

# 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



SEW, © Prof. Uwe Aßmann

1

## 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=](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://de.wikipedia.org/wiki/Dynamisches_Software-Testverfahren)
- ▶ <http://www.testingstandards.co.uk/Component%20Testing.pdf> Britisher Standard mit schönen Definitionen



## 51.1 Aufgaben und Arten von Testwerkzeugen



SEW, © Prof. Uwe Aßmann

3

### **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!**

# 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**.

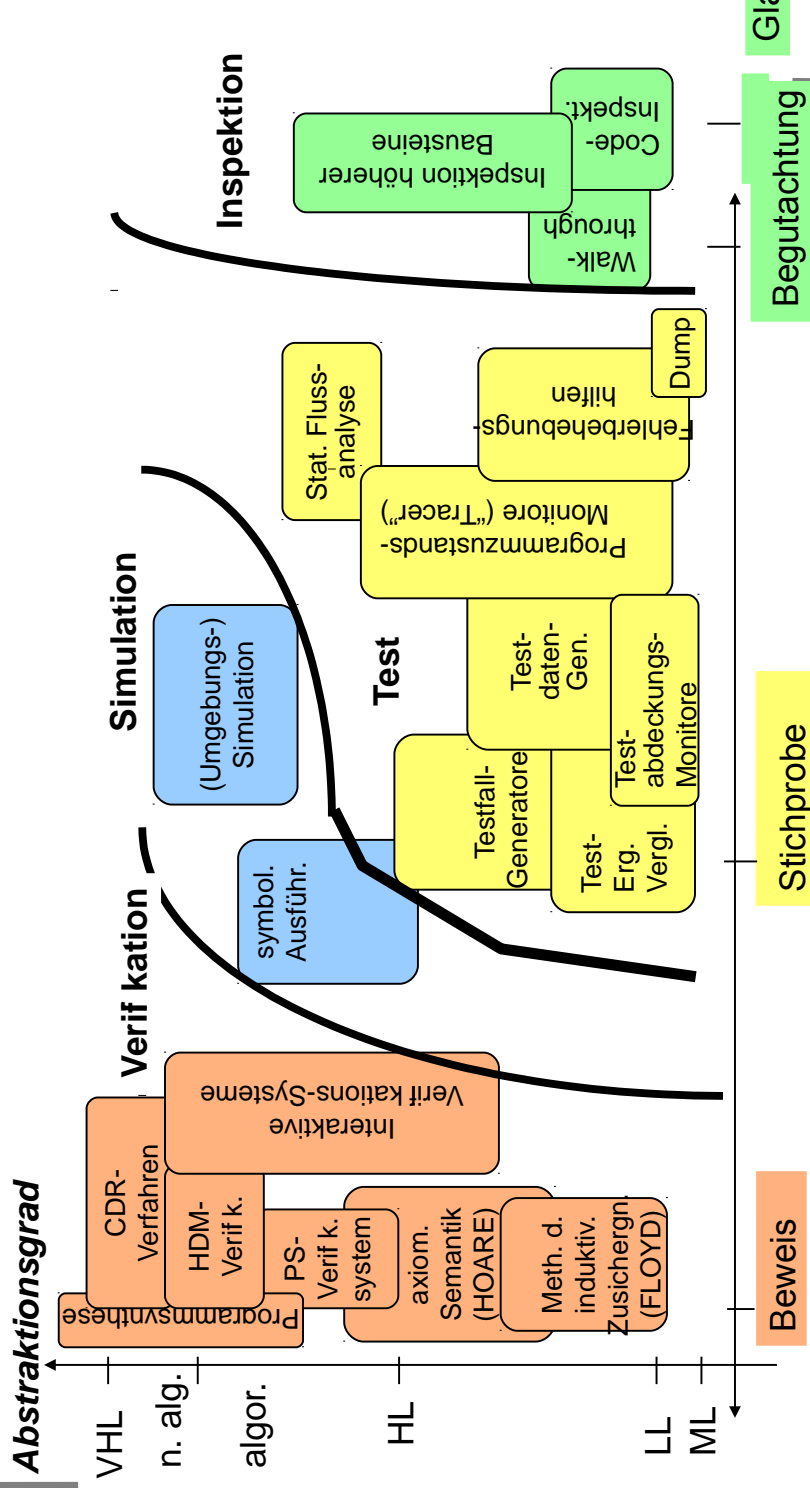
- **Statische Programmanalyse**, ohne dass das Programm ausgeführt wird.
- **Dynamische Programmanalyse** durch Ausführung (oder Simulation) in einer geeigneten Testumgebung mit ausgesuchten Testdaten.

“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	<b>Strukturabdeckung</b> 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

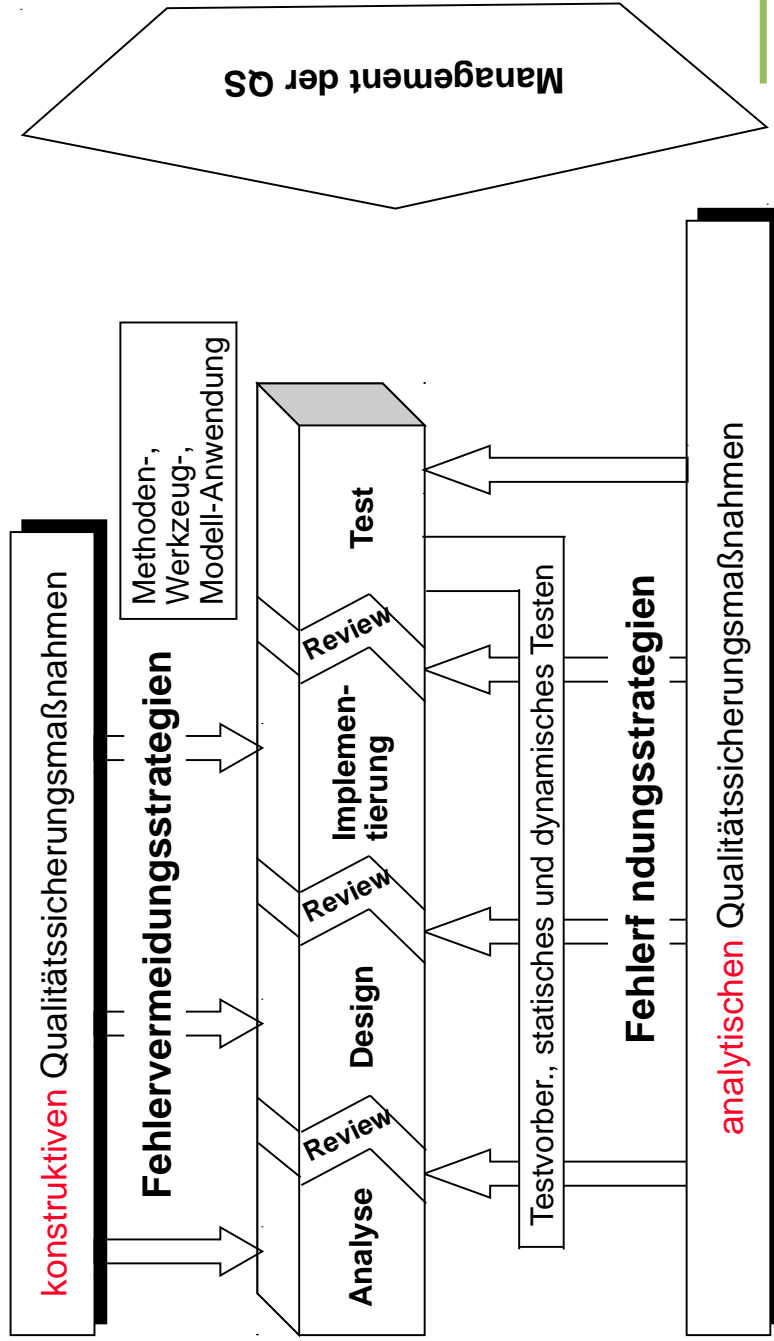
# Verifikations- und Validations-Techniken



