

71. Test Tools

1

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 13-0.1, 02.01.14

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



Softwareentwicklungswerkzeuge (SEW) © Prof. Uwe Aßmann

Obligatory Literature

2

- ▶ English Glossary: 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=
- ▶ IMBUS testing <http://www.imbus.de>



Literature

3

- ▶ 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> Britisher Standard mit schönen Definitionen

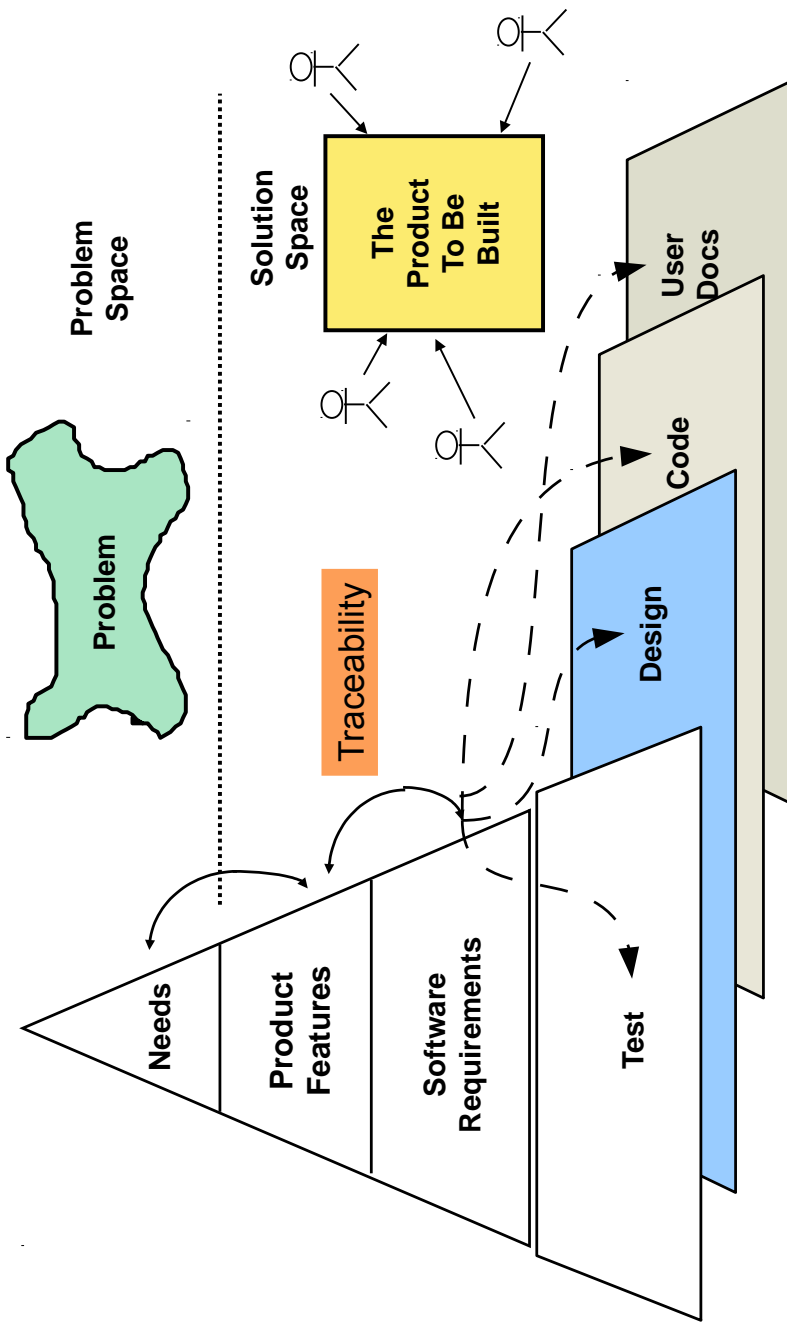
Prof. U. Asmann, Softwareentwicklungswerkzeuge (SEW)



Quality Management in MDSD (QM, Quality Assurance, QA)

4

- ▶ Quality Management consists of Requirements Management and Test Management on connected requirements, test, design, implementation models



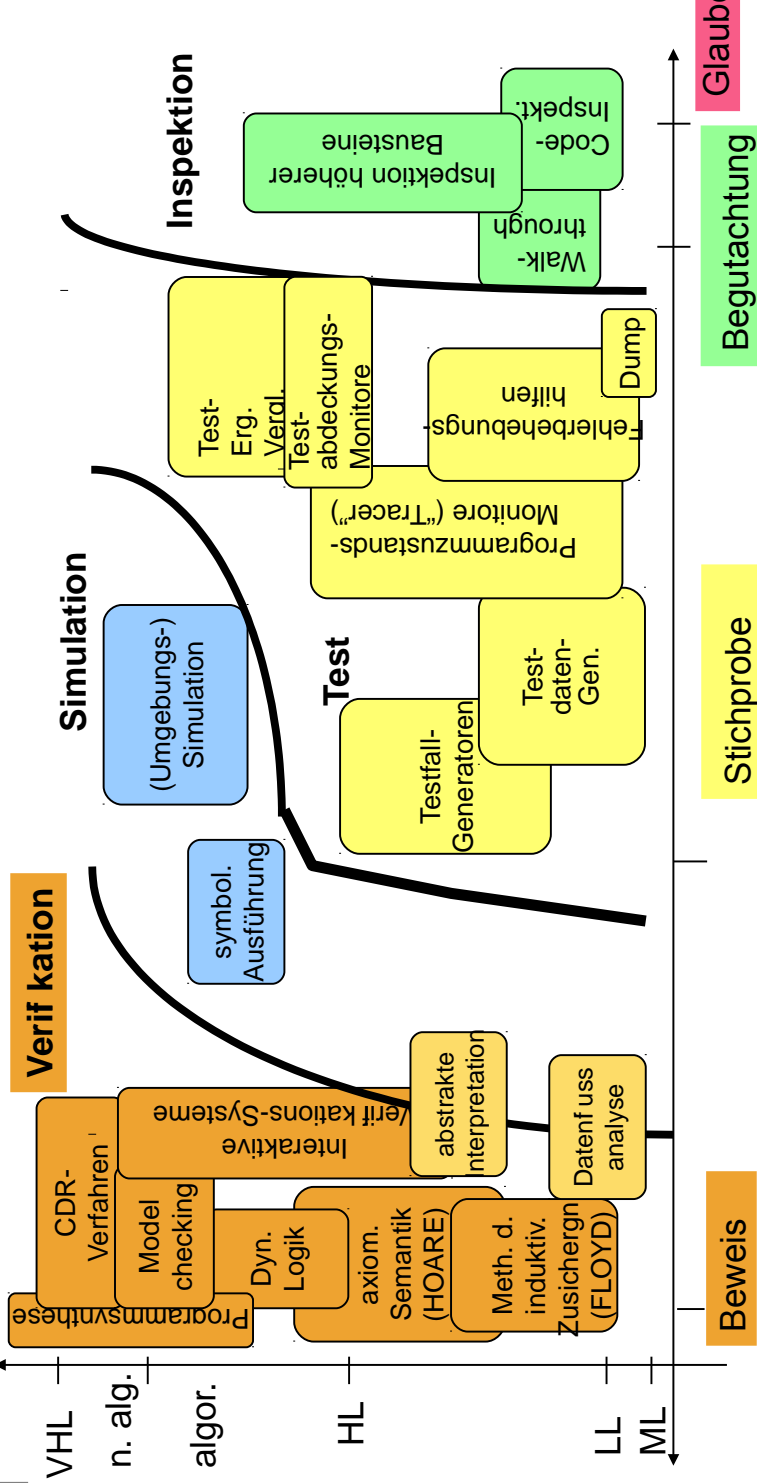
Prof. U. Asmann, Softwareentwicklungswerkzeuge (SEW)



Verifikations- und Validations-Techniken zur Qualitätssicherung (QS)

5

Abstraktionsgrad



Prof. U. Asmann, Softwareentwicklungswerkzeuge (SEW)



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

Validations-Formalisierung

71.1 Aufgaben und Arten von Testwerkzeugen

6



Softwareentwicklungswerkzeuge (SEW) © Prof. Uwe Asmann

Aufgaben von Testwerkzeugen

7

Testwerkzeuge sind Softwaresysteme, die die Validation der FURPS-Qualitätsmerkmale von Programmen auf Basis von **Test-Methoden unterstützen**.

- ▶ **Statische Programmanalyse**, ohne dass das Programm ausgeführt wird.
- ▶ **Dynamische Programmanalyse** durch Ausführung (oder Simulation) in einer geeigneten Testumgebung mit ausgesuchten Testdaten (**Stichproben-Test**).
- ▶ **Verfolgbarkeit** zu Anforderungen wird durch **Code-Modellabbildungen** gesichert

Statische Testmethode (Analyse)	Zweck
Symbolische Ausführung	Ausführung mit symb. Werten
Simulation	Ausführung mit konkreten Werten durch einen Simulator; ressourcenintensiv, aber realistisch (z.B. Debugging)
Abstrakte Interpretation	Ausführungen mit abstrakten Werteklassen
Datenflussanalyse	Abstrakte Interpretation mit Fokus auf Erreichbarkeit von Werten
Kontrollflussanalyse	Abstrakte Interpretation mit Fokus auf Ermittlung von Kontrollflussbedingungen
Metriken	Zählverfahren für statische Merkmale von Programmen



Prüftechniken im Dynamischen Test

8

- ▶ **Strukturorientierter Test** (Überdeckung, coverage test)
 - Kontrollflussorientiert (Maß für die Überdeckung des Kontrollflusses, code coverage)
 - Anweisungs-, Zweig-, Bedingungs- und Pfadüberdeckungstests
 - Datenflussorientiert (Maß für die Überdeckung des Datenflusses, data-flow coverage)
 - Defs-/Uses Kriterien, Required k-Tupels-Test, Datenkontext-Überdeckung
- ▶ **Funktionsorientierter Test** (Funktionsabdeckung, 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

“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	Strukturabdeckung Codeüberdeckung Anweisungsüberdeckg. Zweigüberdeckung Entscheidungsüberd. Weg-Überdeckung



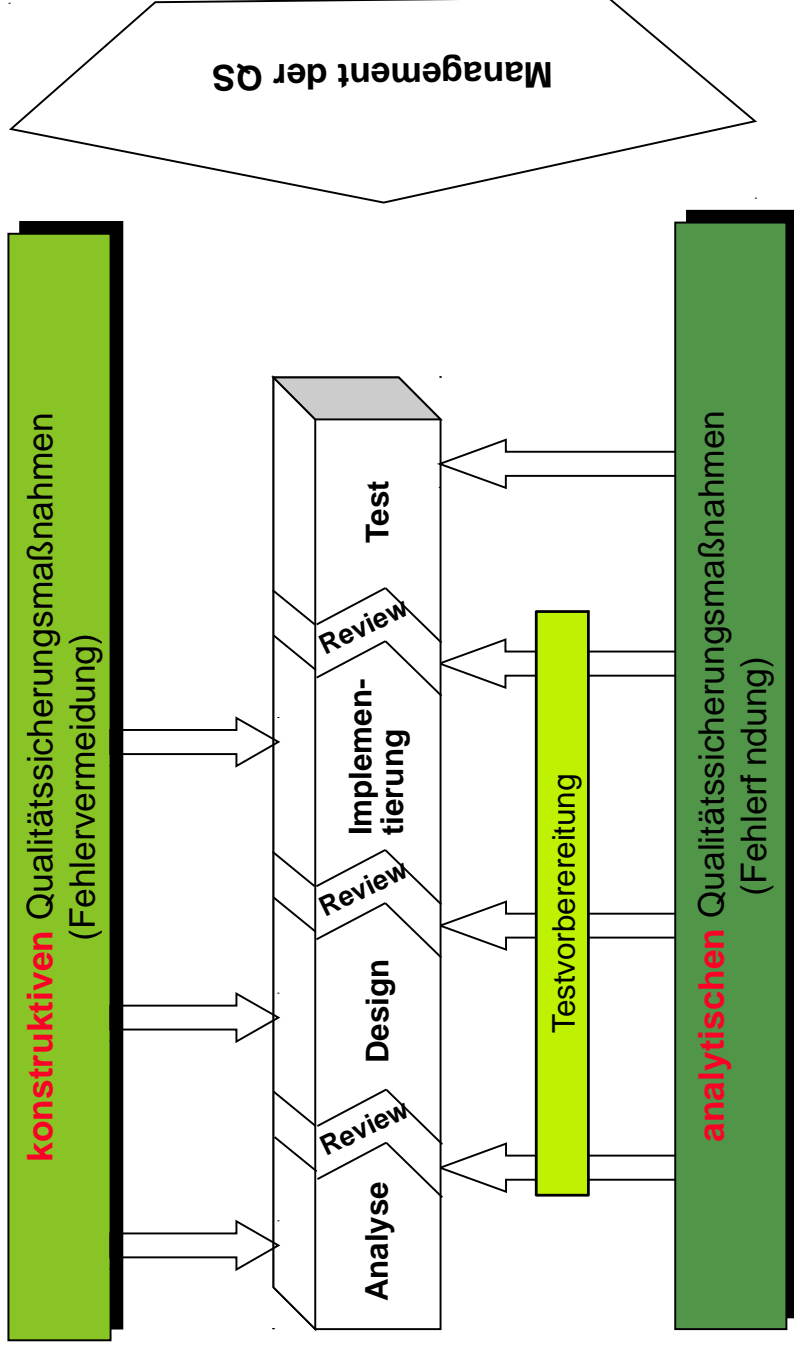
[Liggesmeyer]

