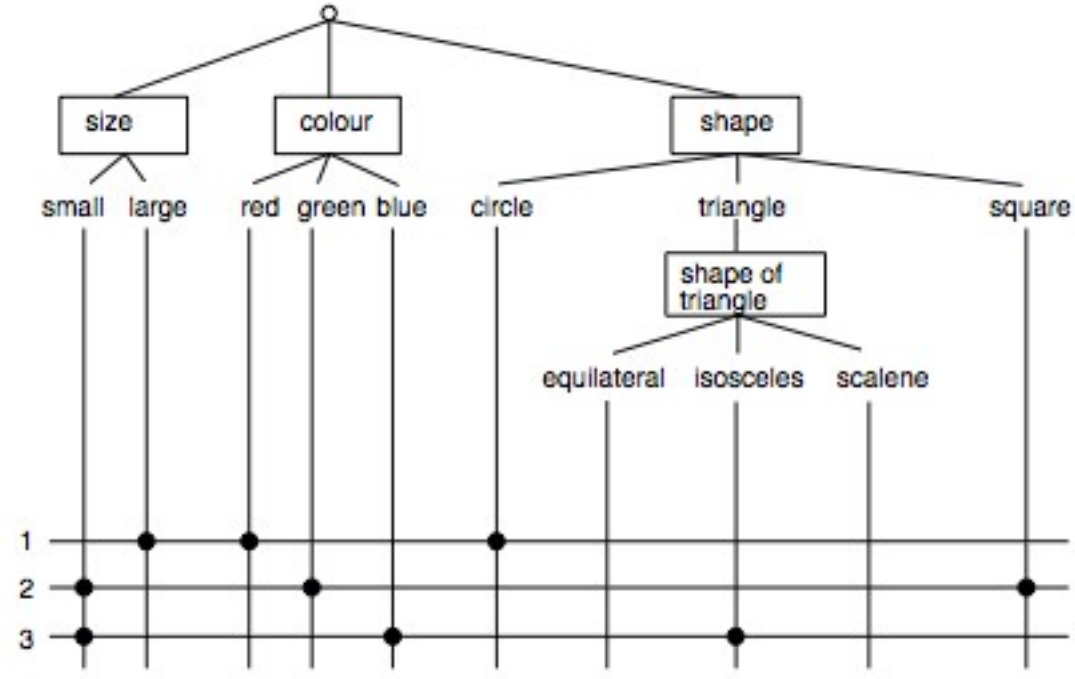
- Einteilung der Testdaten in Kategorien
  - http://de.wikipedia.org/wiki/Klassifikationsbaummethode
  - Grochtmann, M., Grimm, K.: Classification Trees For Partition testing, Software testing, Verification & Reliability, Vol. 3 (2), June 1993, Wiley, pp. 63 – 82.
  - Grimm, Klaus: Systematisches Testen von Software: Eine neue Methode und eine effektive Teststrategie. Oldenburg, 1995. GMD-Berichte Nr. 251.
  - Grochtmann, M. Test Case Design Using Classification Trees. STAR'94, 8 - 12 May 1994, Washington. http://www.systematic-testing.de/documents/star1994.pdf



Kategorien (Facetten, Aspekte) der Testfalldaten

- Testfälle werden in einer Matrix der einzelnen Kategorien und ihrer Werte ermittelt
  -





Testfall Nr

Vorteile der Klassifikationsbaum-Methode

- Aspektorientierung reduziert die Komplexität
  - mehrere Klassifikationen ermöglichen es, das Problem in Dimensionen aufzuteilen
  - Visualisierung, auch gerade für Manager und Gutachter
- Abdeckungsgrad
  - Wohlüberlegte Testfallkonstruktion deckt die meisten Fehlerfälle ab
- Test-Ende-Kriterium
  - falls alle Testfälle der Kreuztabelle erfüllt
- Automatisierung
  - der CTE kann bereits elementare Testfälle generieren

Werkzeugpalette von TESSY

Editor CTE	Vervollständigung und Überwachung des **Klassif kationsbau-mes** ➔ systematische Def. von funktionsorient. Testfällen ➔ Erstellen Klassif.baum für aktuelles Testobjekt ➔ Generierung von Testfällen
Environment-Editor	Organisation **testvorbereitender** Festlegungen zur Testumge-bung des Testobjekts (Unit-Test)
TESSY-System	Ermittlung der Exportschnittstelle durch Parsen der Quellen ➔ Funktionen (mit globalen Variabl., Parametern, Rückgabewerten und Datentypen) bilden eigentliche **Testobjekte**
Testdaten-Editor TDE und **Browser**	**Eingabe** konkreter **Testdaten** und **Sollwerte** zu jedem def nier-ten Testfall des Testobjektes **Browserfenster** dienen getrennter Eingabe der Daten und über-sichtlicher parametergesteuerter Auswahl der Testbedingungen
Monitoring EXP (execution panel)	Nach Auswahl des Testfalls generieren des Testtreibers ➔ **Test-durchführun**g mit Messung der Zweigüberdeckung ➔ Registrie-rung der Ergebnisse in Echtzeit ➔ Protokollierung ➔ Herstellung Ausgangszustand
Testauswertung EVP (evaluation panel)	Generieren der **Testdokumentation** unterschiedlicher Granu-larität ➔ Aufbereitung zur Weiterverarbeitung in speziellen Dokumentationswerkzeugen

51.2.2 Coverage Tools – Werkzeuge zur Pfadabdeckung

- http://de.wikipedia.org/wiki/Kontrollflussorientierte_Testverfahren
- http://de.wikipedia.org/wiki/Dynamisches_Software-Testverfahren

Steuerflussorientierter Test (code coverage)

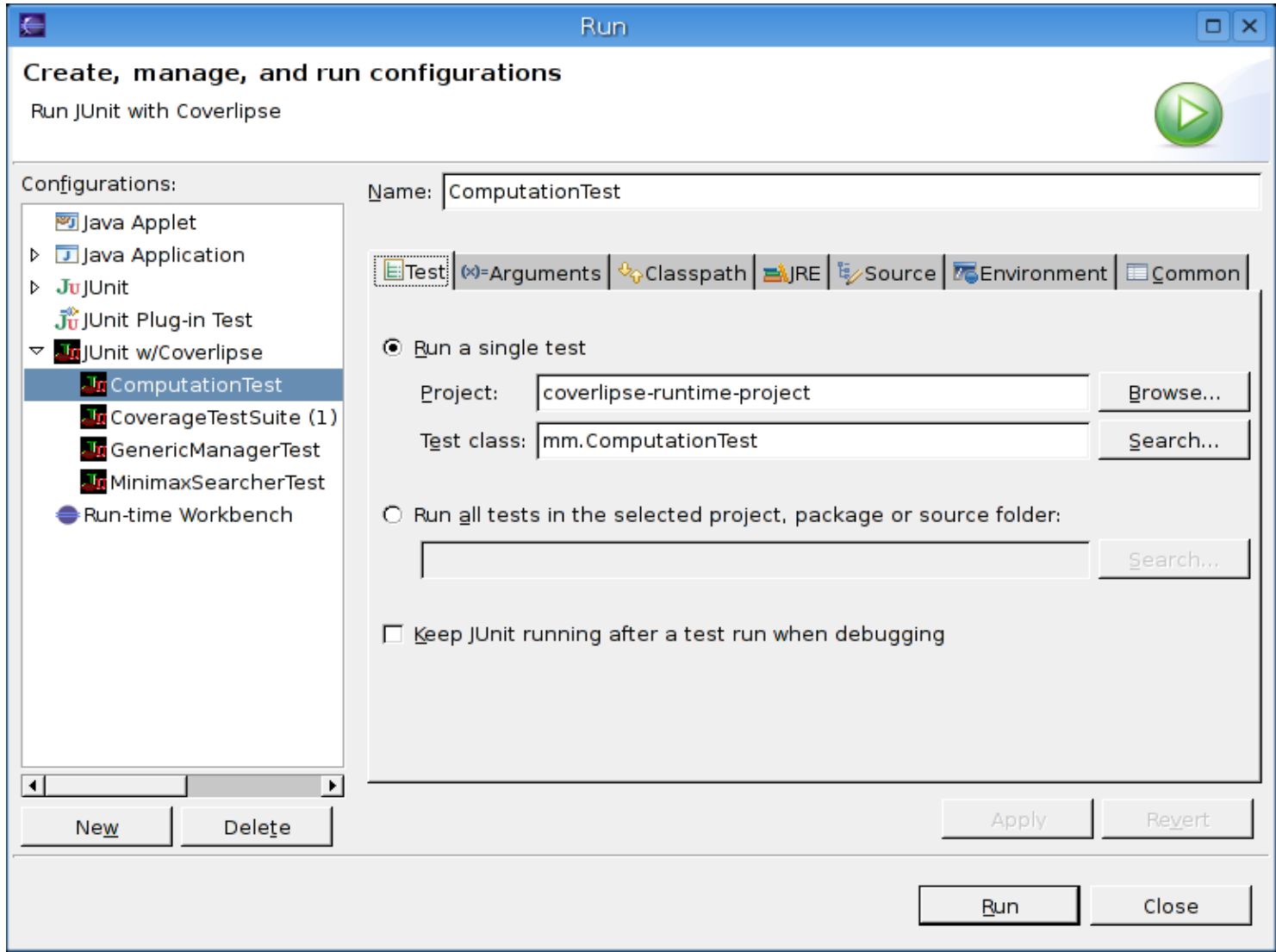
Überdeckungs-einheit	Arbeitsweise	Zweck
Anweisung	Möglichst viele Anweisungen werden mit Testfällen überdeckt	Entdeckung toten Codes
Bedingung (Alternative)	Jede alternative Belegung einer Bedingung wird durch einen Testfall getestet	Alle Kanten des Steuerfluss-Graphen werden überdeckt
Bedingungs-kombination	Möglichst viele Kombinationen mehrerer Bedingungen werden getestet. Abdeckung azyklischer Pfade durch das Programm	Problem: kombinatorische Explosion
eingeschränkte Bedingungs-kombination	Alle Kombinationen derjenigen Teil- Bedingungen, die unabhängig voneinander die Gesamtbedingungsergebnis beeinflussen (Unabhängigkeit der Teilbedingungen)	Reduktion des Aufwandes
Pfad	Abdeckung auch zyklischer Pfade	Im Allgemeinen unmöglich; Einschränkung auf Durchlaufsschranke k
Boundary-Test	Abdeckung aller Pfade bei höchtens einmaligem Durchlauf durch eine Schleife	Begrenzung auf k<=1
Interior-Test	Abdeckung aller Pfade bei höchtens zwemaligem Durchlauf durch eine Schleife	k<=2