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

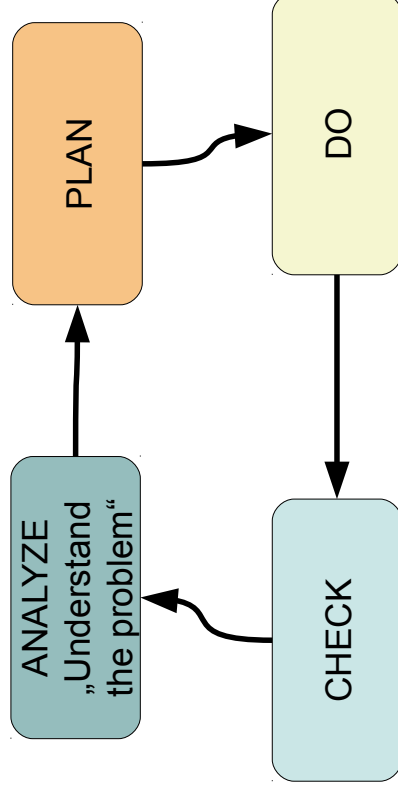
# Kategorisierung der QS-Maßnahmen

Maßnahmen der Software- Qualitätssicherung werden differenziert nach:

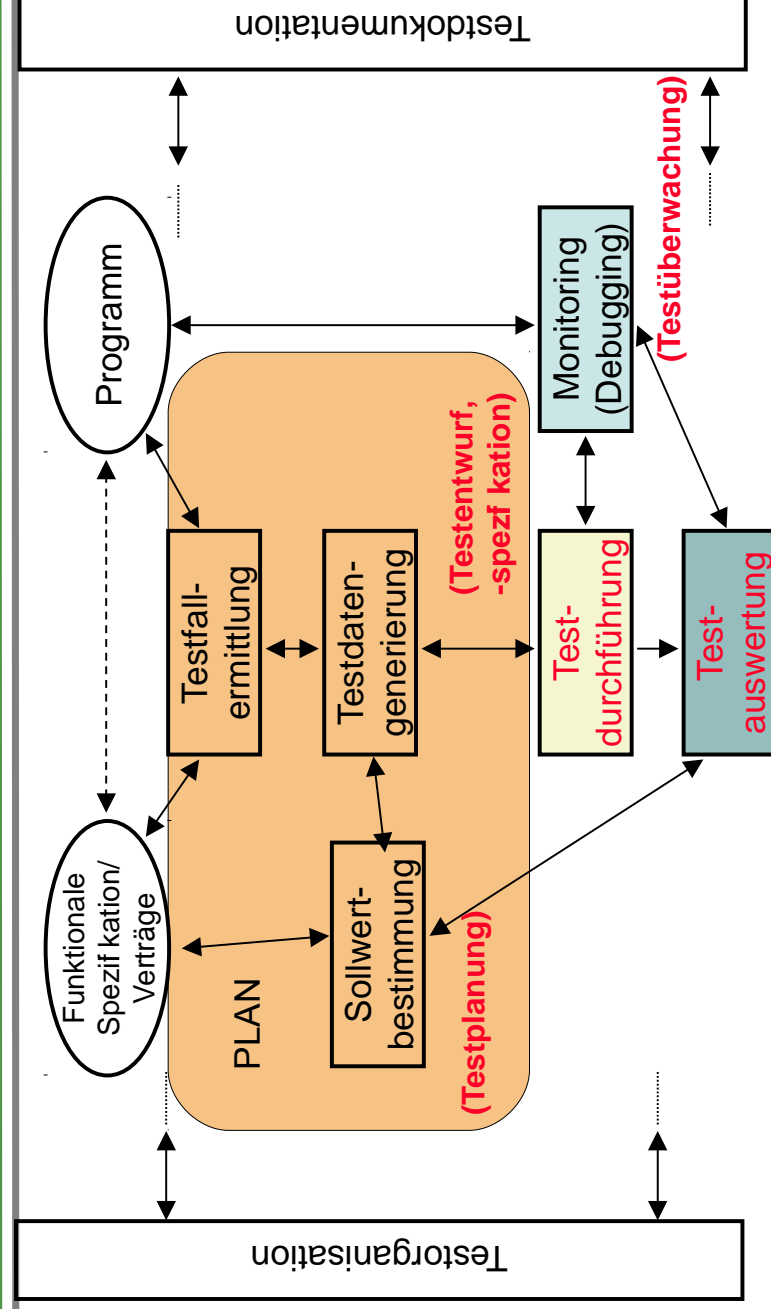


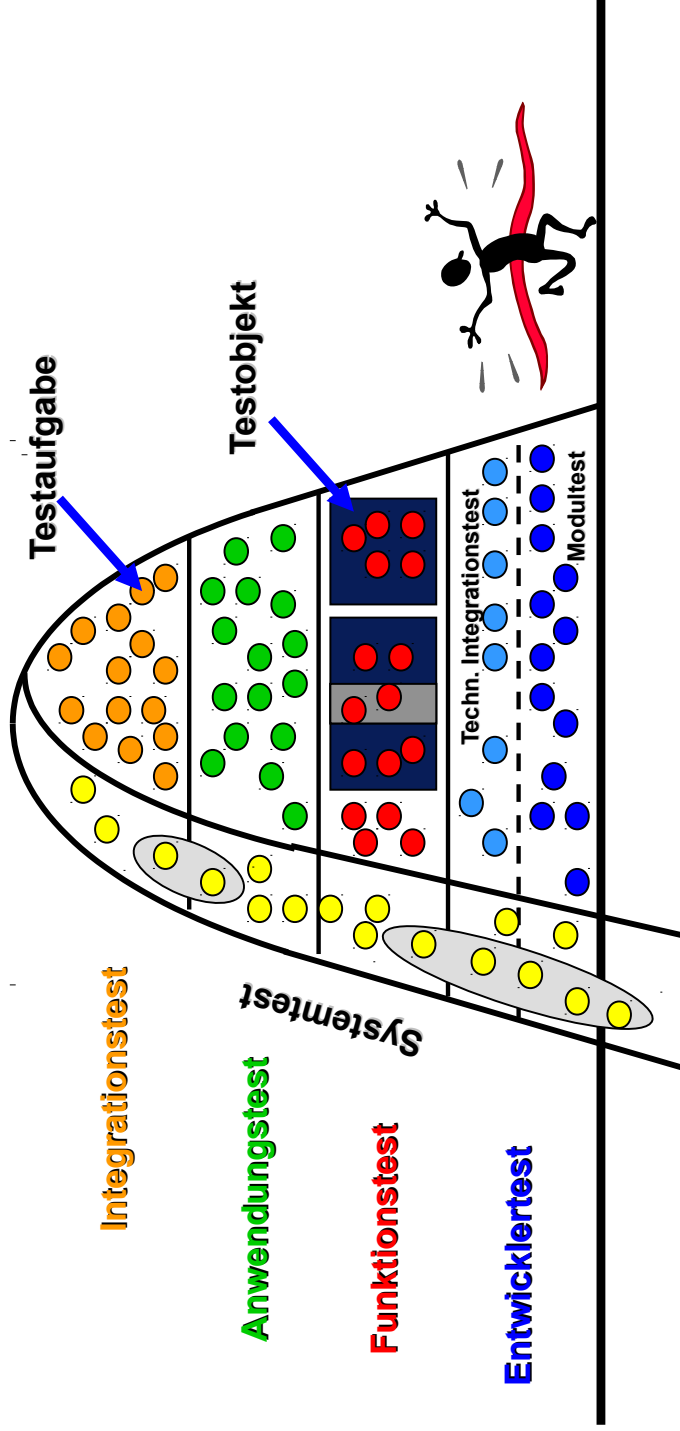
# Testing wendet den "Polya Cycle" an

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



## Test-Management





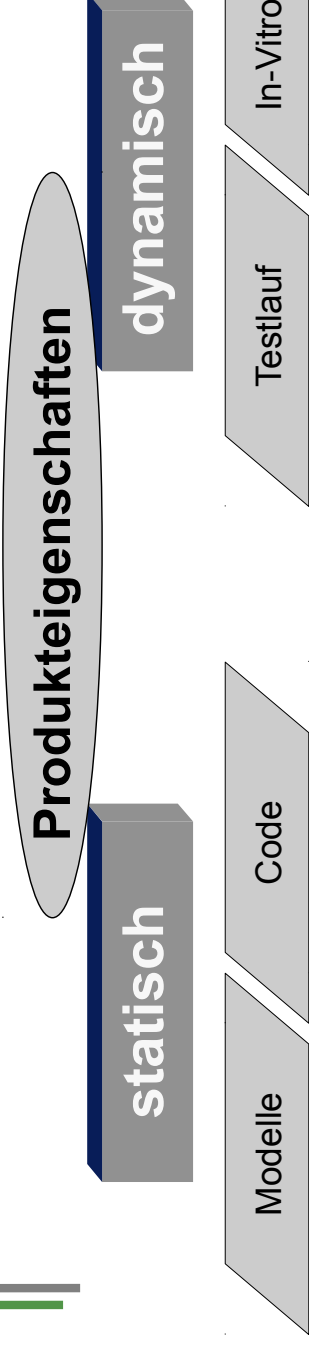
Quelle: Kugel, Thomas: Qualitätssicherung in der Praxis der Softwareerstellung; Vortrag der GI-Regionalgruppe Dresden am 18.10.2001; URL: <http://www.gi-dresden.de/files/181001.pdf>