<http://de.wikipedia.org/wiki/Softwareetest>

Kategorisierung der QS-Maßnahmen

9

Maßnahmen der Software- Qualitätssicherung (QS) werden differenziert nach:



Prof. U. Asmann, Softwareentwicklungswerkzeuge (SEW)

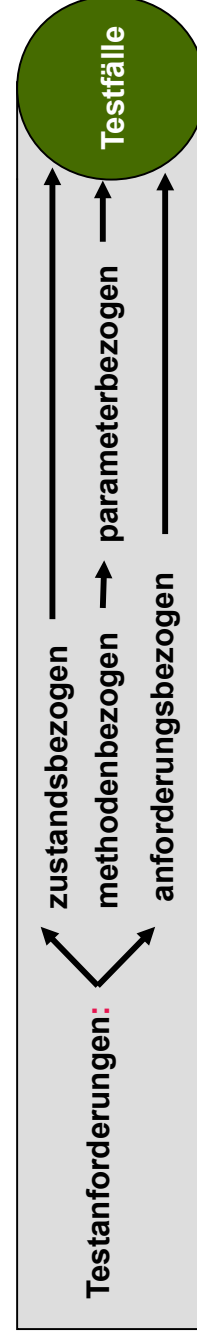


Der Testfall in objektorientiertem Test und das Testmodell

10

- Ein **Testfall** besteht aus:
 - 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
 - einer Prüfung, ob das veränderte Objekt dem erwarteten Endzustand entspricht (Soll/Ist-Vergleich)
- Statische Testfälle sind zu genauso modellieren, werden aber durch statische Analysen verifiziert
- Für QM müssen statische und dynamische Testfälle im Repository **persistiert** werden, d.h.
 - Es gibt ein *Testmodell* mit einem *Test-Metamodell*
 - Die Beziehung des Testmodells zu Requirements und zum Code (**Verfolgbarkeit, traceability**) muss durch die Evolution des Systems aufrecht erhalten werden

Prof. U. Asmann, Softwareentwicklungswerkzeuge (SEW)

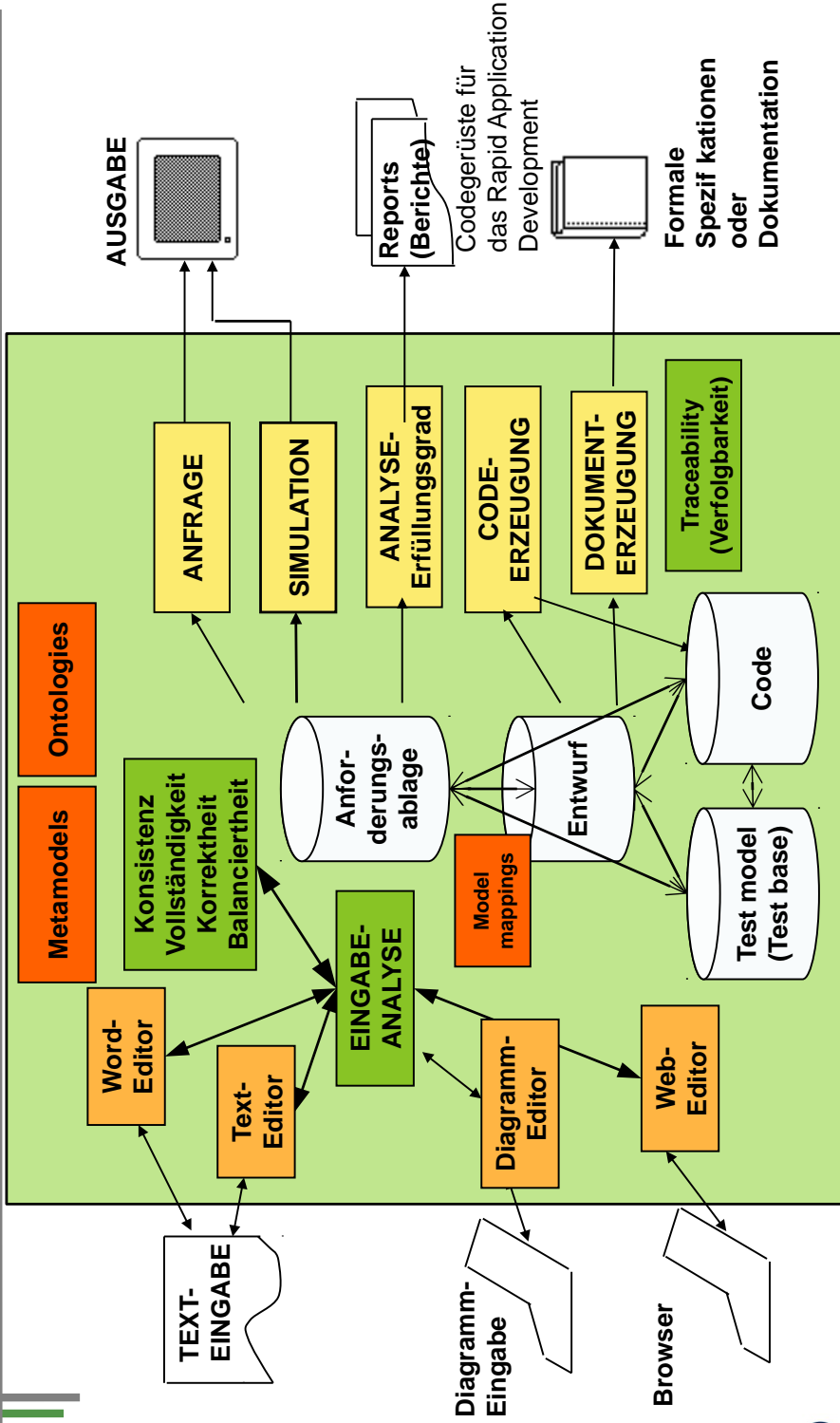


Quelle: Kugel, Thomas: Qualitätssicherung in der Praxis der Softwareerstellung; Vortrag der GI-Regionalgruppe Dresden am 18.10.2001; URL: <http://www.gi-dresden.de/files/181001.pdf>



QM with Traceability between Tests, Requirements, Design, and Code

11



Prof. U. Asmann, Softwareentwicklungswerkzeuge (SEW)

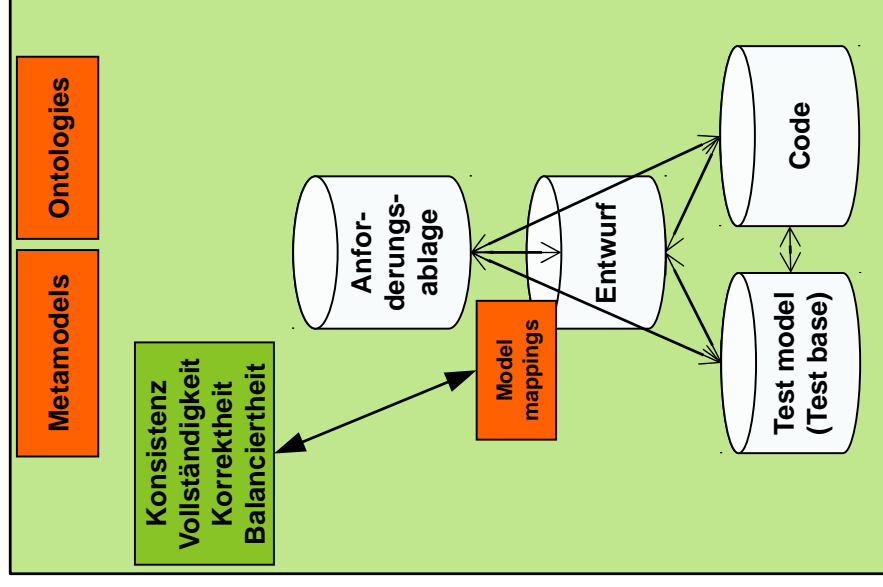


Qualitätsmanagement und Zertifizierung

12

- ▶ Für QM müssen über Modellabbildungen die Abdeckung aller Anforderungen durch statische und dynamische Testfälle nachgewiesen werden
 - Metamodelle und Ontologien
 - Erreichbarkeitsanalysen für Modellabbildungen
- ▶ QM ist Voraussetzung für Zertifizierung eines Produkts (sicherheitskritische Software für Flugzeug, Auto, Roboter)

Prof. U. Asmann, Softwareentwicklungswerkzeuge (SEW)



Anforderungen an Werkzeuge für den Entwicklertest

13

- ▶ **Generierung** von modulare Testskripten aus Testfällen
 - Die Testskripte, die die möglichen Sequenzen von Methodenaufrufen enthalten, sollen modular aufgebaut sein
 - Wiederverwendung über Komponentenkonfigurationen hinweg, zwischen Produkten, sollten möglich sein
- ▶ Einfache Kontrolle der **Abdeckung** aller Methodenaufrufe und Zustände (Matrix für alle Zustandsübergänge mit Test-Endekriterium)
- ▶ Automatisierte **regressionsfähige** Testausführung
 - Automatisierter Ist/Soll-Vergleich der Ausgaben des Programms

Prof. U. Asmann, Softwareentwicklungswerkzeuge (SEW)



71.2 Werkzeugeinsatz in einzelnen Testaktivitäten



14



Auswahl-Liste von Test-Frameworks

15

- ▶ 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/

Prof. U. Asmann, Softwareentwicklungswerkzeuge (SEW)

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

16

	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

Tracetricon
Framework

Tracetricon, Dresden.
Für automotiv Tests

www.tracetricon.de

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>

Prof. U. Asmann, Softwareentwicklungswerkzeuge (SEW)



71.2.1 Die Klassifikationsbaum-Methode

17

- Ein Metamodell zur Klassifikation von Testdaten
- Kann zur Persistierung von Testdaten genutzt werden, sowie zur Verfolgbarkeit

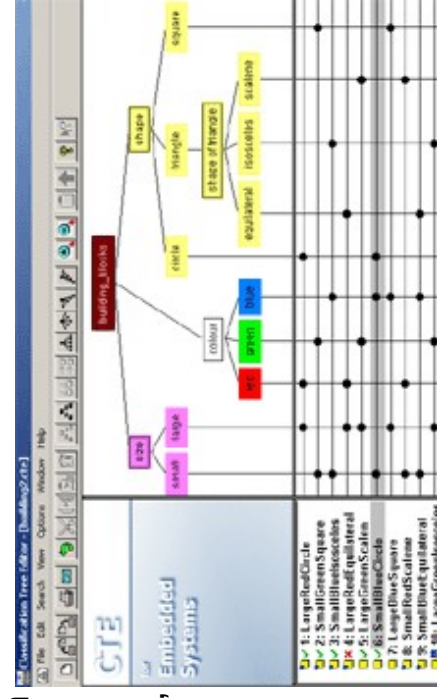


Softwareentwicklungswerkzeuge (SEW) © Prof. Uwe Aßmann

Klassifikationsbaum (Classification Tree)

18

- ▶ Einteilung der Testdaten in Kategorien
- ▶ Einteilung in ein Modell basierend auf dem Classification Tree Metamodel
- http://de.wikipedia.org/wiki/Klassifikation_sbaumethode
- 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. 271.
- Grochtmann, M. Test Case Design Using Classification Trees. STAR'94, 8 - 12 May 1994, Washington. <http://www.systematic-testing.de/documents/star1994.pdf>



Prof. U. Aßmann, Softwareentwicklungswerkzeuge (SEW)