Datenflussorientierter Test (data flow coverage)

Überdeckungs-einheit	Arbeitsweise	Zweck
All defs	Für alle Definitionen von Variablen gilt: ein Pfad zu einer Benutzung muss getestet werden	Entdeckung toter Variablen (Definitionen)
All p-uses	Für eine Definition einer Variablen werden alle Benutzungen *in Prädikaten* getestet	Einfluss der Variable auf den Steuerfluss
All c-uses	Für eine Definition einer Variablen werden alle Benutzungen *außerhalb von Prädikaten* getestet (in rechten Seiten oder in Zeigern auf linken Seiten)	Einfluss der Variable auf den Datenfluss

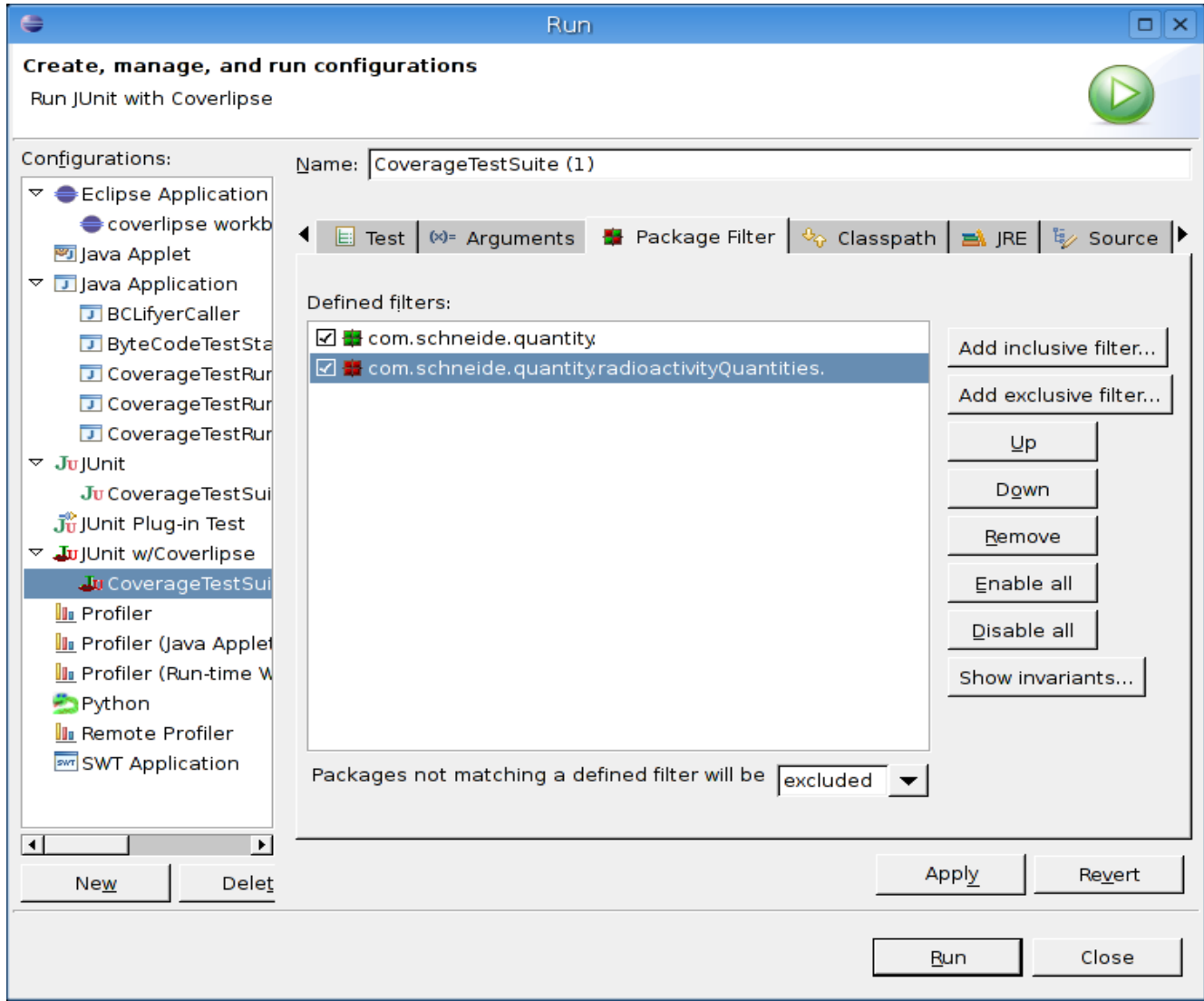
Bsp.: Coverlipse basiert auf JUnit

- Selektion von Junit-Testfällen und deren Pfadabdeckungsanalyse



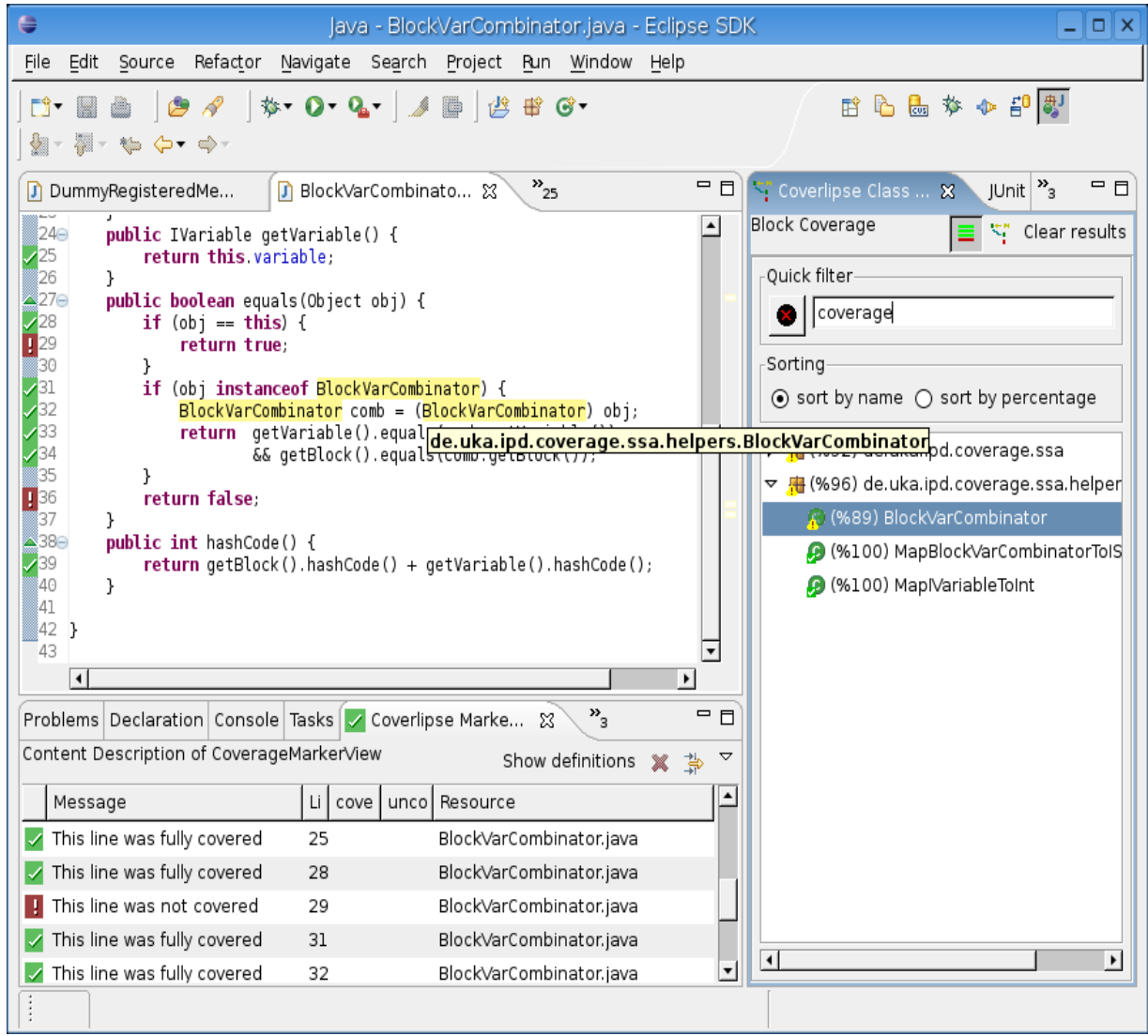
Coverage Tools – Werkzeuge zur Pfadabdeckung