## 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 Abgeschlossenheit 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 **Testobjekte** 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 Einfusses 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



- Kompilierbarkeit
- Änderbarkeit
- Überprüfbarkeit
- Verständlichkeit
- Wartbarkeit
- Metriken
- Programmierrichtlinien

- Lauffähigkeit
- Umsetzung der Anforderungen
- Schnittstellenkonformität
- Laufzeit / Performance
- Speicherverwaltung / Memory Leaks
- Robustheit
- Korrektheit der Algorithmen

## Testendekriterien

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>

Prof. U. Alßmann, SEW

13

## 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)

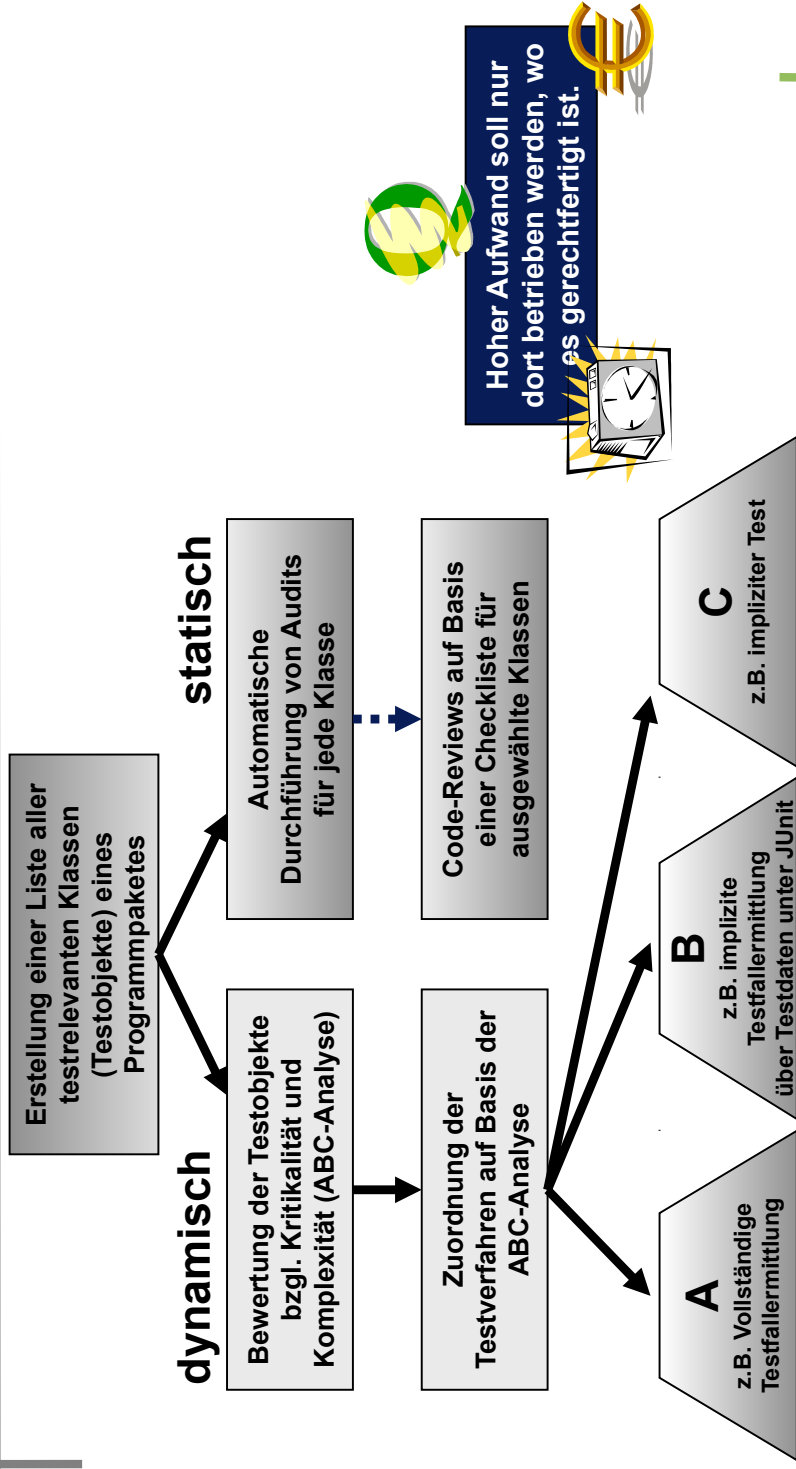
Wo sind die Testdaten ?