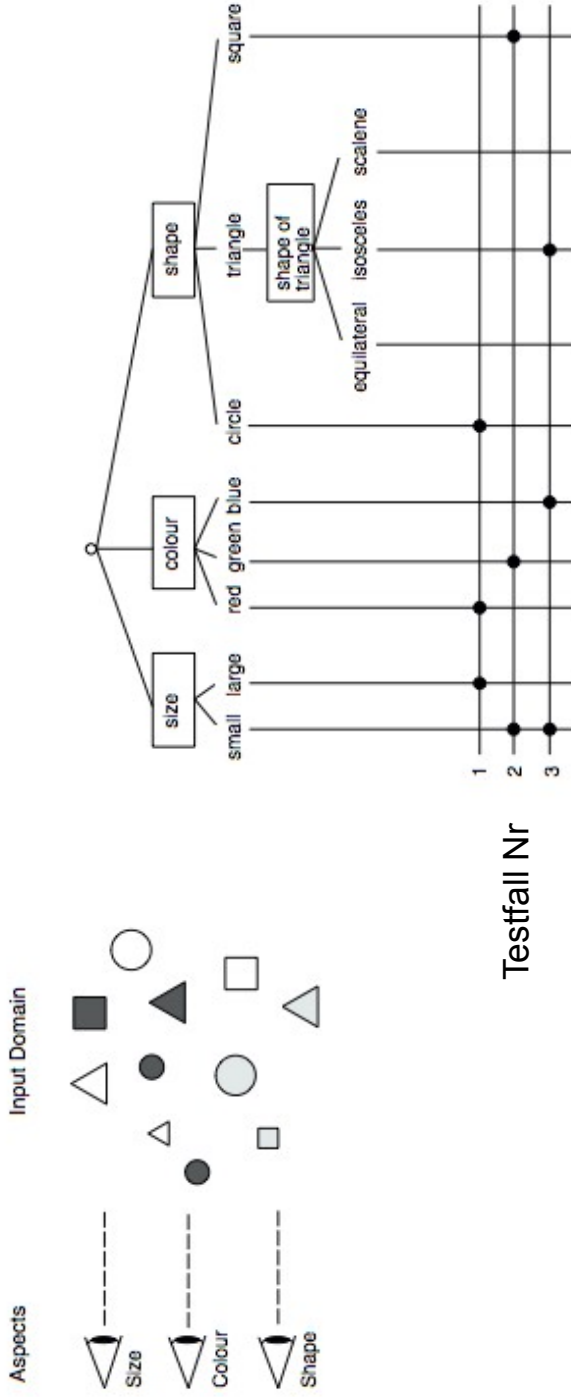


Kategorien (Facetten, Aspekte) der Testfalldaten

19

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

▪



Prof. U. Asmann,



20

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

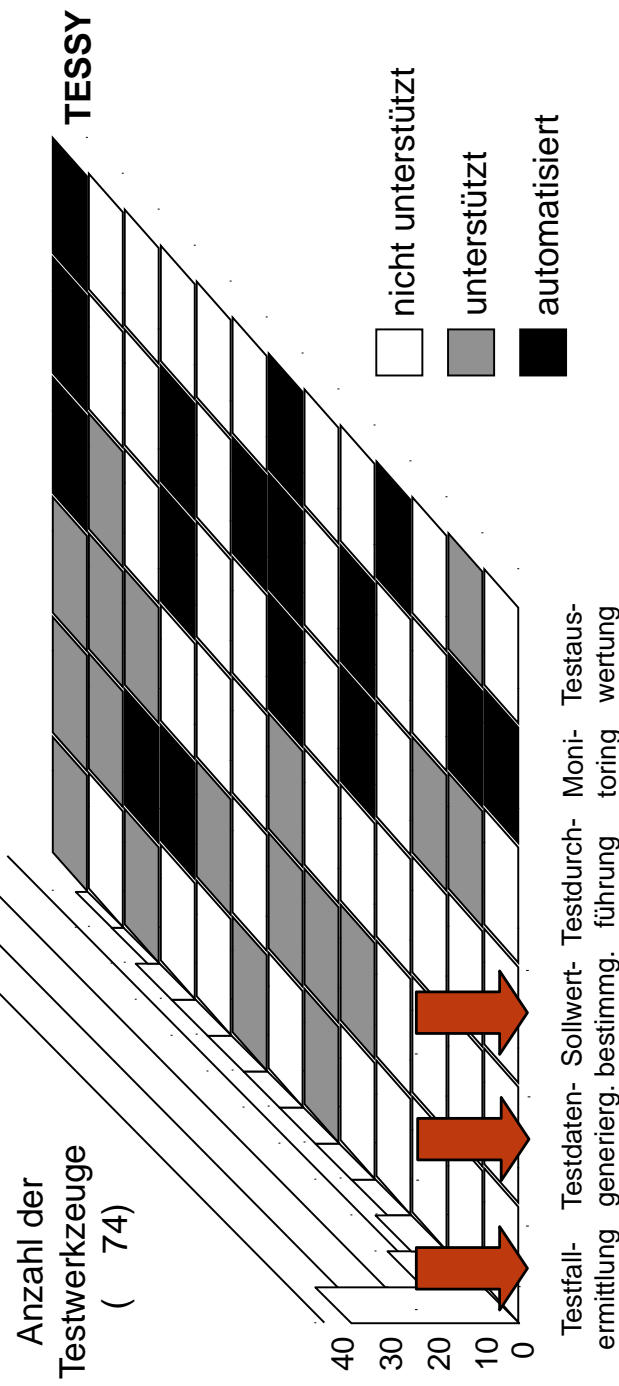
Prof. U. Asmann, Softwareentwicklungswerkzeuge (SEW)



Aufgaben und Merkmale:

- ▶ Testfallermittlung auf der Grundlage einer **Kombination von funktions- und strukturorientierten Testverfahren** sowie der **Klassifikationsbaum-Methode**
- ▶ Regressionstests
- ▶ Vorgehen:
 - Ausgangspunkt **funktionsorientierte** Testfälle
 - **strukturorientierte** Testfallermittlung nach der Klassifikationsbaum-Methode
 - Bestimmung der **Programmüberdeckung** nach Auswertung der Durchlaufhäufigkeiten
 - **Wiederholung** funktionsorientierter(1) oder strukturorientierter(2) Testfälle bis **maximale** Überdeckung der Programmzweige erreicht.

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 Verifikation von Software; GMD-Bericht Nr. 260, R. Oldenbourg Verlag 1996

Editor CTE	Vervollständigung und Überwachung des Klassifikationsbaumes → systematische Def. von funktionsorient. Testfällen → Erstellen Klassif.baum für aktuelles Testobjekt → Generierung von Testfällen
Environment-Editor	Organisation testvorbereitender Festlegungen zur Testumgebung 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 definierten Testfall des Testobjektes Browserfenster dienen getrennter Eingabe der Daten und übersichtlicher parametergesteuerter Auswahl der Testbedingungen
Monitoring EXP (execution panel)	Nach Auswahl des Testfalls generieren des Testtreibers → Testdurchführung mit Messung der Zweigüberdeckung → Registrierung der Ergebnisse in Echtzeit → Protokollierung → Herstellung Ausgangszustand
Testauswertung EVP (evaluation panel)	Generieren der Testdokumentation unterschiedlicher Granularität → Aufbereitung zur Weiterverarbeitung in speziellen Dokumentationswerkzeugen

71.2.2 Coverage Tools – Werkzeuge zur Pfadabdeckung

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

Steuerflussorientierter Test (code coverage)

Überdeckungseinheit	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
Bedingungskombination	Möglichst viele Kombinationen mehrerer Bedingungen werden getestet. Abdeckung azyklischer Pfade durch das Programm	Problem: kombinatorische Explosion
eingeschränkte Bedingungskombination	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 höchstens einmaligem Durchlauf durch eine Schleife	Begrenzung auf $k \leq 1$
Interior-Test	Abdeckung aller Pfade bei höchstens zweimaligem Durchlauf durch eine Schleife	$k \leq 2$

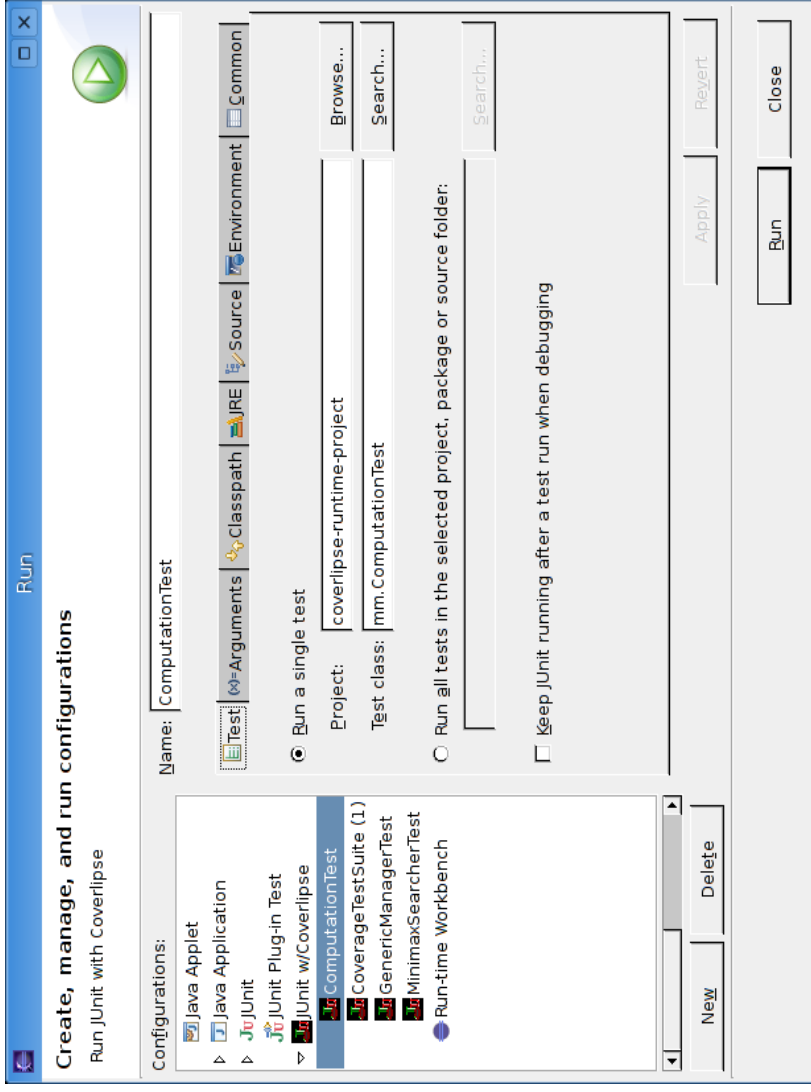
Datenflussorientierter Test (data flow coverage)

Überdeckungseinheit	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

27

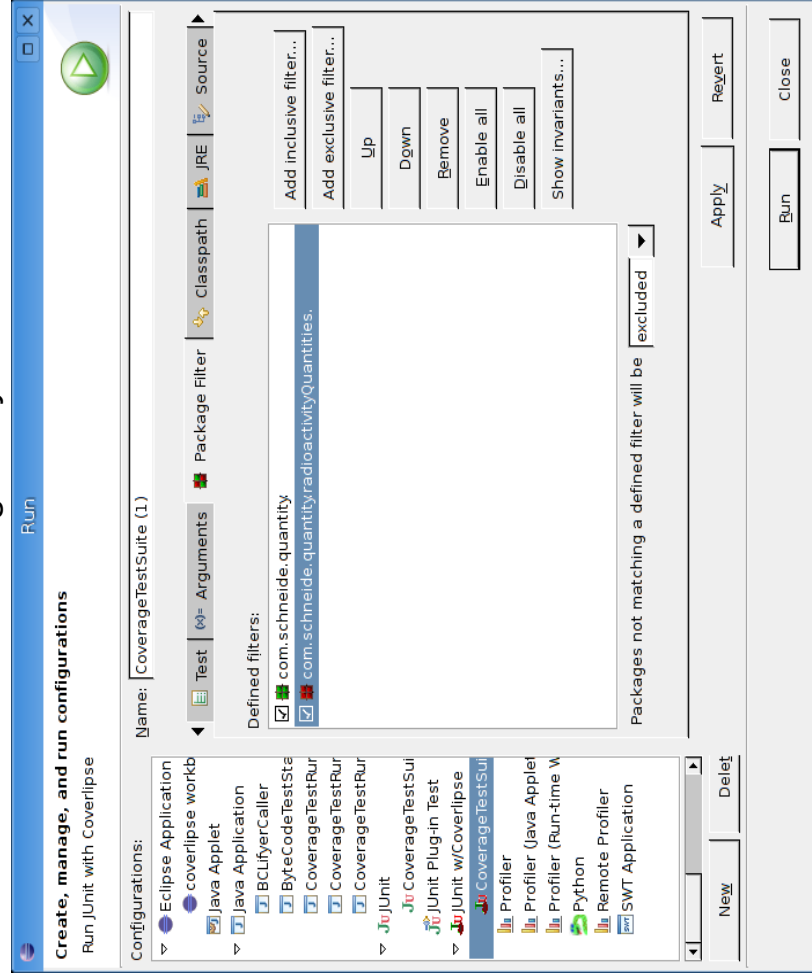
- ▶ Selektion von JUnit-Testfällen und deren Pfadabdeckungsanalyse