- Paketfilterung stellt die Pakete zur Pfadabdeckungsanalyse ein



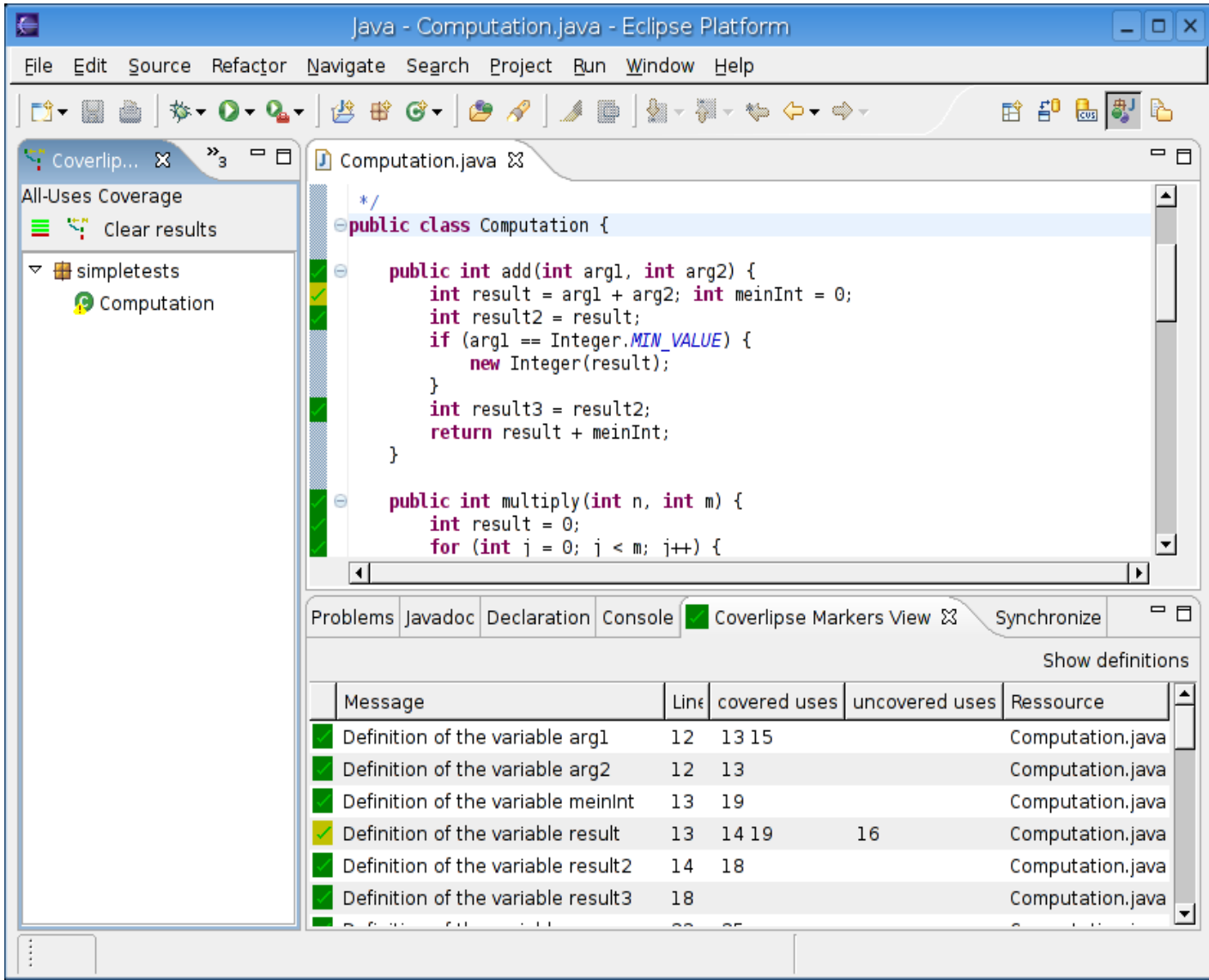
Coverlipse

- block coverage / statement coverage



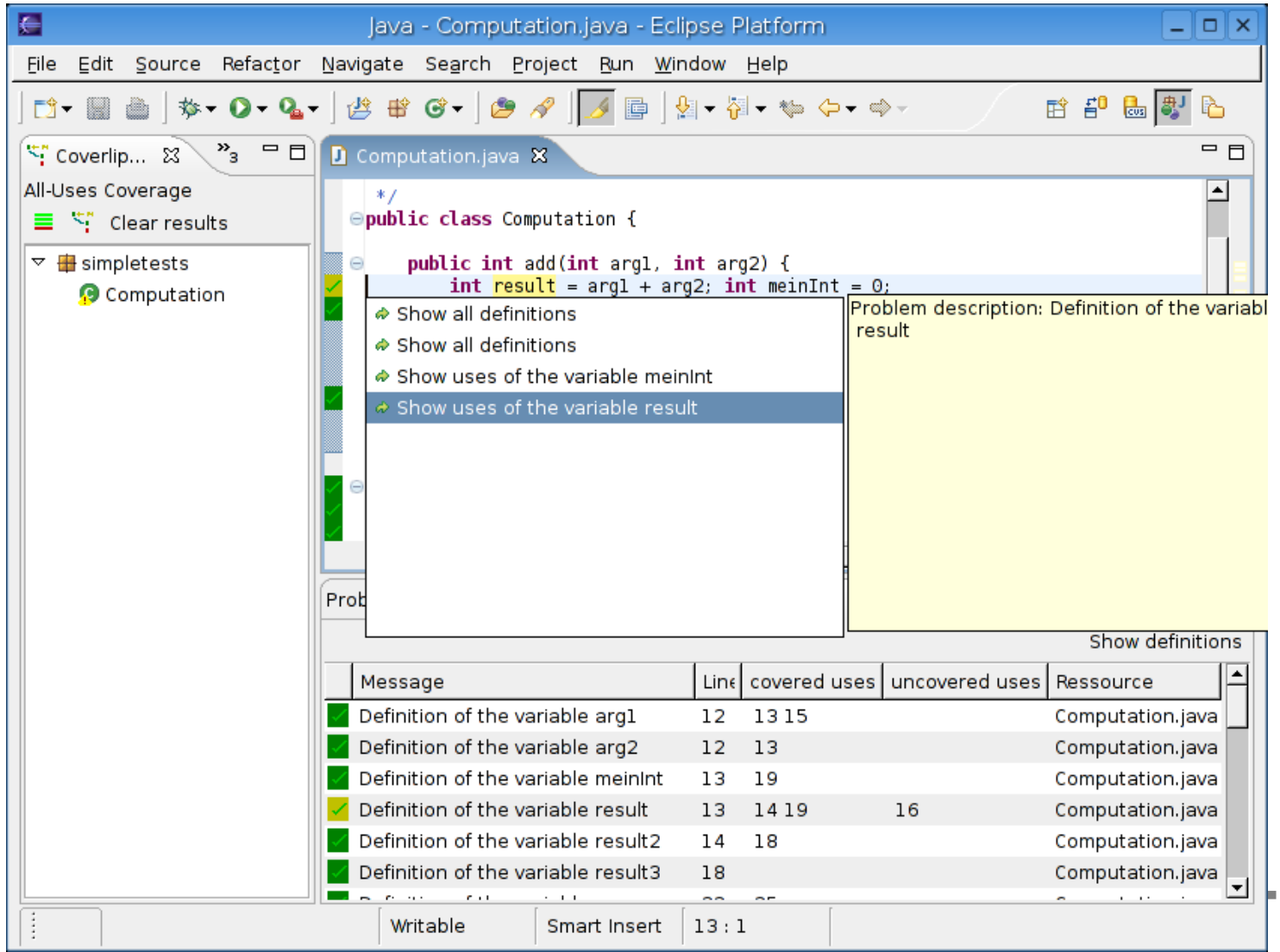
Coverlipse

- All-uses-coverage



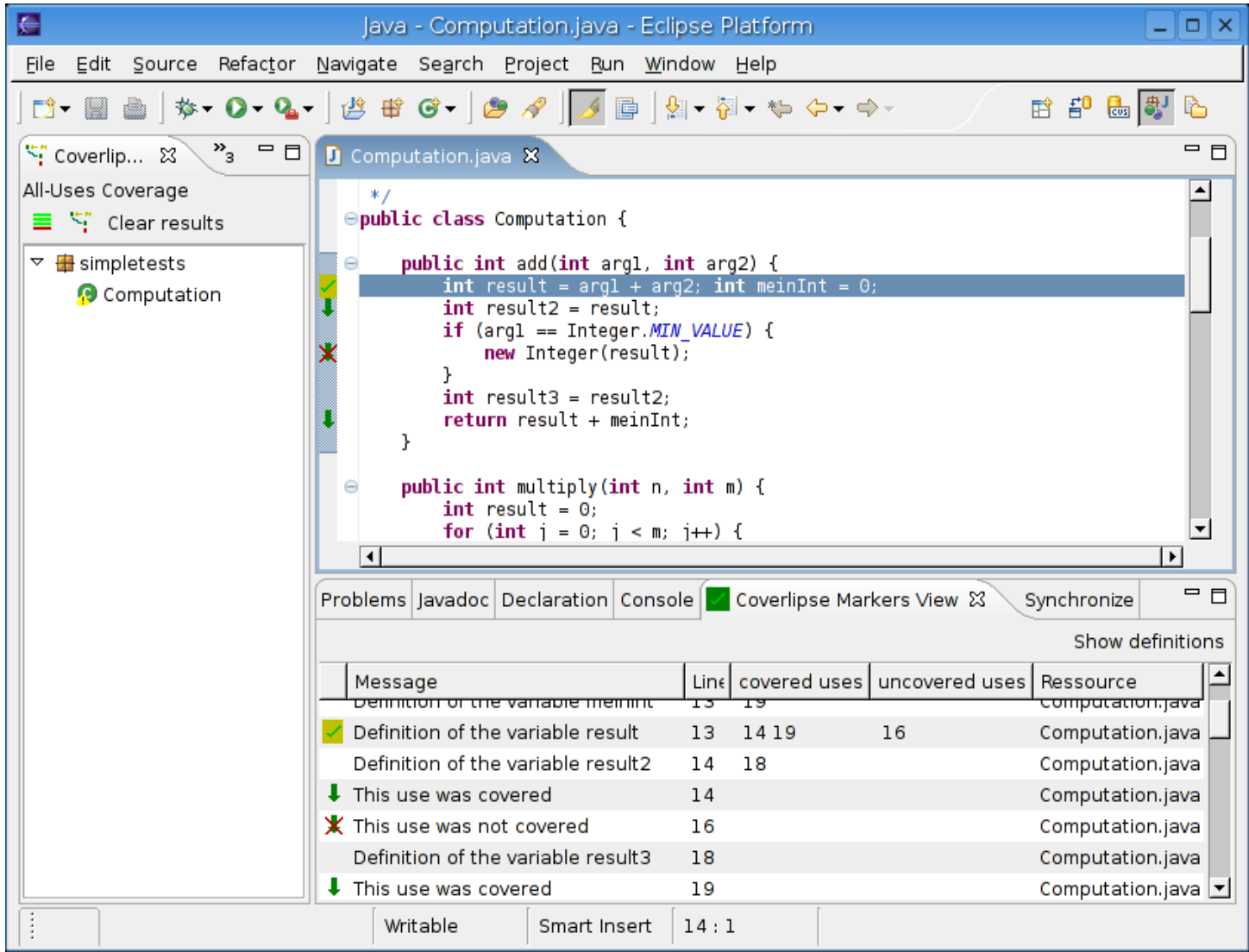
Coverlipse

- Problembeschreibung aus einem Use



Coverlipse

- all-uses-coverage information



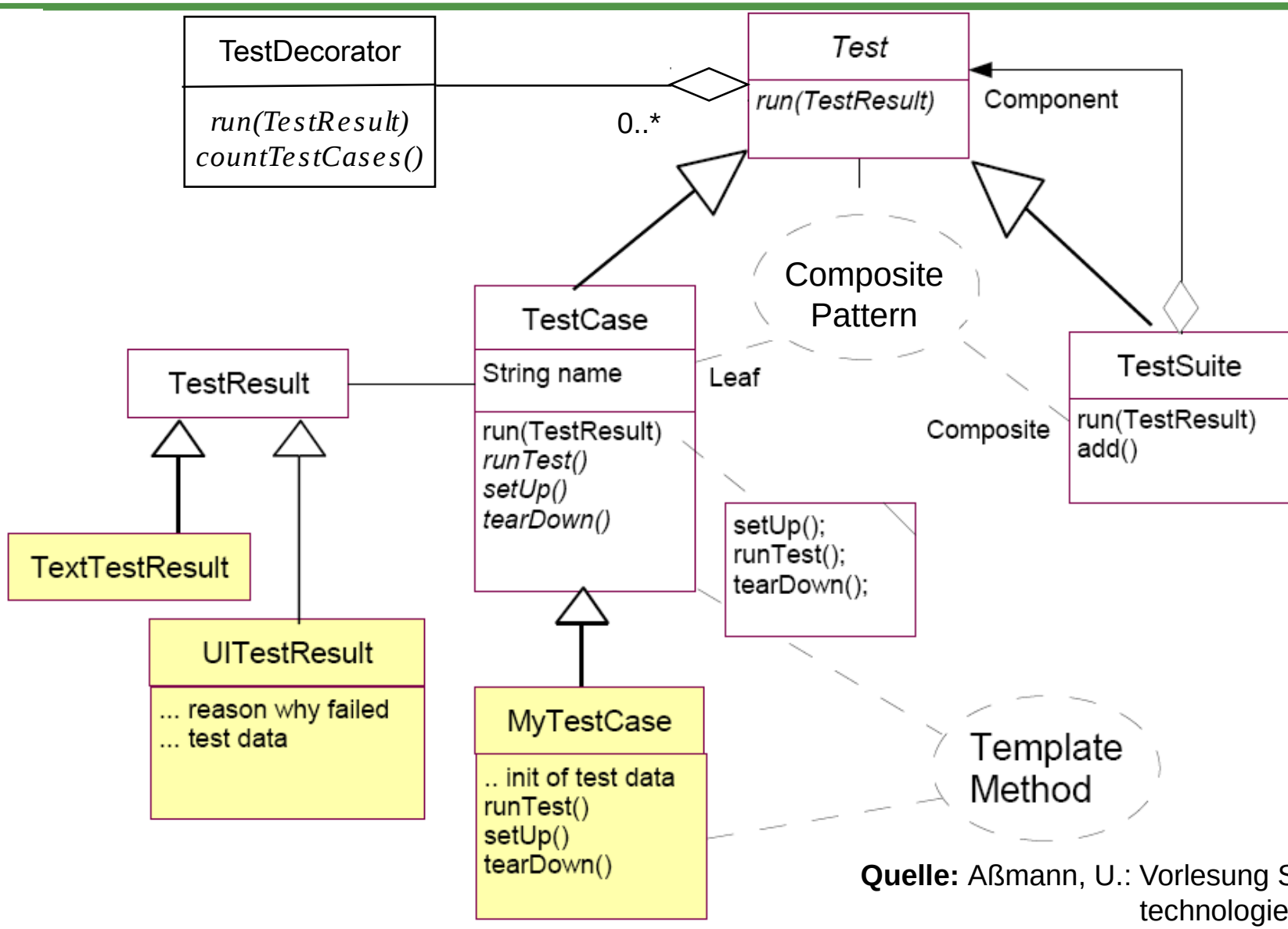
51.4 Funktionalität und Werkzeuge ausgewählter Test-Umgebungen

Framework JUnit für Komponententest

- **Entwickler:**
  - Software ist frei und im Kern von Kent Beck und Erich Gamma geschrieben
  - erhältlich von Free Software Foundation (http://www.gnu.org) oder als IBM Common Public License von http://sourceforge.net/projects/junit/
- **Anwendungsgebiet:**
  - Schreiben und Ausführen automatisierter Tests von Programmkomponenten (Units) isoliert von anderen Programmeinheiten
  - die vorgefundenen Programmzustände werden mit den erwarteten verglichen und Abweichungen automatisch gemeldet
  - einfache Organisation der Testfälle für den Black-Box-Test einschließlich Erzeugung von Klassen, die eine Sammlung von Testfällen unterstützen
  - inkrementelle Programmentwicklung in kleinen Schritten (erst Tests schreiben, dann Code entwickeln; wiederholbare Tests, regressionsfähig)
  - Gewährleistung einer einheitlichen, flexiblen Testdokumentation
- **Softwarebasis:**
  - *Open Source* Test-Framework in junit.jar, Quellen in src.jar mit der Möglichkeit, es selbst zu erweitern (siehe www.junit.org)