In den Variablen, die Zustände beschreiben  
In den Methodenparametern





# 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/181008/kugel.pdf>, SEW

17

## 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 Komponentenkonfigurationen hinweg, zwischen Produkten, sollten möglich sein
- ▶ **Schnelle und flexible 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 Logfile 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
<b>JUnit</b>	(K. Beck, E. Gamma) Open Source	www.JUnit.org
<b>JUnitDoclet</b>	Objectfab Dresden	www.objectfab.org
<b>Coverlipse</b>	Sourceforge	coverlipse.sourceforge.net
<b>DDTunit</b>	Sourceforge	http://ddtunit.sourceforge.net/ et/

**Weitere Nachweise:** [http://www.cetus-links.org/oo\\_testing.html#oo\\_testing\\_major\\_anchor\\_testing\\_utilities\\_tools](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
--	-------------	-----

<b>SilkTest</b>	Segue Software	<a href="http://www.segue.com">www.segue.com</a>
<b>TestBench</b>	Imbus	<a href="http://www.imbus.de">www.imbus.de</a>
<b>Visual 2000</b>	McCabe & Associates, USA	<a href="http://www.mccabe.com">www.mccabe.com</a>
<b>Cantata++</b>	IPL, Bath, UK	<a href="http://www.iplbath.com">www.iplbath.com</a>
<b>ClickTracks</b>	ClickTracks Analytics, Inc.CA	<a href="http://www.clicktracks.com">www.clicktracks.com</a>

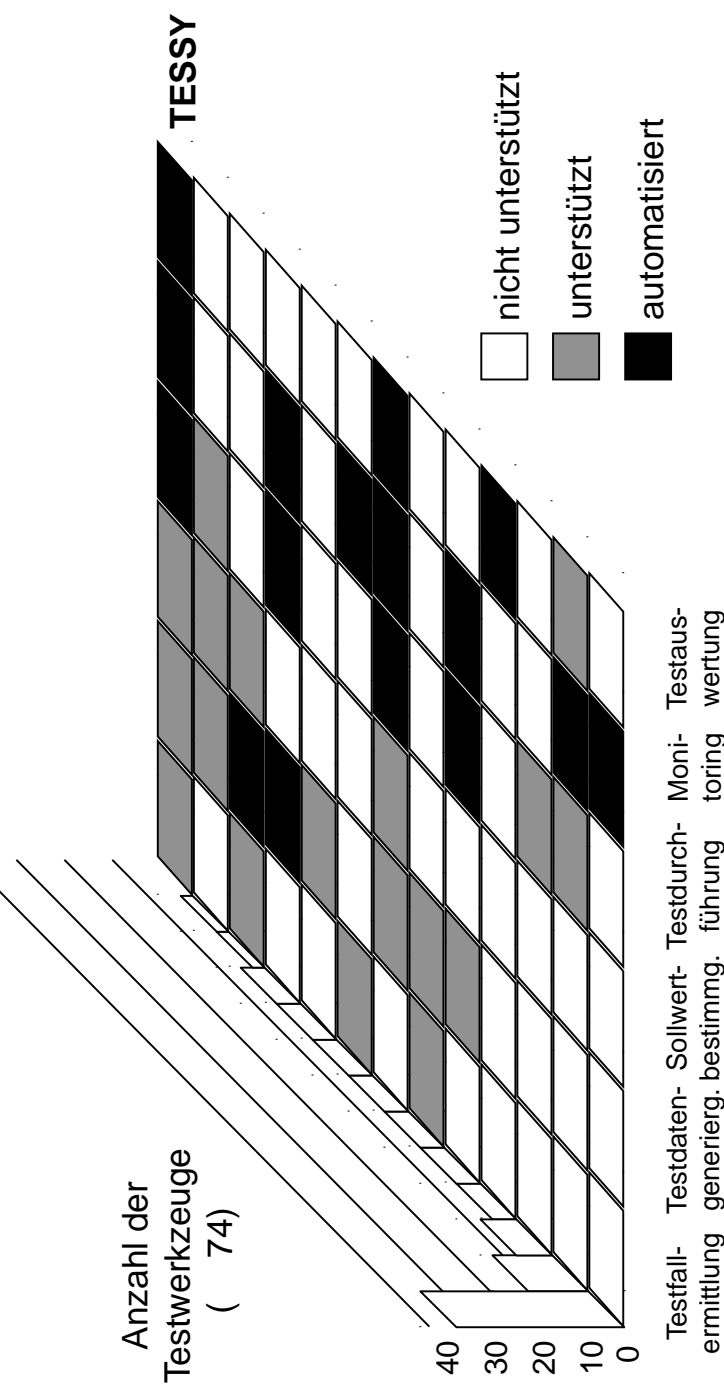
**Weitere Nachweise:** [http://www.cetus-links.org/oo\\_testing.html#oo\\_testing\\_major\\_anchor\\_testing\\_utilities\\_tools](http://www.cetus-links.org/oo_testing.html#oo_testing_major_anchor_testing_utilities_tools)  
<http://www.testingfaqs.org/>  
<http://www.imbus.de/tool-list.shtml>

## 51.2.1 Die Klassifikationsbaum- Methode

## 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 strukturierten Testverfahren** sowie der **Klassifikationsbaum-Methode**
- ▶ **Automatisierung von Regressionstests**
- ▶ Prinzipielles Test-Vorgehen:
  - Ausgangspunkt **funktionsorientierte** Testfälle
  - **strukturierte** Testfallermittlung nach der Klassifikationsbaum-Methode
  - Bestimmung der **Programmüberdeckung** nach Auswertung der Durchlaufhäufigkeiten
  - **Wiederholung** funktionsorientierter(1) oder strukturiertorientierter(2) Testfälle bis **maximale** Überdeckung der Programmzweige erreicht.

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

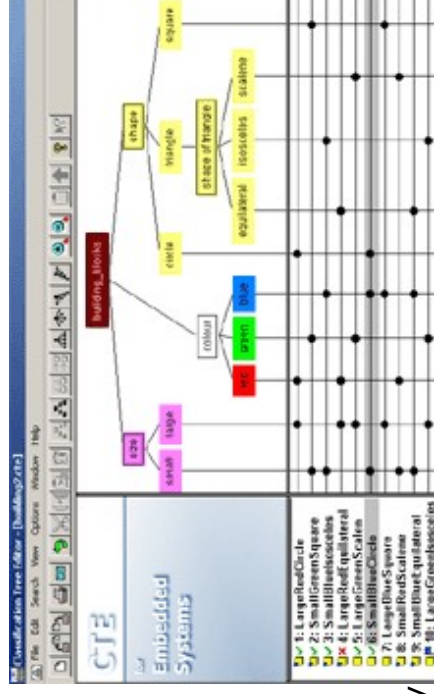


Testfall-ermittlung: Testdaten-generierung, Sollwert-bestimmung, Testdurchführung, Monitoring, Testauswertung

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

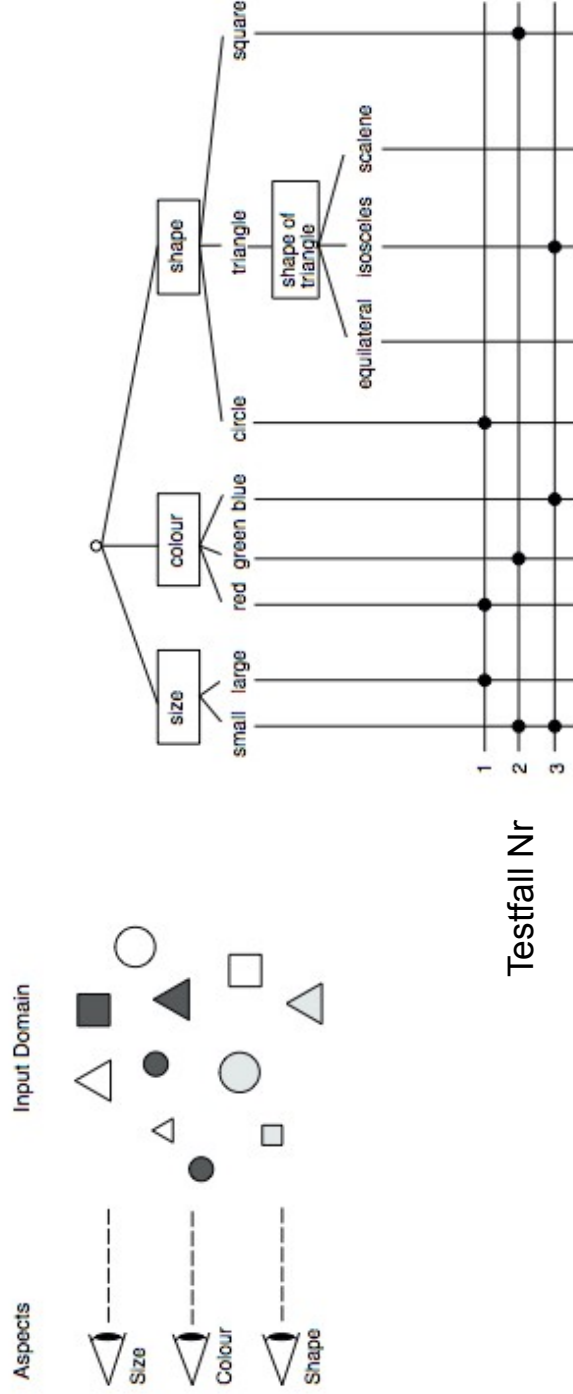
# Klassifikationsbaum (Classification Tree)