Coverage Tools – Werkzeuge zur Pfadabdeckung

28

- ▶ Paketfilterung stellt die Pakete zur Pfadabdeckungsanalyse ein



Coverlipse

29

- ▶ block coverage / statement coverage

```
24e public IVariable getVariable() {
25     return this.variable;
26 }
27e public boolean equals(Object obj) {
28     return true;
29 }
30
31 if (obj instanceof BlockVarCombinator) {
32     BlockVarCombinator comb = (BlockVarCombinator) obj;
33     return getVariable().equals(comb.getVariable());
34 }
35
36 return false;
37 }
38e public int hashCode() {
39     return getBlock().hashCode() + getVariable().hashCode();
40 }
41
42 }
43 }
```

Line	covered	uncovered
25	25	0
28	28	0
29	0	29
31	31	0
32	32	0



Coverlipse

30

- ▶ All-uses-coverage

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

```

Message	Line	covered	uncovered
Definition of the variable arg1	12	13	15
Definition of the variable arg2	12	13	0
Definition of the variable meinInt	13	19	0
Definition of the variable result2	14	18	16
Definition of the variable result3	18	18	0



Coverlipse

31

- ▶ Problembeschreibung aus einem Use

java - Computation.java - Eclipse Platform

File Edit Source Refactor Navigate Search Project Run Window Help

Coverlip... 3
All-Uses Coverage
Clear results
simpletests
Computation