Quelle: Link, J.: Softwaretests mit JUnit – Techniken der testgetriebenen Entwicklung; dpunkt.verlag 2005

Teststruktur von JUnit



Quelle: Aßmann, U.: Vorlesung Softwaretechnologie II SS06

Testklassen („Werkzeuge“) von JUnit

Test	Schnittstellenklasse, die es nach dem *Composite Pattern* erlaubt, beliebig viele Testumgebungs- und Testfallobjekte zu einer umfassenden Test-Hierarchie zu kombinieren
TestSuite	Zusammenfassung beliebig vieler Tests in einer Klasse, um sie dann gemeinsam ausführen zu können. Hinzufügen beliebig vieler Testfälle und selbst weiterer Testsuites, womit sie eine Reihe von Tests zusammenführt
TestCase	Sammlung von Testfällen, gruppiert die Testfälle um eine gemeinsame Menge von Testobjekten. Der Testfall wird aus einer bestimmten Konfiguration von Objekten aufgebaut, gegen die der Test läuft. Damit wird das Verhalten der Testobjekte ermittelt
TestDecorator	- erlaubt Verwendung gleichzeitig mehrerer Erweiterungen - fungiert als Testframework einer Oberklasse - implementiert das Decorator-Muster nach Gamma

Lebenszyklus eines Testfalls:

1. **Testfallerzeugung:** Framework erzeugt für Testmethoden der zugehörigen Testklasse jeweils ein eigenes Objekt der Klasse
2. **Testlauf:** JUnit führt die gesammelten Testfälle voneinander isoliert aus. Reihenfolge der Ausführens der Testfälle ist undefiniert.

Quelle: Westphal, F.: Unit Testing mit JUnit; URL: http://www.FrankWestphal.de

Funktionalität von LOGISCOPE

