

71. Testwerkzeuge

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 12-1.1, 29.01.13

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



Obligatorische Literatur

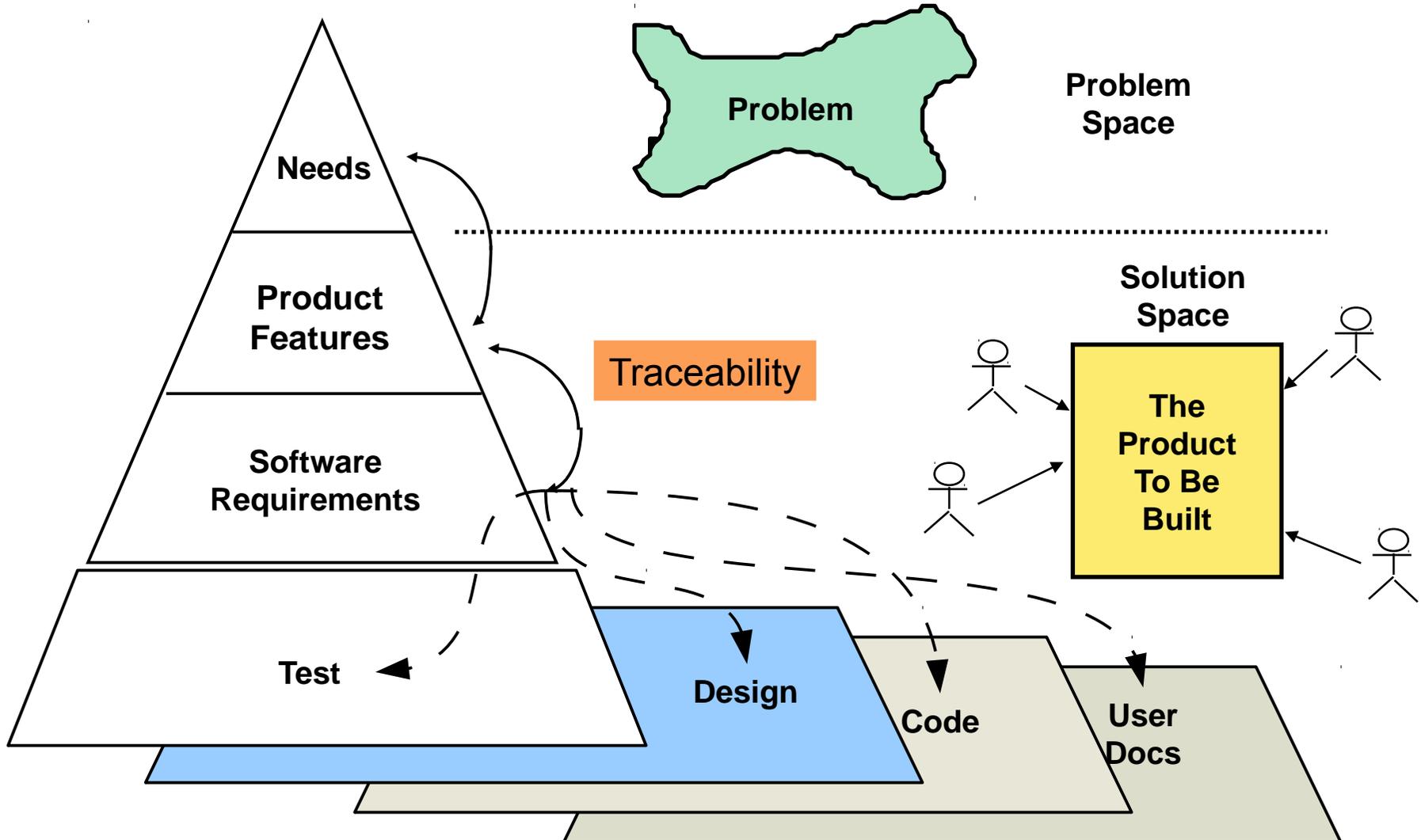
2

- ▶ 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