```
public class Computation {  
    public int add(int arg1, int arg2) {  
        int result = arg1 + arg2; int meintInt = 0;  
        Show all definitions  
        Show uses of the variable meintInt  
        Show uses of the variable result  
    }  
}
```

Problem description: Definition of the variable meintInt

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 meintInt	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

Writable Smart Insert 13 : 1



Coverlipse

32

- ▶ all-uses-coverage information

java - Computation.java - Eclipse Platform

File Edit Source Refactor Navigate Search Project Run Window Help

Coverlip... 3
All-Uses Coverage
Clear results
simpletests
Computation

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

Problem description: Definition of the variable meintInt

Message	Line	covered uses	uncovered uses	Resource
Definition of the variable meintInt	13	14 19	16	Computation.java
Definition of the variable result	13	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

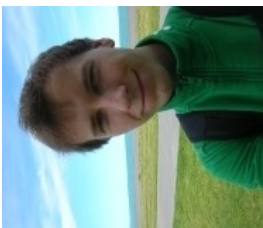
Writable Smart Insert 14 : 1



71.3 Testautomatisierung mit Test-Frameworks

33

- JouleUnit (courtesy Claas Wilke)
YouTube-Video zum Thema: <http://is.gd/energyLabel>
Mehr Infos zum Projekt:
<http://www.qualitytone.org/>
<http://www.jouleunit.org/>
claas.wilke@tu-dresden.de



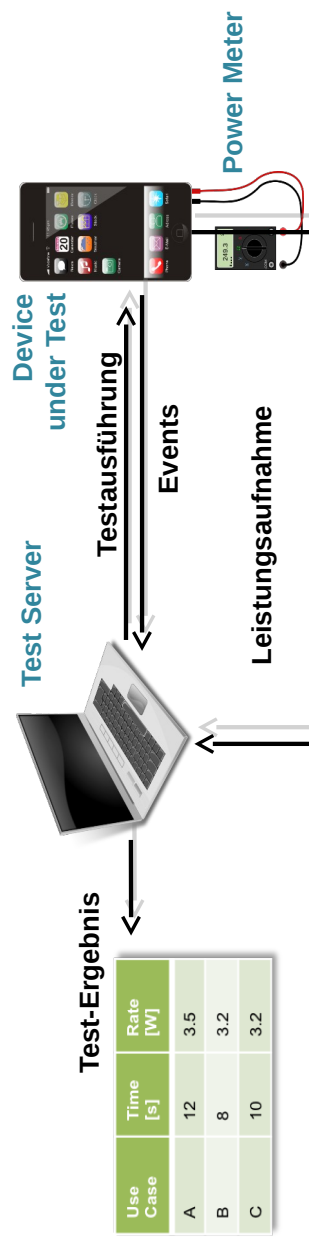
Softwareentwicklungswerkzeuge (SEW) © Prof. Uwe Alßmann

EnergieTest mit JouleUnit

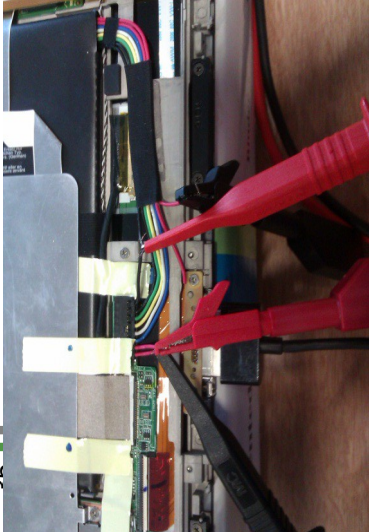
- ▶ Generisches, wiederverwendbares profiling framework [WGR13]
 - Basiert auf Unit-Tests: Testfälle definieren Workloads
- ▶ Wiederverwendbar für verschiedene Geräte (z.B., Android, NAOs)

34

Prof. U. Alßmann, Softwareentwicklungswerkzeuge (SEW)



Energie-Test



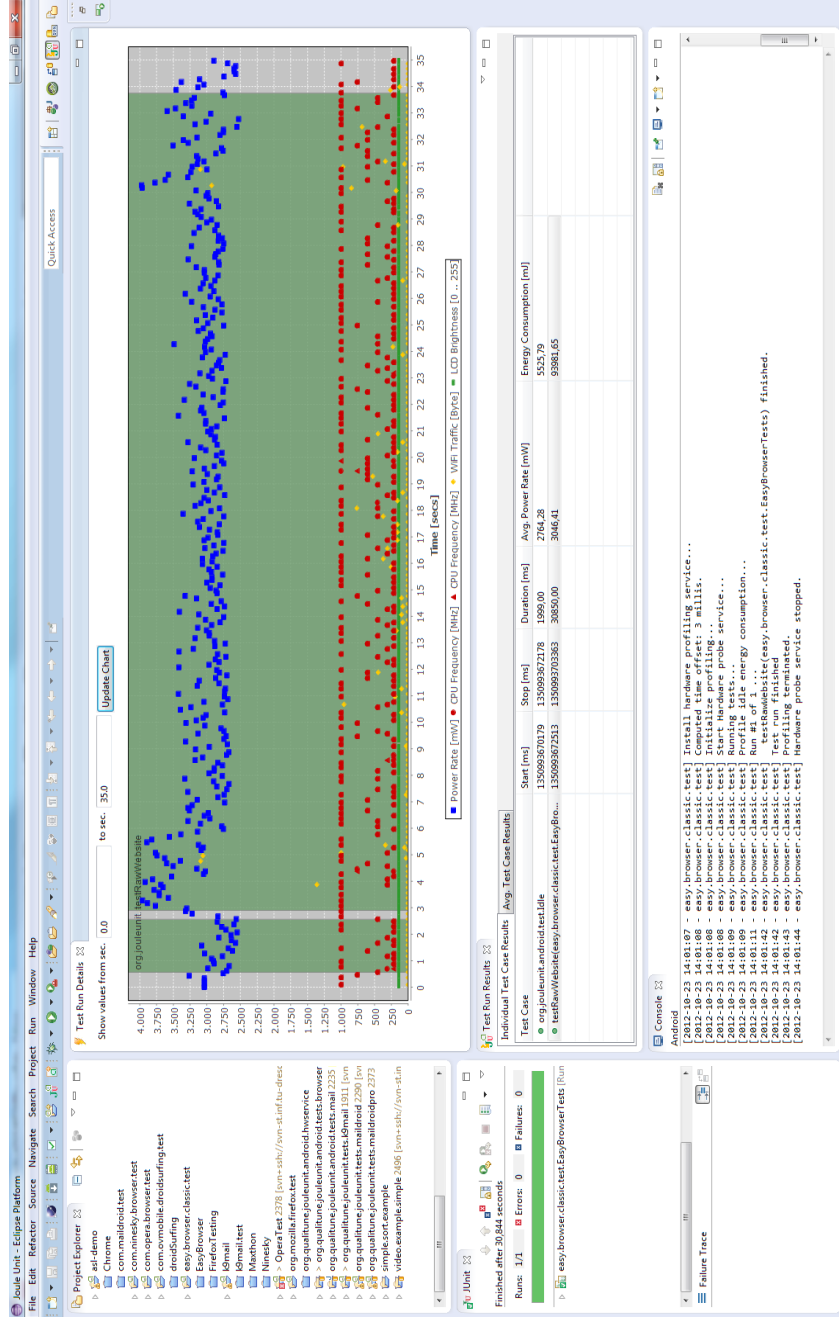
JouleUnit + QMark

- ▶ JUnit-Erweiterung für Energie-Tests von Android-Apps
- ▶ Wiederverwendung von JUnit Tests
 - Wiederverwendung funktionaler Tests
 - Kein wesentlicher Mehraufwand
- ▶ Ausführung lokal auf eigenem Smart Phone
 - Testentwicklung
 - Grobes Verbrauchsfeedback
- ▶ Ausführung remote auf QMark Test-Server
 - Genaue Messungen der Leistungsaufnahme



JouleUnit Workbench (Eclipse)

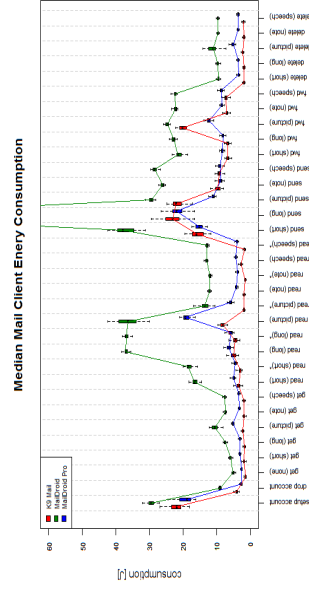
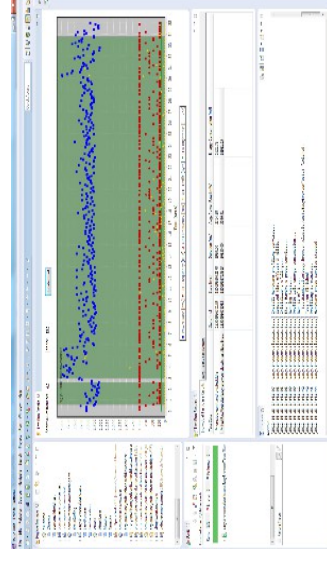
37



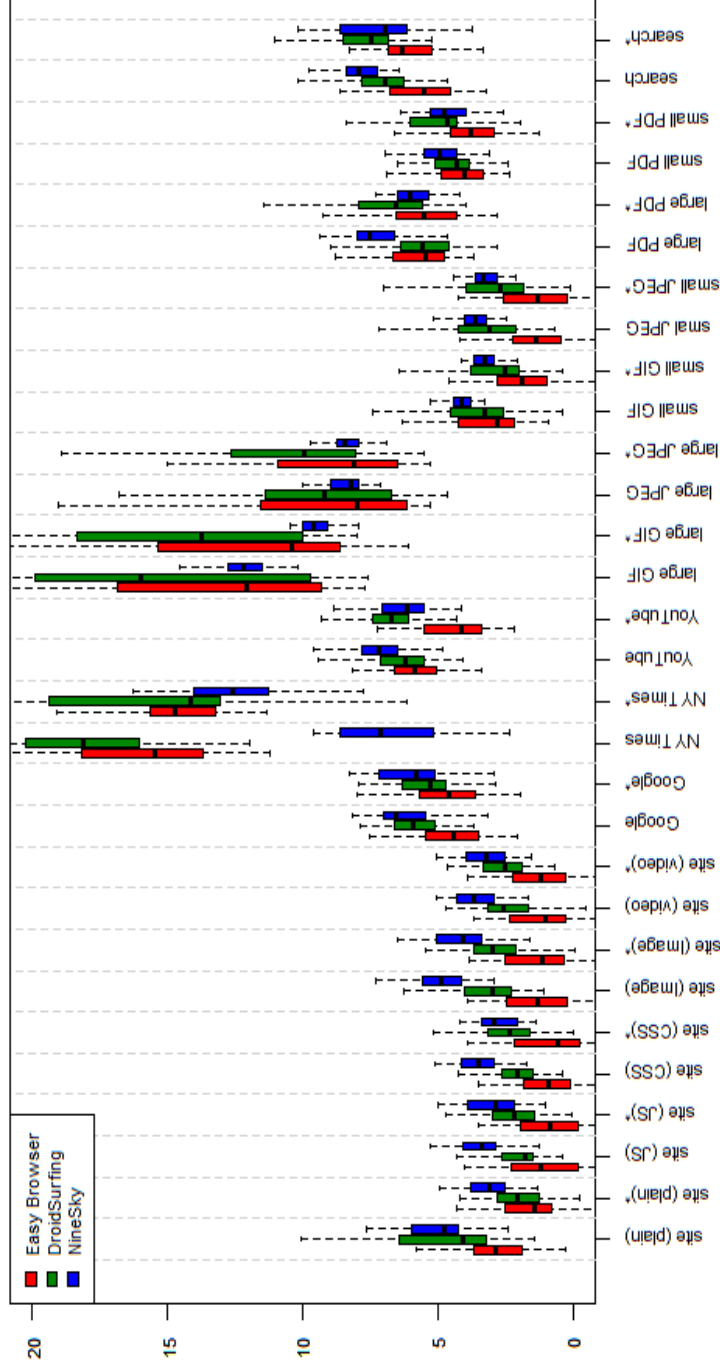
„Ähnliche“ Anwendungen vergleichen

38

- ▶ Definition von Benchmarks
 - Web browsing
 - Emailing
- ▶ Instanziierung für bestehende Anwendungen
 - EasyBrowser, DroidSurfing, NineSky
 - K9 Mail, MailDroid, MailDroid Pro
- ▶ Profiling 50 mal pro Anwendung

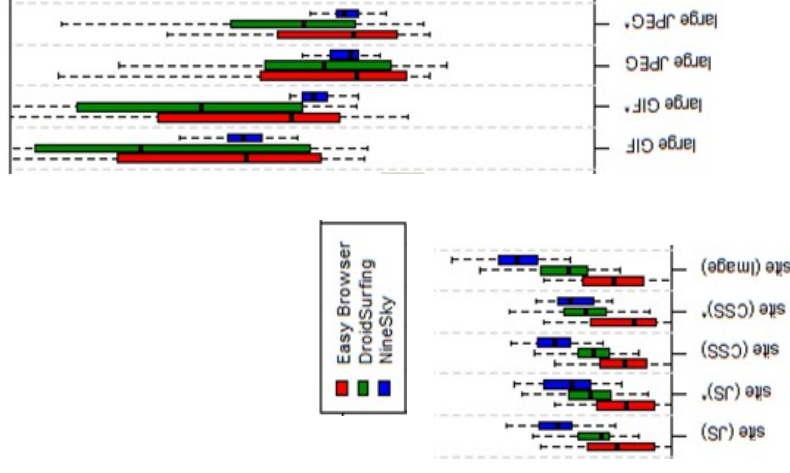


Median Web Browser Energy Consumption

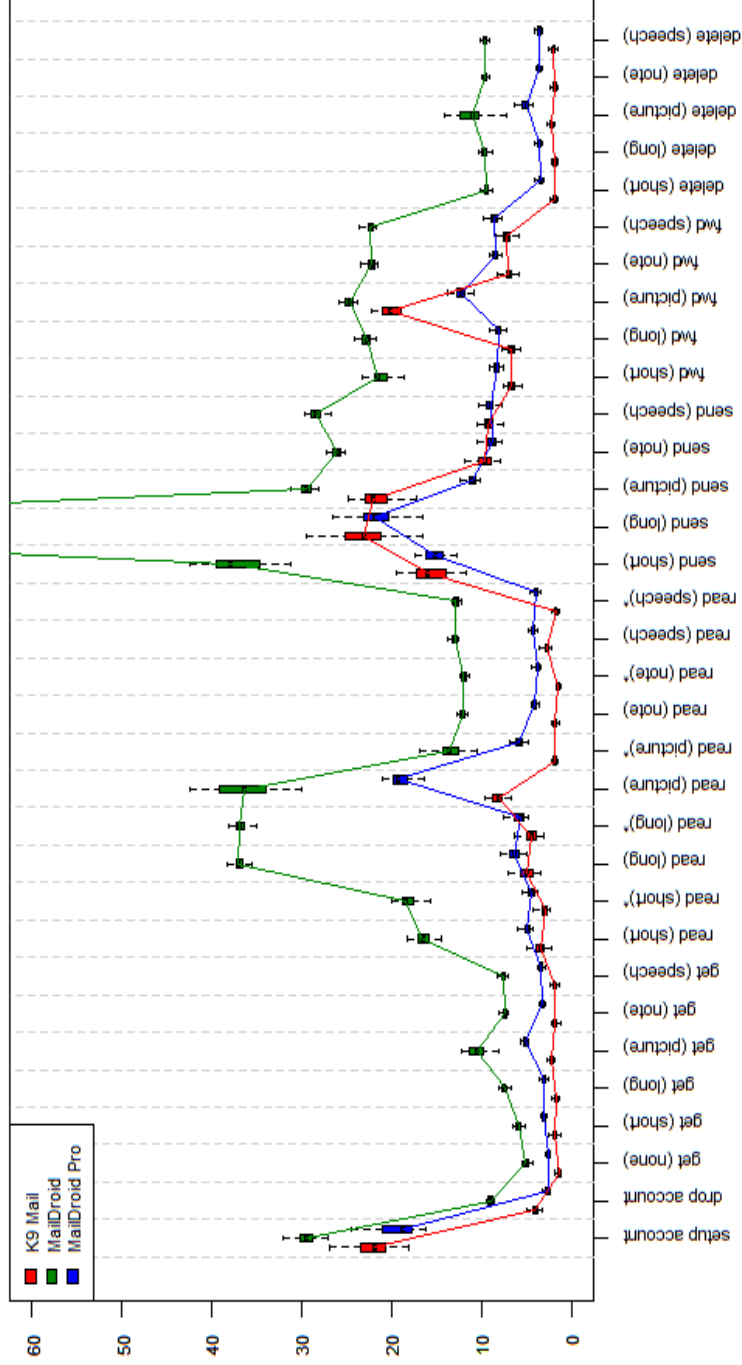


Vergleich von Web Browsern

- ▲ Hohe Varianz in Messungen (durch hohe Varianz in Antwortzeiten)
 - Aber: vergleichbare Trends
- ▲ NineSky schlecht für kleinere Seiten
 - Aber: besser für große Bilder (da schneller)
- ▲ Advertisement in EasyBrowser, DroidSurfing negative Auswirkung nur bei langen Ladezeiten
- ▲ Verschiedene Browser optimal für verschiedene Anwendungsfälle

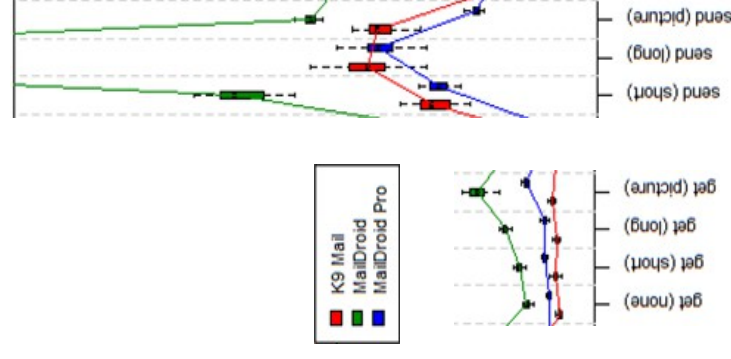


Median Mail Client Energy Consumption