- **Hersteller:** Telelogic North America Inc., Irvine, USA (Hersteller des Requirement Management Systems DOORS)
- **Merkmale:**
  - Durchgängiges Werkzeug für die Phasen **Entwicklung, Testung** und **Wartung**
  - Leicht zu benutzendes, interaktives Werkzeug, das ermittelt
    – die Testeff zienz,
    – die Zweigüberdeckung,
    – die Testüberdeckungsvervollkommnung,
    – die Def nition neuer Tests.
  - Der ausgeführte Test liefert
    – Trace-Protokolle,
    – ungetestete Zweige im Quellcode,
    – Programmlogik in Form von Aufruf- und Steuerf uss-Graphen,
    – Programm-Komplexität auf Basis wählbarer Metriken.
  - Unterstützt die **Testvorbereitung und -auswertung** durch
    – Instrumentierung des Compiler-Prozesses,
    – Def nition neuer Testszenarios,
    – graphische Auswertung der summarischen Testergebnisse,
    – automatische Erstellung der Testdokumentation.

Werkzeuge von LOGISCOPE

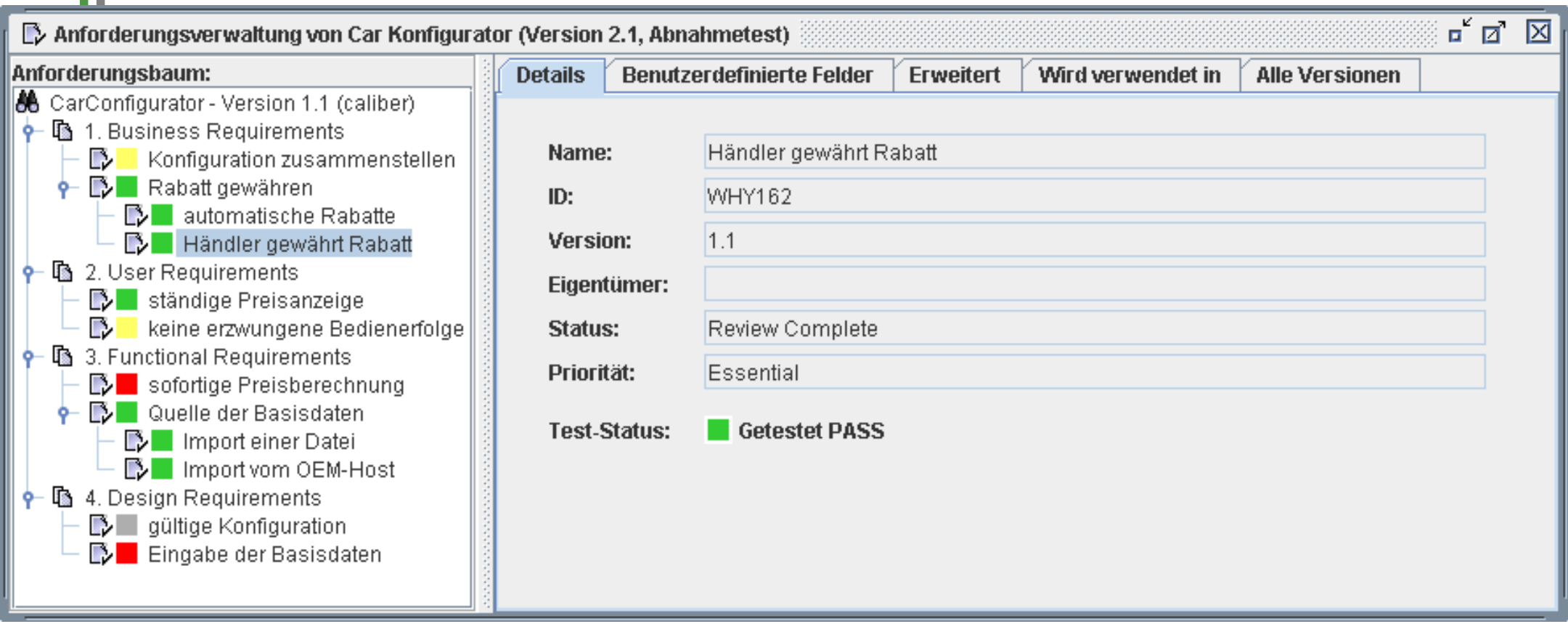
Test-Aktivität	Bezeichnung	Aufgabe
Testorga-nisation	***ProjectOrganizer***	bereitet die zu analysierende Applikation vor durch die Def nition von Testdatenf les, die Integration externer Tools,wie z.B. Debugger, Publishing Programme u.a.
	CodeChecker	verif ziert die Konformität einer Applikation gegen ein Qualitäts-Modell(z.B. Software-metriken, Empfehlungen ISO/IEC 9126, ISO-9001, DO-178B, ...)
Testfall-ermittlung	***RuleChecker***	def niert eine Menge einzuhaltender Codier-regeln, Namens- und Darstellungskonven-tionen. Auswahl aus Regel-Liste und direkte Anzeige im Quellcode.