- ▶ Einteilung der Testdaten in Kategorien
  - <http://de.wikipedia.org/wiki/Klassifikation> nsbaummethode
  - 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



## 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

<b>Editor CTE</b>	Vervollständigung und Überwachung des <b>Klassif kationsbau- mes</b> → systematische Def. von funktionsorient. Testfällen → Erstellen Klassif.baum für aktuelles Testobjekt → Generierung von Testfällen
<b>Environment- Editor</b>	Organisation <b>testvorbereitender</b> Festlegungen zur Testumgebung des Testobjekts (Unit-Test)
<b>TESSY-System</b>	Ermittlung der Exportschnittstelle durch Parsen der Quellen → Funktionen (mit globalen Variabl., Parametern, Rückgabewerten und Datentypen) bilden eigentliche <b>Testobjekte</b>
<b>Testdaten- Editor TDE und Browser</b>	<b>Eingabe</b> konkreter <b>Testdaten</b> und <b>Sollwerte</b> zu jedem def nieren Testfall des Testobjektes <b>Browserfenster</b> dienen getrennter Eingabe der Daten und übersichtlicher parametergesteuerter Auswahl der Testbedingungen
<b>Monitoring EXP</b> (execution panel)	Nach Auswahl des Testfalls generieren des Testtreibers → <b>Test- durchführung</b> mit Messung der Zweigüberdeckung → Registrierung der Ergebnisse in Echtzeit → Protokollierung → Herstellung Ausgangszustand
<b>Testauswertung EVP</b> (evaluation panel)	Generieren der <b>Testdokumentation</b> unterschiedlicher Granularität → 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/Kontrollflussorientierte_Testverfahren)
- [http://de.wikipedia.org/wiki/Dynamisches\\_Software-Testverfahren](http://de.wikipedia.org/wiki/Dynamisches_Software-Testverfahren)

### Steuerflussorientierter Test (code coverage)

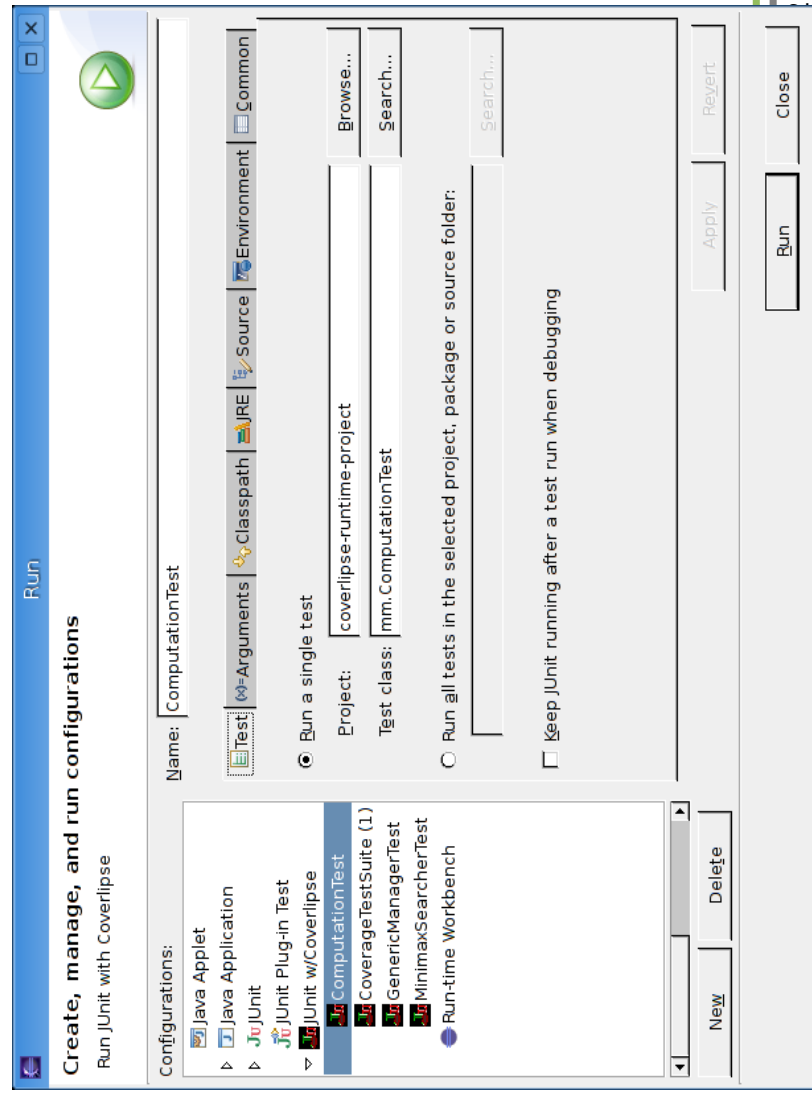
Überdeckungs-einheit	Arbeitsweise	Zweck
<b>Anweisung</b>	Möglichst viele Anweisungen werden mit Testfällen überdeckt	Entdeckung toten Codes
<b>Bedingung (Alternative)</b>	Jede alternative Belegung einer Bedingung wird durch einen Testfall getestet	Alle Kanten des Steuerfluss-Graphen werden überdeckt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

# 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 <i>in Prädikaten</i> getestet	Einfluss der Variable auf den Steuerfluss
All c-uses	Für eine Definition einer Variablen werden alle Benutzungen <i>außerhalb von Prädikaten</i> getestet (in rechten Seiten oder in Zeigern auf linken Seiten)	Einfluss der Variable auf den Datenfluss