Vergleich von Email Clients

- ▲ Geringe Varianz in Messungen
 - Vergleichbare Trends ausmachbar
- ▲ MailDroid schlechter für alle Szenarien
 - Grund: Advertisement
 - Negativer Einfluss steigt für lange Szenarien
 - MailDroid pro & K9 Mail verhalten sich ähnlich
- ▲ Unterschiede sich im Energieverbrauch
 - Advertisement ist zu vermeiden



71.3.2 Modellgetriebenes Testen mit dem Werkzeug MATE

43

- Georg.Pueschel@tu-dresden.de



Softwareentwicklungswerkzeuge (SEW) © Prof. Uwe Alßmann

Modellgetriebenes Testen mit



44

- Modellgetriebenes Testen (MBT):** Black-Box-Testfälle werden aus Modellen generiert, z.B. aus State Charts, Petri Netzen, Aktivitätsdiagrammen, Sequenzdiagrammen

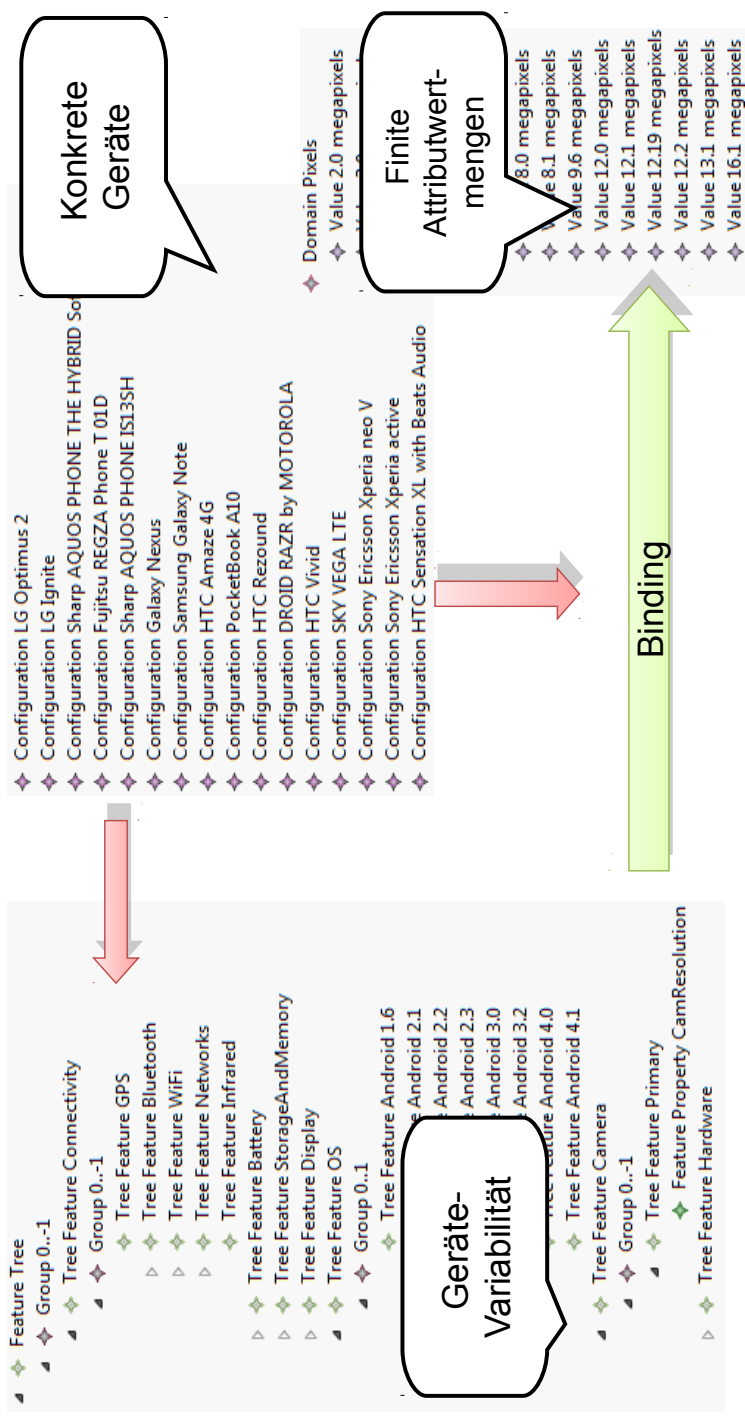
Prof. U. Alßmann, Softwareentwicklungswerkzeuge (SEW)

The screenshot shows the Google Play Store interface. Three callout boxes highlight specific issues:

- Problem 1:** Große Anzahl verschiedener Plattformen (Large number of different platforms)
- Problem 2:** Apps unterscheiden sich je nach Gerät (Apps differ by device)
- Problem 3:** App sind in Zukunft Kontext-~~ad~~aptiv (Apps will be context-adaptive in the future)



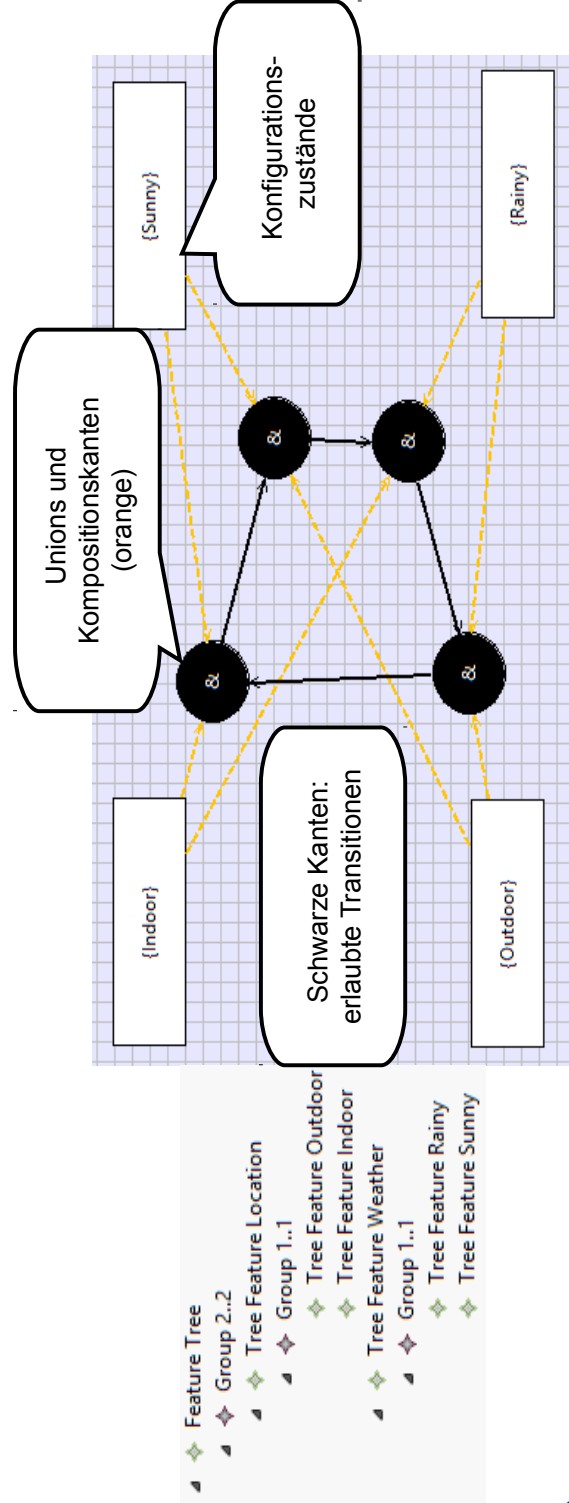
Plattformmanagement durch attributierte Feature-Modelle



Variabilität in Applikationen kann ebenfalls durch Features beschrieben und Requires-/Excludes-Kanten logisch mit dem Plattformmodell verknüpft werden.

Laufzeit-dynamische Features

- ▶ Durch (De-)Aktivieren von Features können Laufzeit-Rekonfigurationen beschrieben werden.
- ▶ Dadurch können wechselnde Kontexteigenschaften beschrieben werden.
- ▶ Erlaubte Rekonfigurationen beschreibt ein Operational Configuration Model



Adaptionbeschreibung

47 Prof. U. Asmann, Softwareentwicklungswerkzeuge (SEW)

Transitions werden durch Bedingungen über Kontextdatenwerte und aktiver Features gesteuert

(Simplified) Timed Petri Net steuert Kontextänderungen in Daten-Szenarien oder OCM (OCM-Szenarien haben eine Übergangszeit)

OCM-Alternative: Szenarien beschreiben Wertänderungen von Kontextdaten

Inputscen: Inputscen Properties Problems
Scenario ID: scen0
Scenario name: scen0
Scenario length: 100
Resources:
Data leaves:
struct11
struct12
struct14

Time	struct11	struct12	struct14
0	0	0	0
1	1	0	0
2	2	0	0
3	3	0	0
4	4	0	0
5	5	0	0
6	6	0	0
7	7	0	0
8	8	0	0
9	9	0	0
10	10	0	0
11	11	0	0
12	12	0	0
13	13	0	0
14	14	0	0
15	15	0	0
16	16	0	0
17	17	0	0
18	18	0	0
19	19	0	0
20	20	0	0
21	21	0	0
22	22	0	0
23	23	0	0
24	24	0	0
25	25	0	0
26	26	0	0
27	27	0	0
28	28	0	0
29	29	0	0
30	30	0	0

Im Ergebnis erzeugt ein Generator aus durch Erreichbarkeitsanalyse Testfälle.

71.4 Funktionalität und Werkzeuge ausgewählter Test-Umgebungen

Funktionalität von LOGISCOPE

49

- ▶ Ursprüngl. Hersteller: Telelogic North America Inc., Irvine, USA (Hersteller des Requirement Management Systems DOORS), jetzt IBM
 - http://publib.boulder.ibm.com/infocenter/rsdp/v1r0m0/index.jsp?topic=/com.ibm.help.download.logiscope.doc/topics/logiscope_version66.html
- ▶ Durchgängiges Werkzeug für die Phasen Entwicklung, Testung und Wartungen für
 - die Zweigüberdeckung,
 - die Grad der Testüberdeckung,
 - 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 Testszenarios,