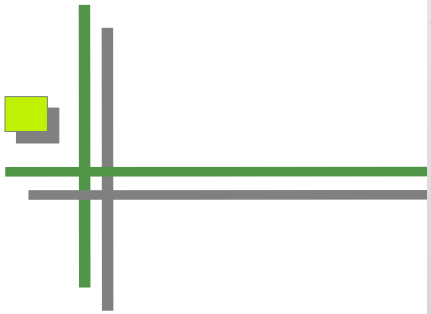
Werkzeuge von LOGISCOPE (2)

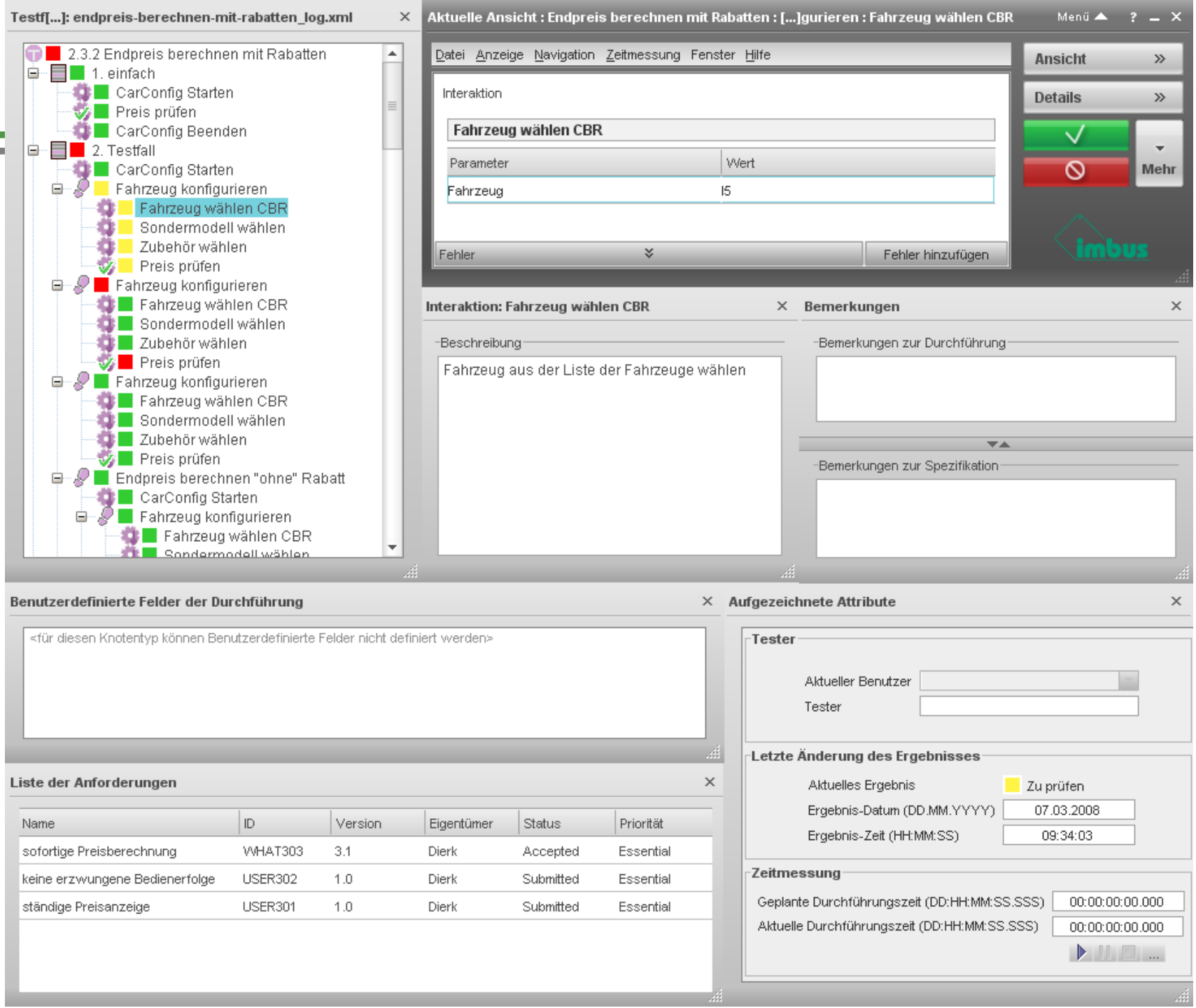
Test-Aktivität	Bezeichnung	Aufgabe
Test-durchführung	***TestChecker***	misst in Verbindung mit einem Debugger die Testüberdeckung in Echtzeit, zeigt im Quell-programm nicht überdeckte Wege an, gene riert Testberichte und übernimmt die Test-fall-Verwaltung.
	ImpactChecker	zeigt die Wirkung der Benutzung von Res-sourcen, wie Files, Funktionen, Datentypen, Konstanten. Variablen usw. Sie wird sowohl im Quellcode als auch in einem “Wirkungs”-Fenster angezeigt.
Testaus-wertung	***Viewer***	stellt sehr verschiedene textuelle und gra-phische Auswertungsmittel zur Verfügung. Er erzeugt Steuerf uss-Graphen, Komponen-ten-Ruf-Graphen, Auswertung von Metriken und visualisierte Vergleiche mit ausgewähl-ten Parametern des Qualitätsmodells(Kiviat-Graph).

imbus TestBench

http://www.imbus.de/produkte/imbus-testbench/hauptfunktionen/


Anforderungsverwaltung von Car Konfigurator (Version 2.1, Abnahmetest)
Anforderungsbaum:
CarConfigurator - Version 1.1 (caliber)
1. Business Requirements
Konfiguration zusammenstellen
Rabatt gewähren
automatische Rabatte
Händler gewährt Rabatt
2. User Requirements
ständige Preisanzeige
keine erzwungene Bedienerfolge
3. Functional Requirements
sofortige Preisberechnung
Quelle der Basisdaten
Import einer Datei
Import vom OEM-Host
4. Design Requirements
gültige Konfiguration
Eingabe der Basisdaten
Details
Benutzerdefinierte Felder
Erweitert
Wird verwendet in
Alle Versionen
Name:
Händler gewährt Rabatt
ID:
WHY162
Version:
1.1
Eigentümer:
Status:
Review Complete
Priorität:
Essential
Test-Status:
Getestet PASS