## Bsp.: Coverlipse basiert auf JUnit

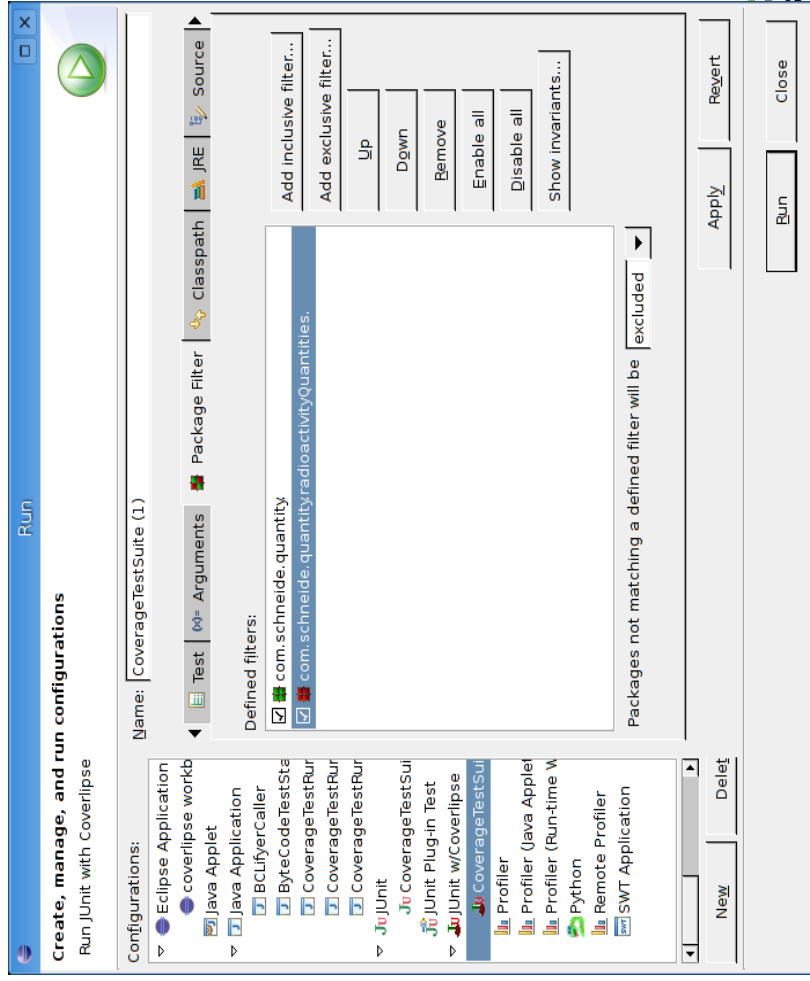
- ▶ Selektion von JUnit-Testfällen und deren Pfadabdeckungsanalyse





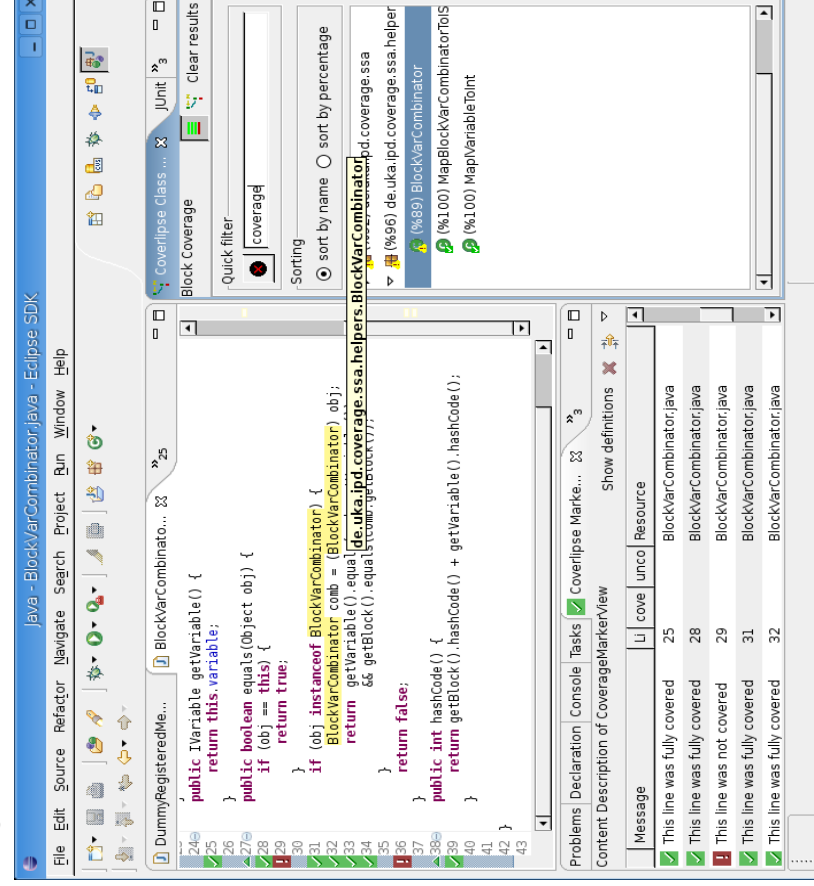
# Coverage Tools - Werkzeuge zur Pfadabdeckung

- ▶ Paketfilterung stellt die Pakete zur Pfadabdeckungsanalyse ein



# Coverlipse

- ▶ block coverage / statement coverage



# Coverlipse

- ▶ All-uses-coverage

The screenshot shows the Eclipse IDE with the 'All-Uses Coverage' view open. The 'Computation' class is selected, and the 'Coverlipse' tool is active. The code editor shows the following code:

```
public class Computation {  
    public int add(int arg1, int arg2) {  
        int result = arg1 + arg2; int meInt = 0;  
        int result2 = result;  
        if (arg1 == Integer.MIN_VALUE) {  
            new Integer(result);  
        }  
        int result3 = result2;  
        return result + meInt;  
    }  
    public int multiply(int n, int m) {  
        int result = 0;  
        for (int i = 0; i < m; i++) {  
            result = result + n;  
        }  
    }  
}
```

The 'All-Uses Coverage' view shows the following table:

Message	Line	covered uses	uncovered uses	Resource
Definition of the variable arg1	12	13 15		Computation.java
Definition of the variable arg2	12	13		Computation.java
Definition of the variable meInt	13	19		Computation.java
Definition of the variable result	13	14 19	16	Computation.java
Definition of the variable result2	14	18		Computation.java
Definition of the variable result3	18			Computation.java

# Coverlipse

- ▶ Problembeschreibung aus einem Use

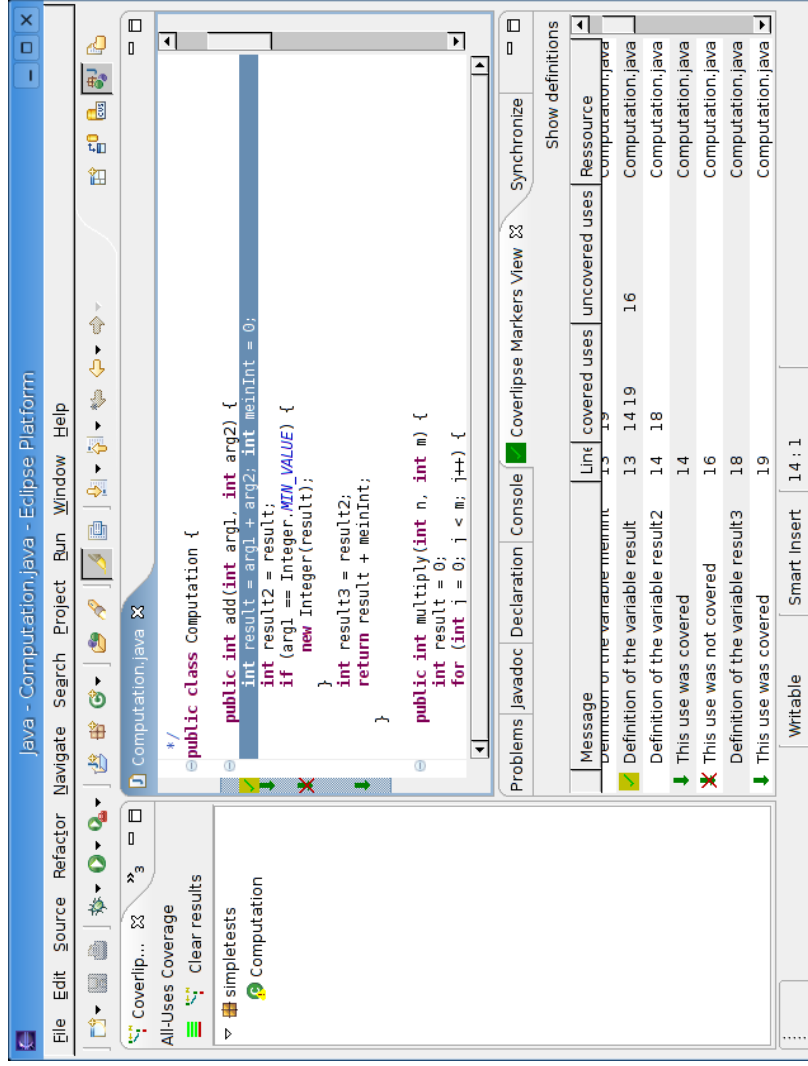
The screenshot shows the Eclipse IDE with the 'Problem Description' view open. The 'Computation' class is selected, and the 'Coverlipse' tool is active. The code editor shows the following code:

```
public class Computation {  
    public int add(int arg1, int arg2) {  
        int result = arg1 + arg2; int meInt = 0;  
        result  
    }  
}
```