 - graphische Auswertung der summarischen Testergebnisse,
 - automatische Erstellung der Testdokumentation.

Prof. U. Asmann, Softwareentwicklungswerkzeuge (SEW)



Werkzeuge von LOGISCOPE

50

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

Prof. U. Asmann, Softwareentwicklungswerkzeuge (SEW)



Werkzeuge von LOGISCOPE (2)

51

Test-Aktivität	Bezeichnung	Aufgabe
Test-durchführung	<i>TestChecker</i>	misst in Verbindung mit einem Debugger die Testüberdeckung in Echtzeit, zeigt im Quellprogramm nicht überdeckte Wege an, generiert Testberichte und übernimmt die Testfall-Verwaltung.
	<i>ImpactChecker</i>	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	<i>Viewer</i>	stellt sehr verschiedene textuelle und graphische Auswertungsmittel zur Verfügung. Er erzeugt Steuerfluss-Graphen, Komponenten-Ruf-Graphen, Auswertung von Metriken und visualisierte Vergleiche mit ausgewählten Parametern des Qualitätsmodells (Kivi-Graph).

Prof. U. Asmann, Softwareentwicklungswerkzeuge (SEW)



imbus TestBench

52



Prof. U. Asmann, Softwareentwicklungswerkzeuge (SEW)



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

Anforderungsverwaltung von Car Konfigurator (Version 2.1, Abnahmetest)

Anforderungsbaum:

- CarConfigurator - Version 1.1 (calliber)
 - 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: Review Complete

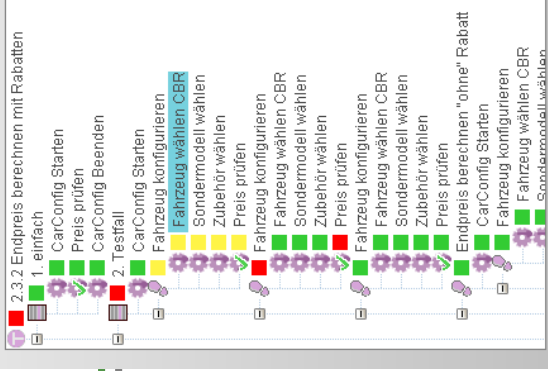
Status: Essential

Priorität: Essential

Test-Status: ■ **Getestet PASS**



Testff[...] endpreis-berechnen-mit-rabatten_log.xml



Aktuelle Ansicht : Endpreis berechnen mit Rabatten : [...]jurieren : Fahrzeug wählen CBR

Interaktion

Datum	Anzeige	Navigation	Zeitmessung	Fenster	Hilfe

Fahrzeug wählen CBR

Parameter	Wert
Fahrzeug	IS

Fehler

Fehler hinzufügen

Interaktion: Fahrzeug wählen CBR

-Beschreibung

Fahrzeug aus der Liste der Fahrzeuge wählen

-Bemerkungen zur Durchführung

Bemerkungen

-Bemerkungen zur Spezifikation

Ansicht

Details

Mehr

Benutzerdefinierte Felder der Durchführung

<für diesen Knotentyp können Benutzerdefinierte Felder nicht definiert werden>

Liste der Anforderungen

Name	ID	Version	Status	Eigentümer	Priorität
sofortige Preisberechnung	WHAT303	3.1	Accepted	Dierk	Essential
keine erzwungene Bedienerfolge	USER302	1.0	Submitted	Dierk	Essential
ständige Preisanzeige	USER301	1.0	Submitted	Dierk	Essential

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



- **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

71.4 Simulation

71.4.1 In-Vitro-Testläufe mit Debuggern

57



Softwareentwicklungswerkzeuge (SEW) © Prof. Uwe Aßmann

Entwanzer (Debugger)

58

- ▶ 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 Haltenwerte
 - Verändern derselben
- ▶ Gute Debugger funktionieren auch mit mehreren Threads, sodass Race Conditions gesucht werden können

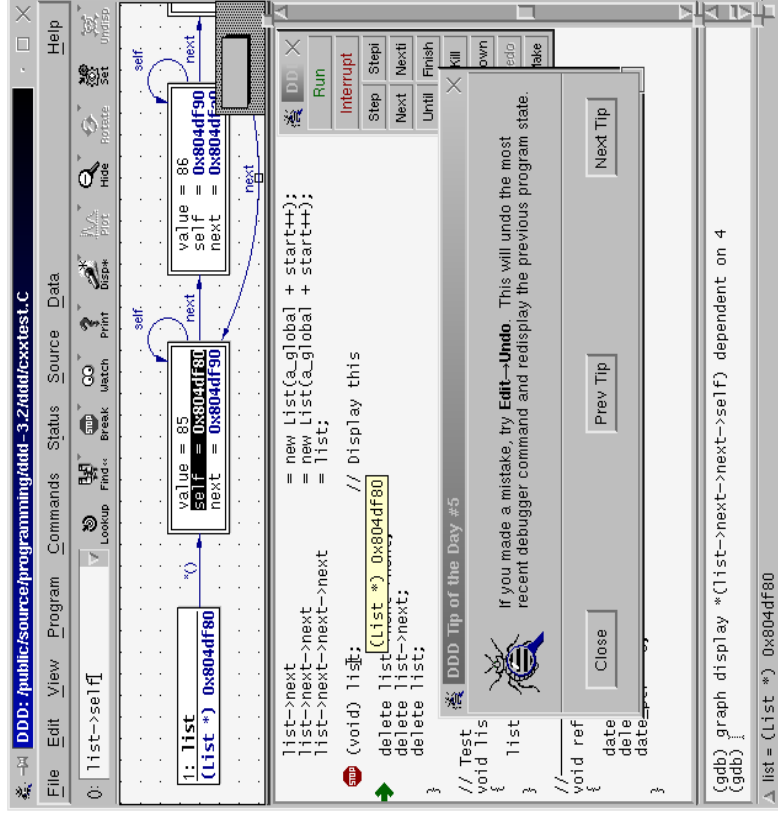


Dynamic Display Debugger (DDD)

59

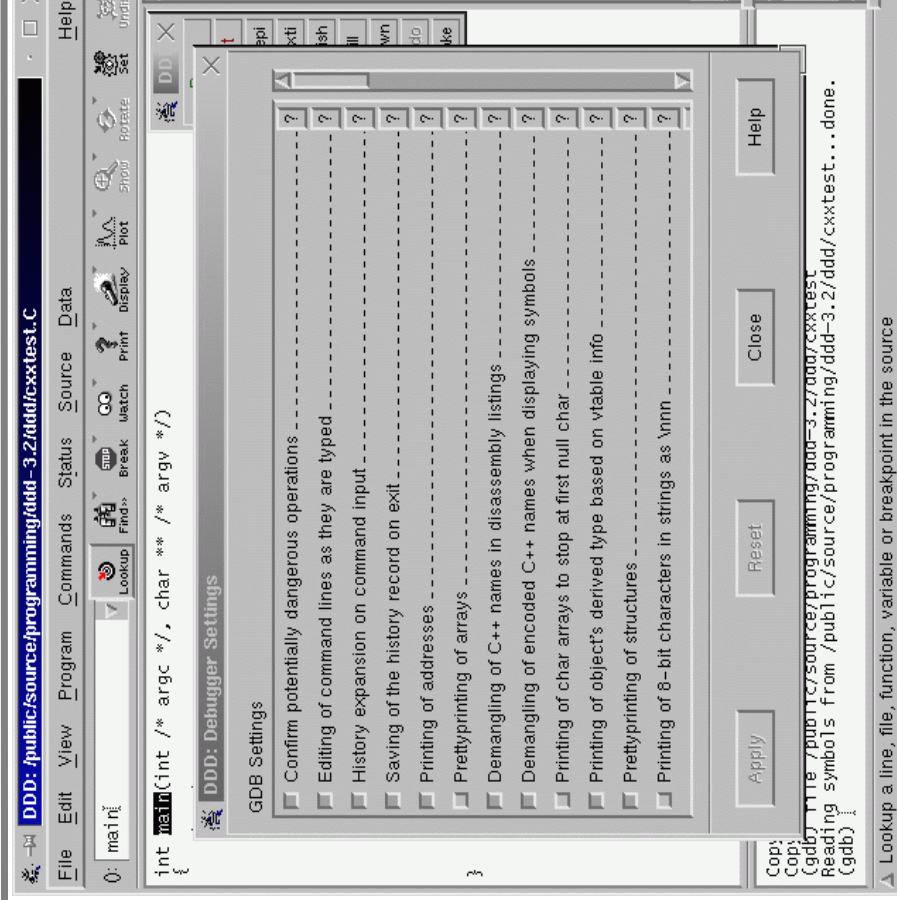
► ddd ist ein Visualisierungs-Front-end für mehrere andere Debugger

- C/C++: GDB, DBX, WDB
- Java: JDB
- Perl: Perl debugger
- bash: bashdb
- make: remake
- Python: pydb
- ddd zeigt Datenstrukturen an
 - mit Attributwerten
 - mit Verzeigerung

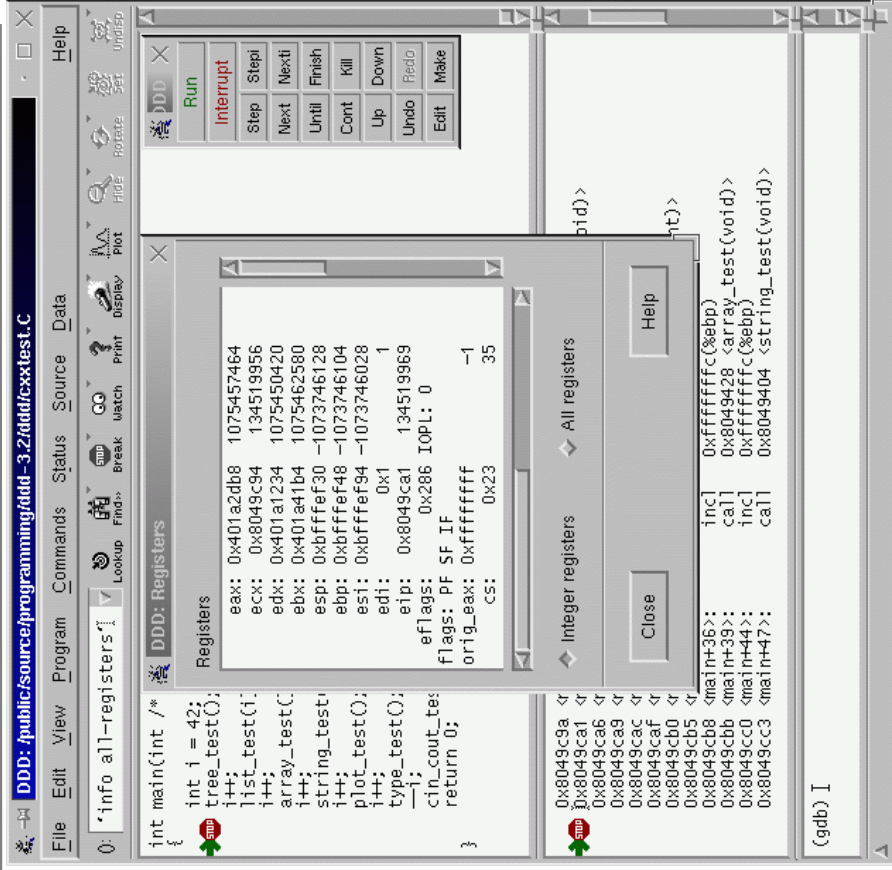