Testf[...]: endpreis-berechnen-mit-rabatten_log.xml
2.3.2 Endpreis berechnen mit Rabatten
1. einfach
CarConfig Starten
Preis prüfen
CarConfig Beenden
2. Testfall
CarConfig Starten
Fahrzeug konfigurieren
Fahrzeug wählen CBR
Sondermodell wählen
Zubehör wählen
Preis prüfen
Fahrzeug konfigurieren
Fahrzeug wählen CBR
Sondermodell wählen
Zubehör wählen
Preis prüfen
Fahrzeug konfigurieren
Fahrzeug wählen CBR
Sondermodell wählen
Zubehör wählen
Preis prüfen
Endpreis berechnen "ohne" Rabatt
CarConfig Starten
Fahrzeug konfigurieren
Fahrzeug wählen CBR
Aktuelle Ansicht : Endpreis berechnen mit Rabatten : [...]gurieren : Fahrzeug wählen CBR
Menü
Datei Anzeige Navigation Zeitmessung Fenster Hilfe
Ansicht
Details
Interaktion
Fahrzeug wählen CBR
Parameter
Wert
Fahrzeug
I5
Mehr
Fehler
Fehler hinzufügen
imbus
Interaktion: Fahrzeug wählen CBR
Beschreibung
Fahrzeug aus der Liste der Fahrzeuge wählen
Bemerkungen
Bemerkungen zur Durchführung
Bemerkungen zur Spezifikation
Benutzerdefinierte Felder der Durchführung
<für diesen Knotentyp können Benutzerdefinierte Felder nicht definiert werden>
Aufgezeichnete Attribute
Tester
Aktueller Benutzer
Tester
Letzte Änderung des Ergebnisses
Aktuelles Ergebnis
Zu prüfen
Ergebnis-Datum (DD.MM.YYYY)
07.03.2008
Ergebnis-Zeit (HH:MM:SS)
09:34:03
Zeitmessung
Geplante Durchführungszeit (DD:HH:MM:SS.SSS)
00:00:00:00.000
Aktuelle Durchführungszeit (DD:HH:MM:SS.SSS)
00:00:00:00.000
Liste der Anforderungen
Name
ID
Version
Eigentümer
Status
Priorität
sofortige Preisberechnung
WHAT303
3.1
Dierk
Accepted
Essential
keine erzwungene Bedienerfolge
USER302
1.0
Dierk
Submitted
Essential
ständige Preisanzeige
USER301
1.0
Dierk
Submitted
Essential

Werkzeug Sotograph für ergebnisorientierten Test

- **Entwickler und Hersteller:**
  - Software-Tomography GmbH, Cottbus; jetzt Hello2Morrow http://www.hello2morrow.com
- **Anwendungszweck:**
  - Generierung und Verwaltung von Testskripten und Skriptfragmenten für komplette statische und metrikbasierte Analysen
  - Gewährleistung einer einheitlichen, flexiblen Testdokumentation
  - Variable Auswertung auf Basis von (UML-)Modellen und Metrik-Browsern
- **Softwarebasis:**
  - Einsatz einer Datenbank als Test-Repository
  - Austausch von Qualitäts-Modellen mittels XML-Files
  - Source Code-Verwaltung mit SNiFF+
- **Beschreibungsmittel für Testskripte:**
  - Matrix, die Zustände und Methodenaufrufe systematisch gegenüberstellt
  - Überprüfung sämtlicher möglicher Zustandsübergänge
  - Nutzung zunächst für Java und C++, spätere Erweiterung möglich
- **Test-Auswertung:**
  - Endekriterium ist Maß der Abdeckung aller Testfälle der Matrix
  - Metrikbasierte graphische 3D-Visualisierung

Quelle: Simon, F., Lewerentz, C., Bischofberger, W.: Software Quality Assessments for System, Architecture, Design and Code; in Meyerhoff D., Laibarra, B. u. a.(Eds.): Software Quality and Software Testing in Internet Times. S. 230 - 249, Springer-Verlag, 2002

51.4 Simulation

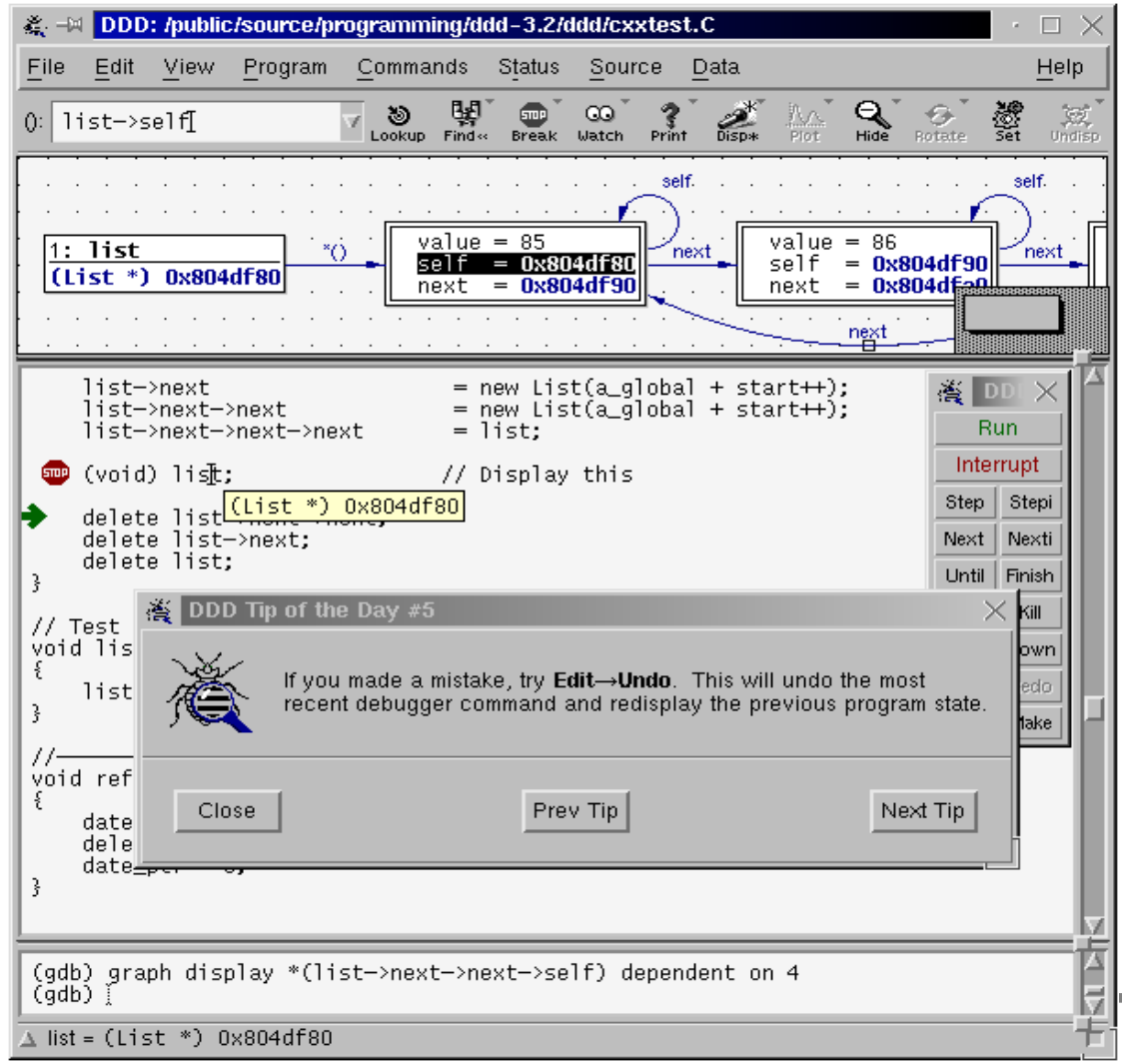
51.4.1 In-Vitro-Testläufe mit Debuggern

Entwanzer (Debugger)