The 'Problem Description' view shows the following table:

Message	Line	covered uses	uncovered uses	Resource
Definition of the variable arg1	12	13 15		Computation.java
Definition of the variable arg2	12	13		Computation.java
Definition of the variable meInt	13	19		Computation.java
Definition of the variable result	13	14 19	16	Computation.java
Definition of the variable result2	14	18		Computation.java
Definition of the variable result3	18			Computation.java

- ▶ all-uses-coverage information



```
public class Computation {
    public int add(int arg1, int arg2) {
        int result = arg1 + arg2;
        int mainInt = 0;
        if (arg1 == Integer.MAX_VALUE) {
            new Integer(result);
        }
        int result3 = result2;
        return result + mainInt;
    }

    public int multiply(int n, int m) {
        int result = 0;
        for (int i = 0; i < m; i++) {

```

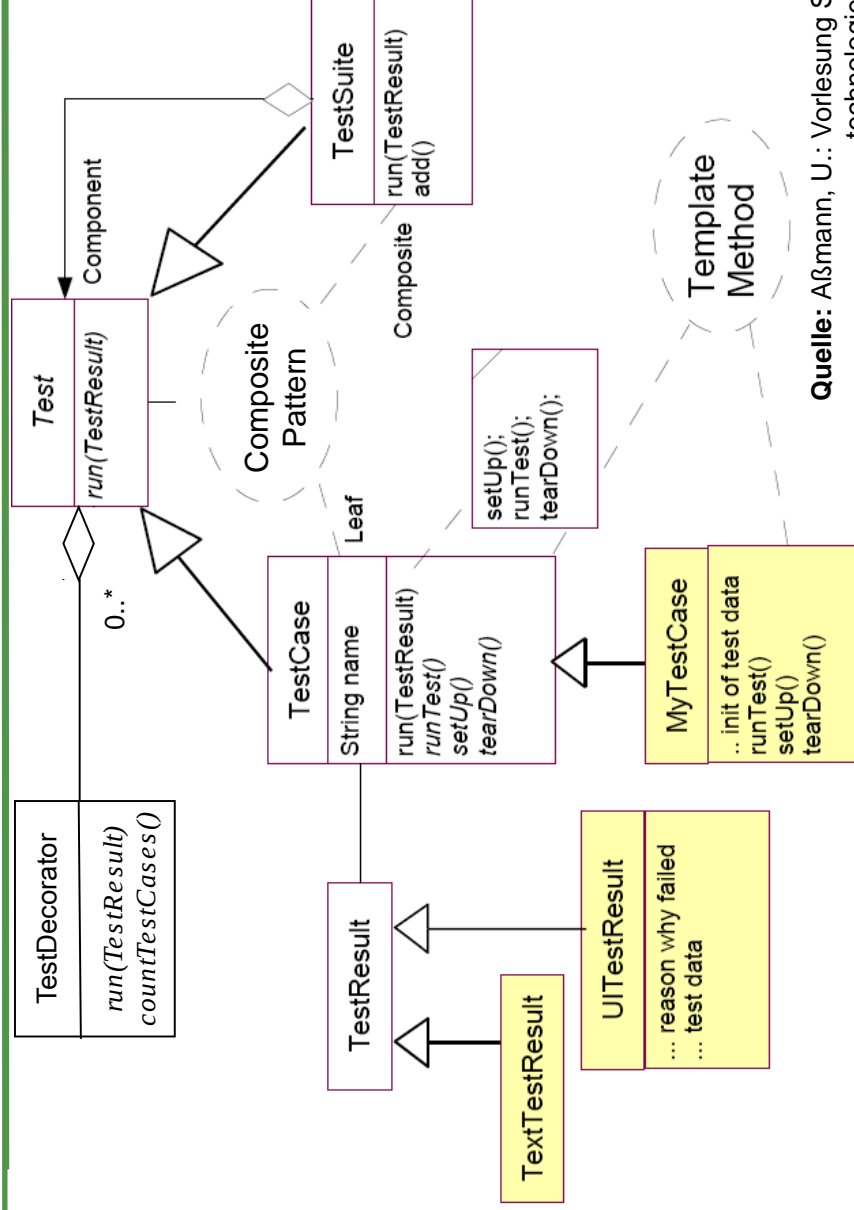
Message	Line	covered	uses	uncovered	uses	Resource
Definition of the variable mainInt	13	14	19			Computation.java
Definition of the variable result	13	14	19		16	Computation.java
Definition of the variable result2	14		18			Computation.java
This use was covered	14					Computation.java
This use was not covered	16					Computation.java
Definition of the variable result3	18					Computation.java
This use was covered	19					Computation.java

## 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](http://www.junit.org))

## Teststruktur von JUnit



# Testklassen („Werkzeuge“) von JUnit

<b>Test</b>	Schnittstellenklasse, die es nach dem <i>Composite Pattern</i> erlaubt, beliebig viele Testumgebungs- und Testfallobjekte zu einer umfassenden Test-Hierarchie zu kombinieren
<b>TestSuite</b>	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
<b>TestCase</b>	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
<b>TestDecorator</b>	<ul style="list-style-type: none"><li>- erlaubt Verwendung gleichzeitig mehrerer Erweiterungen</li><li>- fungiert als Testframework einer Oberklasse</li><li>- implementiert das Decorator-Muster nach Gamma</li></ul>

## 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.

# 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 Testeffizienz,
    - die Zweigüberdeckung,
    - die Testüberdeckungsvervollkommnung,
    - die Definition neuer Tests.
  - Der ausgeführte Test liefert
    - Trace-Protokolle,
    - ungetestete Zweige im Quellcode,
    - Programmlogik in Form von Aufruf- und Steuerfluss-Graphen,
    - Programm-Komplexität auf Basis wählbarer Metriken.
  - Unterstützt die **Testvorbereitung und -auswertung** durch
    - Instrumentierung des Compiler-Prozesses,
    - Definition neuer Testscenarios,
    - graphische Auswertung der summarischen Testergebnisse,
    - automatische Erstellung der Testdokumentation.