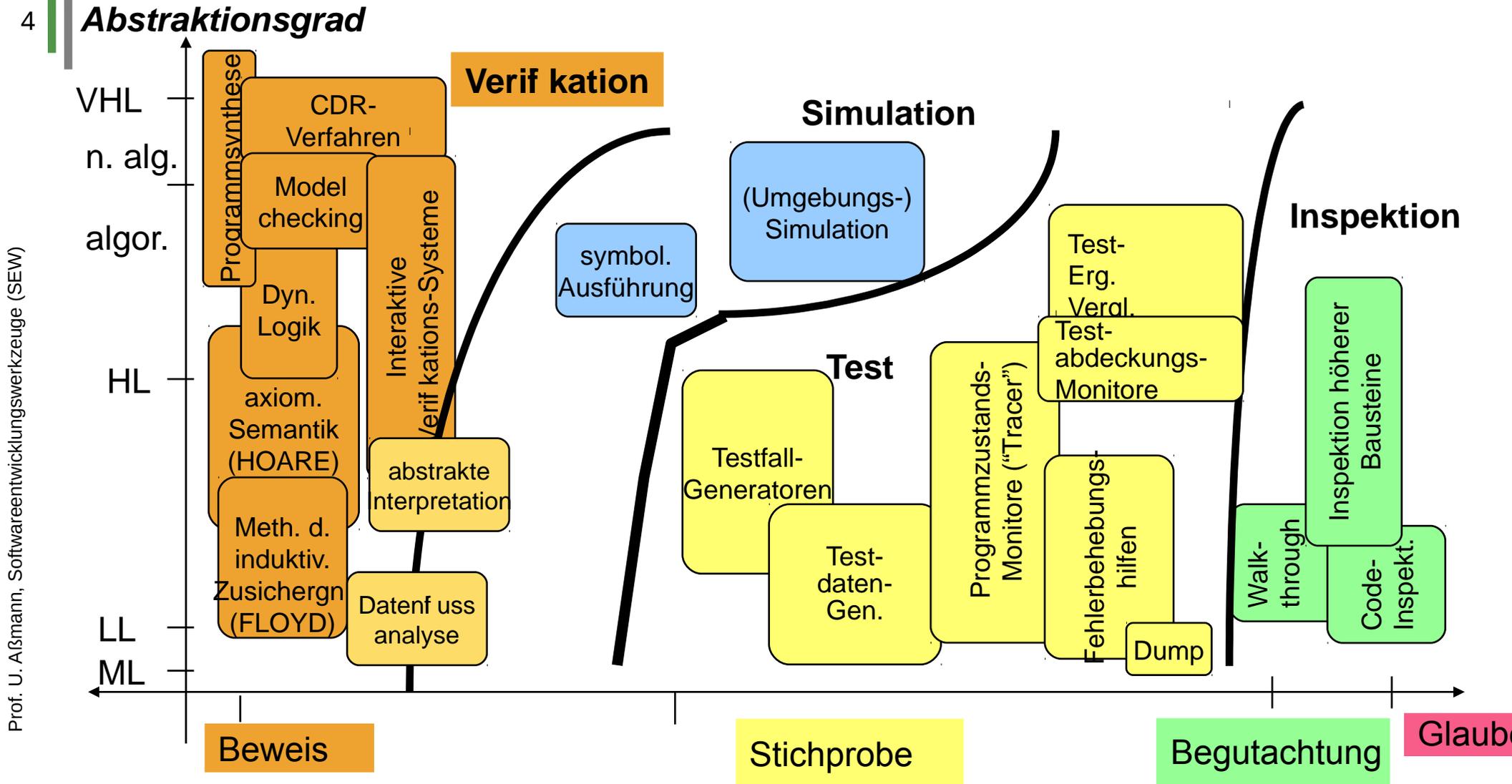
Introduction to Quality Management (QM, Quality Assurance, QA)

3

- ▶ Quality Management consists of Requirements Management and Test Management



Verifikations- und Validations-Techniken zur Qualitätssicherung (QS)



Prof. U. Alßmann, Softwareentwicklungswerkzeuge (SEW)