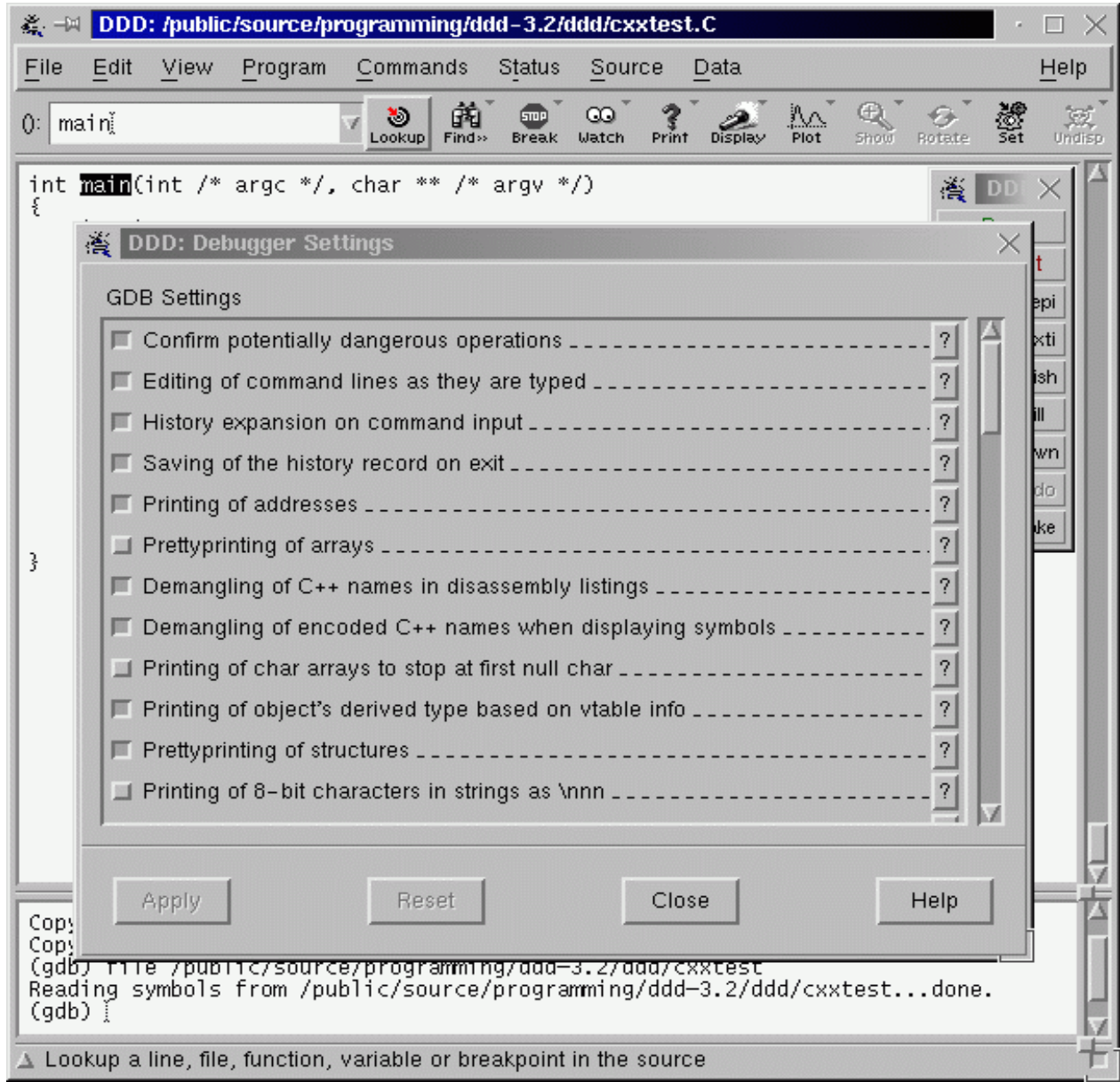
- Ein **Entwanzer (Debugger)** lässt ein Programm in-vitro ablaufen und kann es jederzeit unterbrechen
  - Man kann *breakpoints* setzen, Zeilen, an denen der Befehlszähler angelangt ist, und die den Ablauf stoppen
  - *watchpoints*: Zeitpunkte, an denen sich eine Variable ändert
  - Anschauen aller Variablen-, Register-, und Haldenwerte
  - Verändern derselben
- Gute Debugger funktionieren auch mit mehreren Threads, sodass Race Conditions gesucht werden können

Dynamic Display Debugger (DDD)

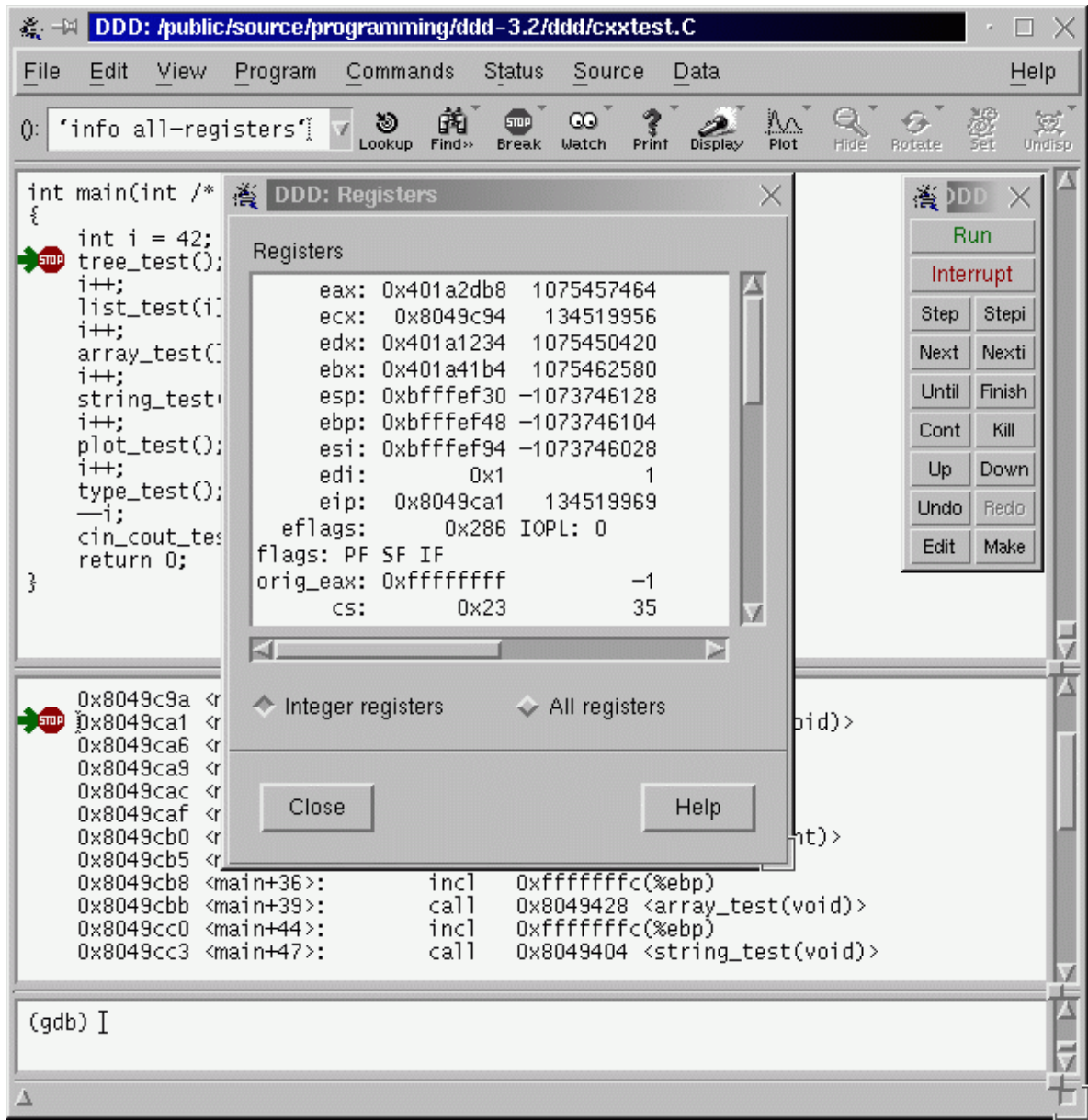
- ddd ist ein Visualisierungs-Front-end für mehrere andere Debugger
  - C/C++: GDB, DBX, WDB
  - Java: JDB
  - Perl: Perl debugger
  - bash: bashdb
  - make: remake
  - Python: pydb
- ddd zeigt Datenstrukturen an
  - mit Attributwerten
  - mit Verzeigerung



ddd Settings



ddd Registerwerte



The End

- Bananaware http://de.wikipedia.org/wiki/Bananaware