ddd Settings

60



ddd Registerwerte



Appendix

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



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!

63

Prof. U. Asmann, Softwareentwicklungswerkzeuge (SEW)

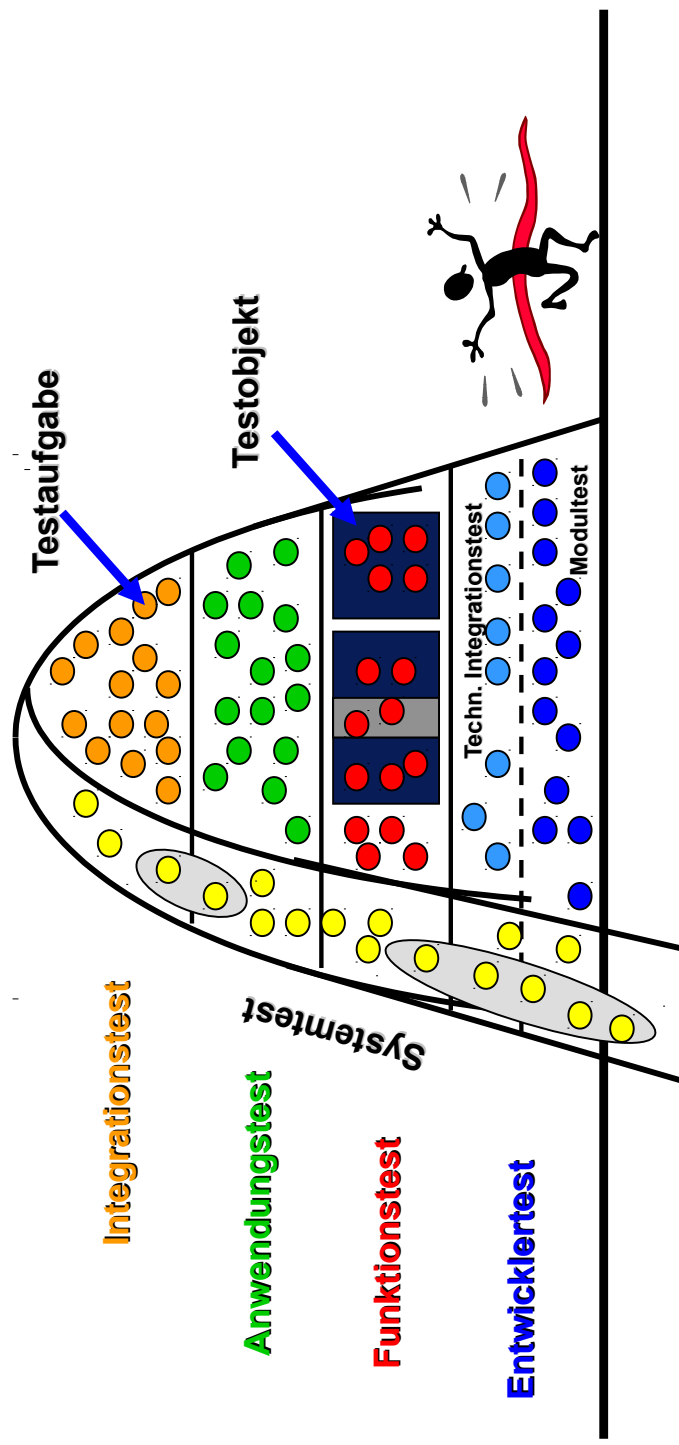


Wir brauchen zukünftig noch höhere Qualität - und das in immer kürzerer Zeit!

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



Der Testberg führt zu Angstgefühlen im Entwickler



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

65

- **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.

Prof. U. Asmann, Softwareentwicklungswerkzeuge (SEW)



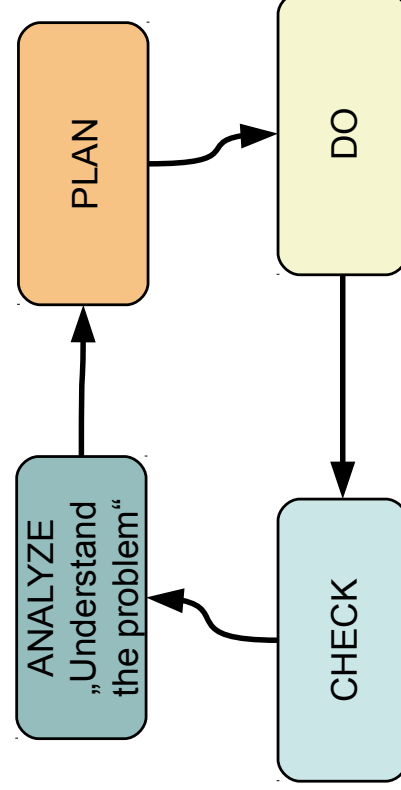
Quelle: Meyerhoff, D.B., Timpe, M., Westheide, J.T.: Teststufen und Testwerkzeuge im objektorientierten Software-Entwicklungsprozess; Softwaretechnik-Trends 18(1998) H. 2, S. 16-19

Testing zur Qualitätssicherung wendet den Feedbackzyklus "Polya Cycle" an

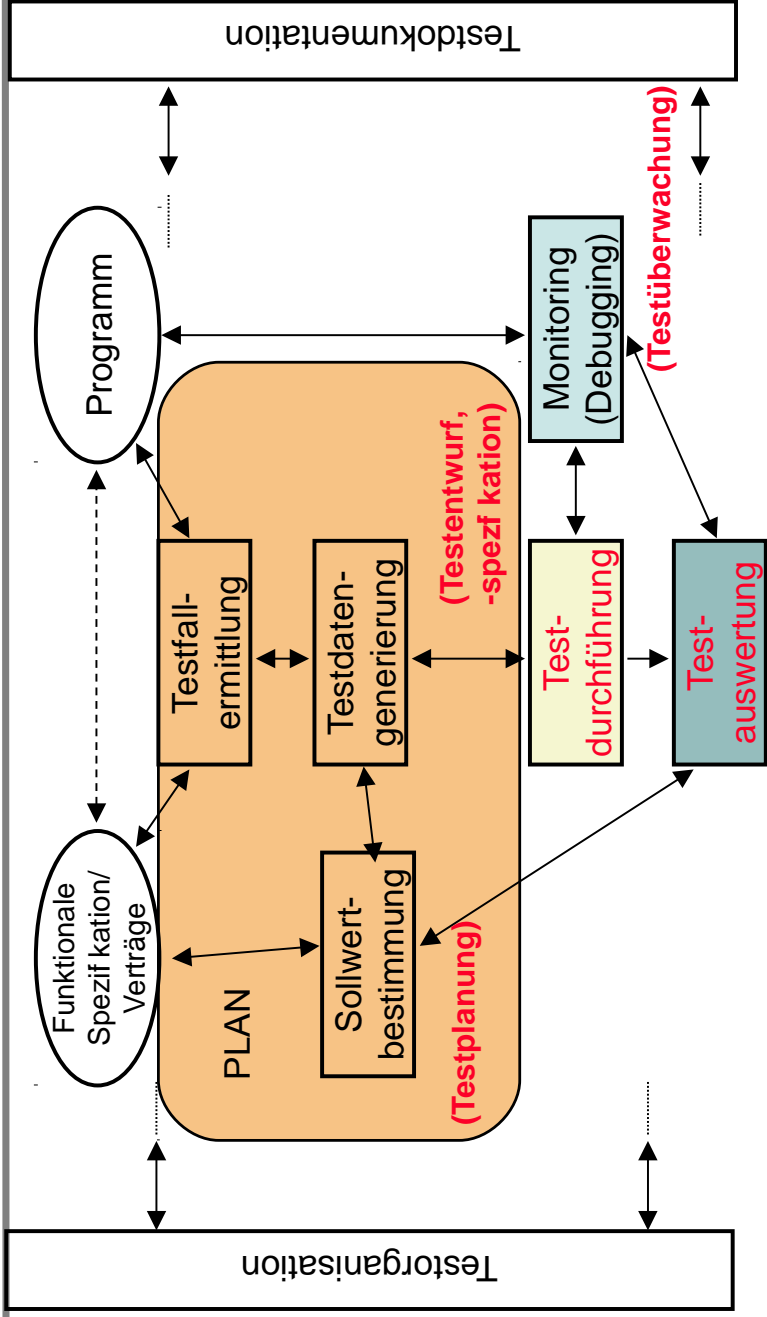
66

- ▶ Testen besteht aus Stichproben, deren Auswahl sorgfältig betrieben werden sollte
- ▶ George Polya. How to Solve It (1945).

Prof. U. Asmann, Softwareentwicklungswerkzeuge (SEW)

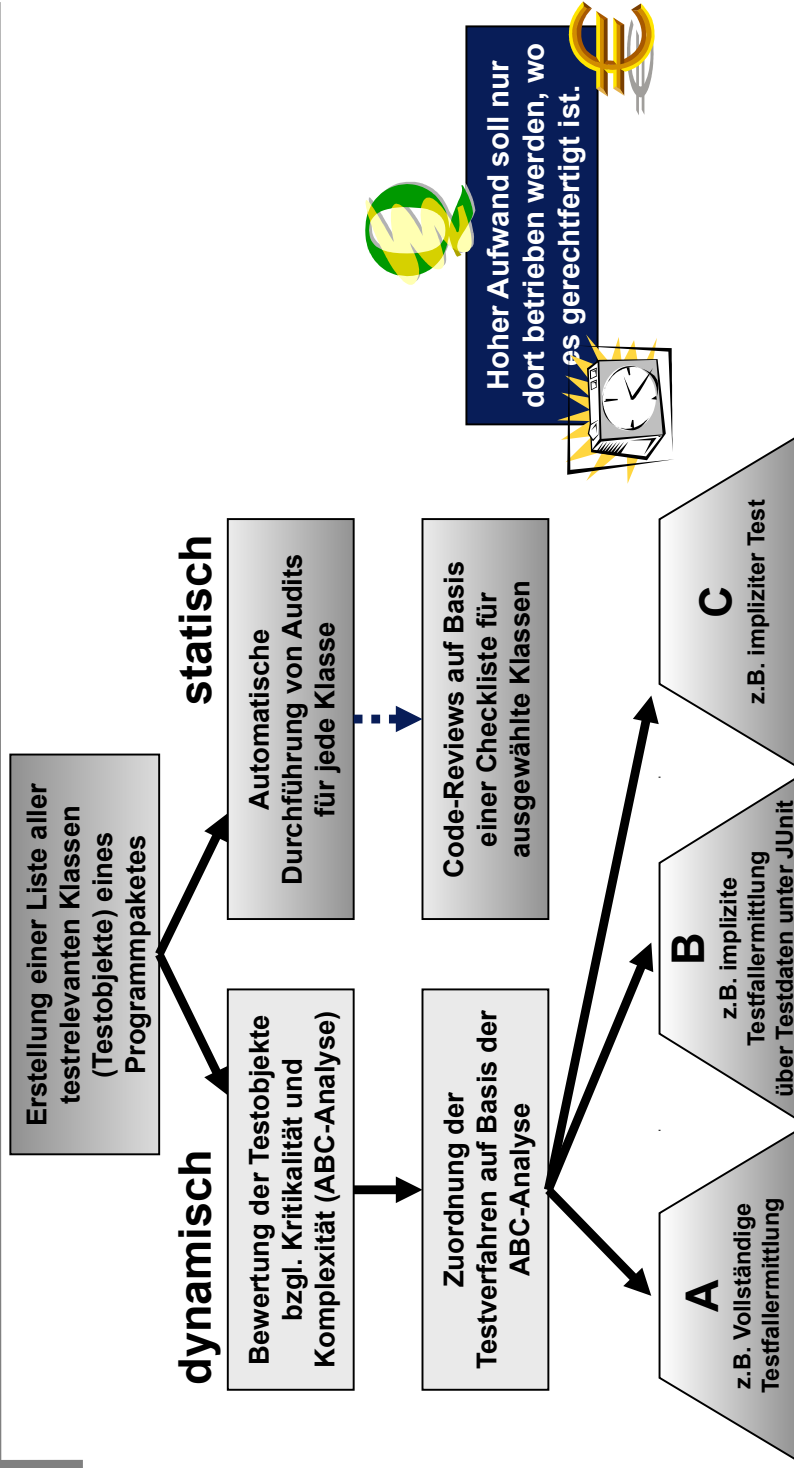


Test-Management



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

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>

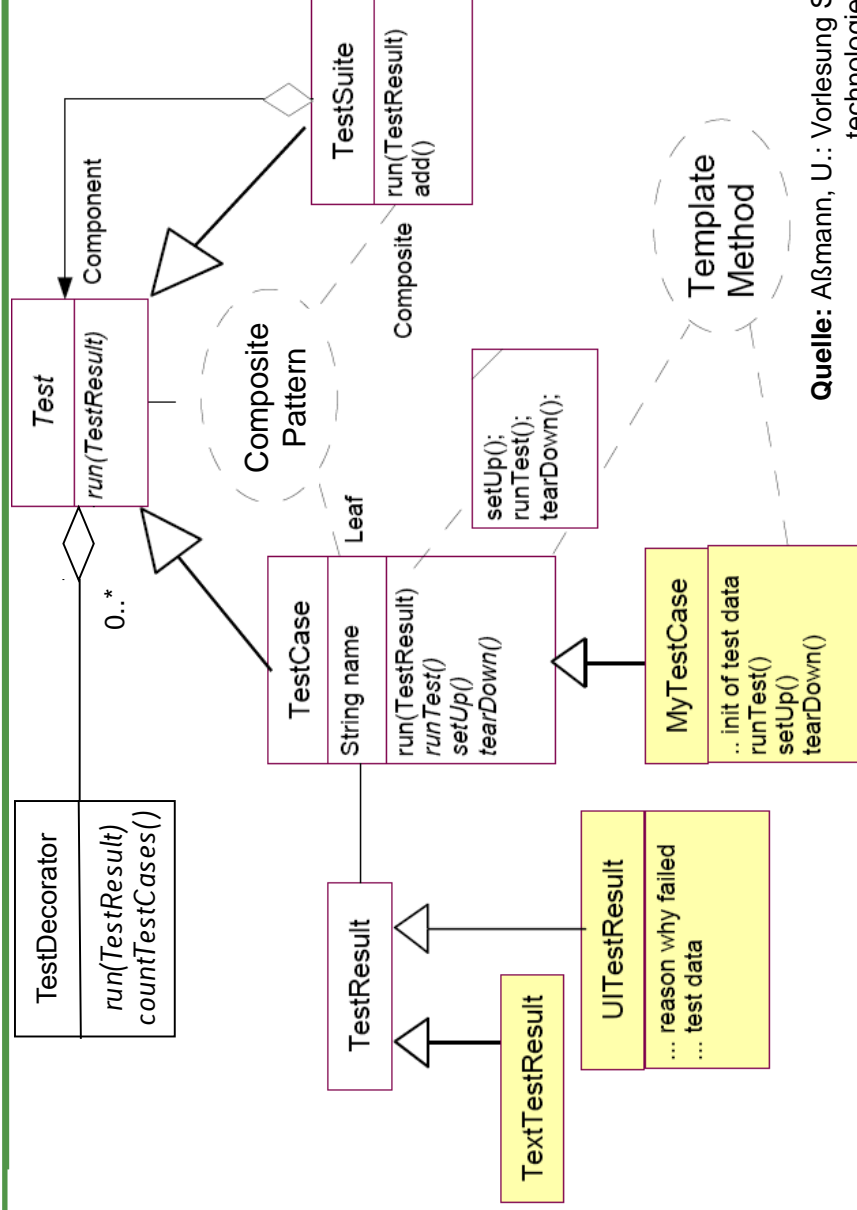
Framework JUnit für Komponententest

- **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: Alsmann, U.: Vorlesung Software-technologie II SS06



Testklassen („Werkzeuge“) von JUnit

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

71

Prof. U. Asmann, Softwareentwicklungswerkzeuge (SEW)

Lebenszyklus eines Testfalls:

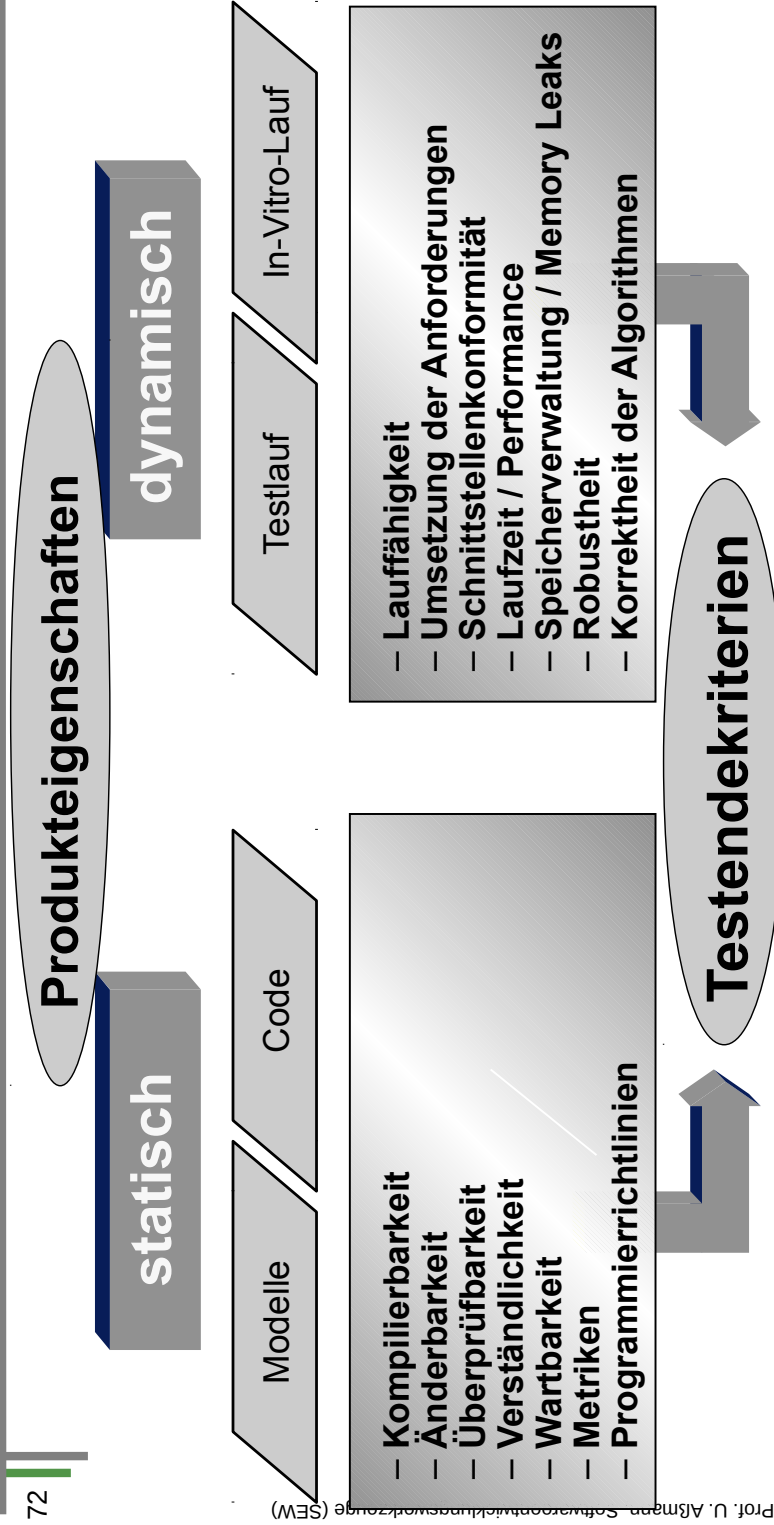
- 1. Testfallerzeugung:** Framework erzeugt für Testmethoden der zugehörigen Testklasse jeweils ein eigenes Objekt der Klasse
JUnit führt die gesammelten Testfälle voneinander isoliert aus. Reihenfolge der Ausführens der Testfälle ist undefiniert.
- 2. Testlauf:**



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

72

Entwicklertest: FURPS Qualitätsziele



Prof. U. Asmann, Softwareentwicklungswerkzeuge (SEW)

Quelle: Kugel, Thomas: Qualitätssicherung in der Praxis der Softwareerstellung; Vortrag der GI-Regionalgruppe Dresden am 18.10.2001;



Quelle: Kugel, Thomas: Qualitätssicherung in der Praxis der Softwareerstellung; Vortrag der GI-Regionalgruppe Dresden

URL: <http://www.gi-dresden.de/files/181001.pdf>

Testen - was?

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

73

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

Prof. U. Asmann, Softwareentwicklungswerkzeuge (SEW)



Testen findet im Rahmen einer SW-Entwicklungsmethode statt

74

Testmethoden:

- ▶ Anforderungsbasiertes Testen
- ▶ Geschäftsprozessbasiertes Testen
- ▶ Lebenszyklusbasiertes Testen
- ▶ Anwendungsfallbasiertes Testen
- ▶ Risikobasiertes Testen
- ▶ Spezifikationsbasiertes Testen
- ▶ Agiles Testen
 - Continuous Integration
- ▶ Exploratives Testen

Teststufen:

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

Prof. U. Asmann, Softwareentwicklungswerkzeuge (SEW)



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