## Werkzeuge von LOGISCOPE

Test-Aktivität	Bezeichnung	Aufgabe
Testorganisation	<b>ProjectOrganizer</b>	bereitet die zu analysierende Applikation vor durch die Definition von Testdatenfiles, die Integration externer Tools, wie z.B. Debugger, Publishing Programme u.a.
	<b>CodeChecker</b>	verifiziert die Konformität einer Applikation gegen ein Qualitäts-Modell(z.B. Softwaremetriken, Empfehlungen ISO/IEC 9126, ISO-9001, DO-178B, ...)
Testfallermittlung	<b>RuleChecker</b>	definiert eine Menge einzuhaltender Codierregeln, Namens- und Darstellungskonventionen. Auswahl aus Regel-Liste und direkte Anzeige im Quellcode.

## Werkzeuge von LOGISCOPE (2)

Test-Aktivität	Bezeichnung	Aufgabe
Testdurchführung	<b>TestChecker</b>	misst in Verbindung mit einem Debugger die Testüberdeckung in Echtzeit, zeigt im Quellprogramm nicht überdeckte Wege an, generiert Testberichte und übernimmt die Testfall-Verwaltung.
	<b>ImpactChecker</b>	zeigt die Wirkung der Benutzung von Ressourcen, wie Files, Funktionen, Datentypen, Konstanten. Variablen usw. Sie wird sowohl im Quellcode als auch in einem "Wirkungs"-Fenster angezeigt.
Testauswertung	<b>Viewer</b>	stellt sehr verschiedene textuelle und graphische Auswertungsmittel zur Verfügung. Er erzeugt Steuerungss-Graphen, Komponenten-Ruf-Graphen, Auswertung von Metriken und visualisierte Vergleiche mit ausgewählten Parametern des Qualitätsmodells(Kivi-Graph).



The screenshot shows the "Car Konfigurator" software interface. The title bar reads "Anforderungsverwaltung von Car Konfigurator (Version 2.1, Abnahmetest)". The interface is divided into two main sections: "Anforderungsbaum:" (Requirements Tree) and "Details".

**Anforderungsbaum:** This section shows a hierarchical tree of requirements for "CarKonfigurator - Version 1.1 (caliber)".

- 1. Business Requirements
  - Konfiguration zusammenstellen (yellow icon)
  - Rabatt gewähren (green icon)
  - automatische Rabatte (green icon)
  - Händler gewährt Rabatt (green icon, highlighted)
- 2. User Requirements
  - ständige Preisanzeige (green icon)
  - keine erzwungene Bedienerfolge (yellow icon)
- 3. Functional Requirements
  - sofortige Preisberechnung (red icon)
  - Quelle der Basisdaten (green icon)
  - Import einer Datei (green icon)
  - Import vom OEM-Host (green icon)
- 4. Design Requirements
  - gültige Konfiguration (grey icon)
  - Eingabe der Basisdaten (red icon)

**Details:** This section provides specific information for the selected requirement "Händler gewährt Rabatt".

- Name:** Händler gewährt Rabatt
- ID:** WHY162
- Version:** 1.1
- Eigentümer:** Review Complete
- Status:** Essential
- Test-Status:** ■ Getestet PASS

Navigation tabs at the top of the details section include: "Details", "Benutzerdefinierte Felder", "Erweitert", "Wird verwendet in", and "Alle Versionen".

The screenshot displays the Inbus software interface, which is used for managing and executing test cases. The main window is titled 'Testf[...]: endpreis-ber-echnen-mit-rabatten\_log.xml'. It features a tree view on the left showing a hierarchy of test cases, including '2.3.2 Endpreis berechnen mit Rabatten' and '1. einfach'. The central area shows a table with columns for 'Interaktion', 'Parameter', 'Wert', and 'Fehler'. Below this, there are sections for 'Interaktion: Fahrzeug wählen CBR', 'Beschreibung', and 'Bemerkungen'. A 'Benutzerdefinierte Felder der Durchführung' section contains a warning message: '<für diesen Knotentyp können Benutzerdefinierte Felder nicht definiert werden>'. A 'Liste der Anforderungen' table is also visible, listing requirements with columns for Name, ID, Version, Eigentümer, Status, and Priorität. The bottom right corner shows a 'Tester' section with fields for 'Aktueller Benutzer', 'Tester', and 'Letzte Änderung des Ergebnisses', along with a 'Zeitmessung' section for planned and actual execution times.

## 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 Methodenaufre 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



SEW, © Prof. Uwe Aßmann

49



## 51.4.1 In-Vitro-Testläufe mit Debuggern



SEW, © Prof. Uwe Aßmann

50

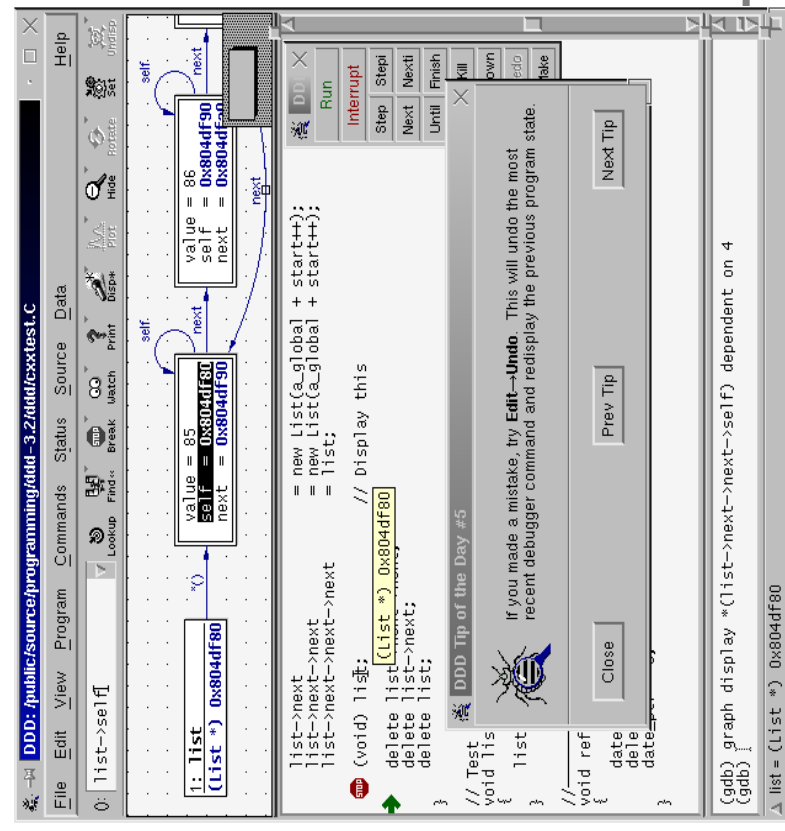


# Entwanzer (Debugger)

- ▶ 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 Haldeiwerte
  - 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: Debugger Settings

- Confirm potentially dangerous operations
- Editing of command lines as they are typed
- History expansion on command input
- Saving of the history record on exit
- Printing of addresses
- Prettyprinting of arrays
- Demangling of C++ names in disassembly listings
- Demangling of encoded C++ names when displaying symbols
- Printing of char arrays to stop at first null char
- Printing of object's derived type based on vtable info
- Prettyprinting of structures
- Printing of 8-bit characters in strings as \nnnn

Buttons: Apply, Reset, Close, Help

# ddd Registerwerte

DDD: Registers

Register	Value
eax	0x401a2db8
ecx	0x8049c84
edx	0x401a1234
ebx	0x401a41b4
esp	0xbffffef30
ebp	0xbffffef48
esi	0xbffffef94
edi	0x1
eip	0x8049ca1
eflags	0x286
ioPL	0
PF SF IF	0xfffffff
orig_eax	0xfffffff
cs	0x23

Integer registers: All registers

Buttons: Close, Help

# *The End*

- ▶ Bananaware <http://de.wikipedia.org/wiki/Bananaware>