71.1 Aufgaben und Arten von Testwerkzeugen



5

Aufgaben von Testwerkzeugen

6

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

7

- ▶ 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

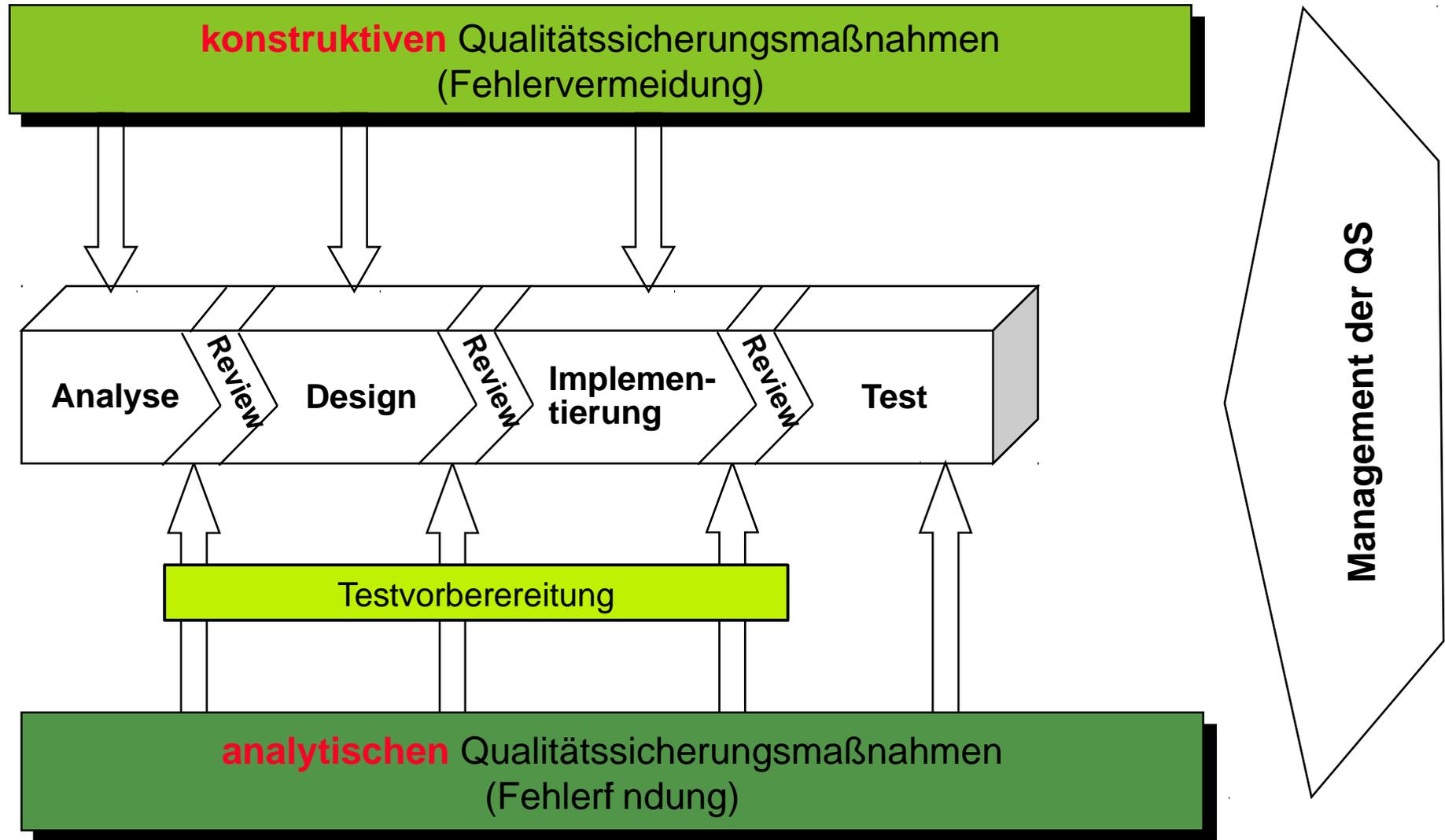
“Black-Box” Test	“Grey-Box” Test	“White-Box” Test
Funktionsabdeckung Äquivalenzklassenanalyse Grenzwertanalyse intuitive Testfallermittlung Zufallstest Fehlererwartung	“Back-to-Back”-Test Test spezieller Werte zustandsbasierter Test	Strukturabdeckung Codeüberdeckung Anweisungsüberdeckg. Zweigüberdeckung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



Kategorisierung der QS-Maßnahmen

8

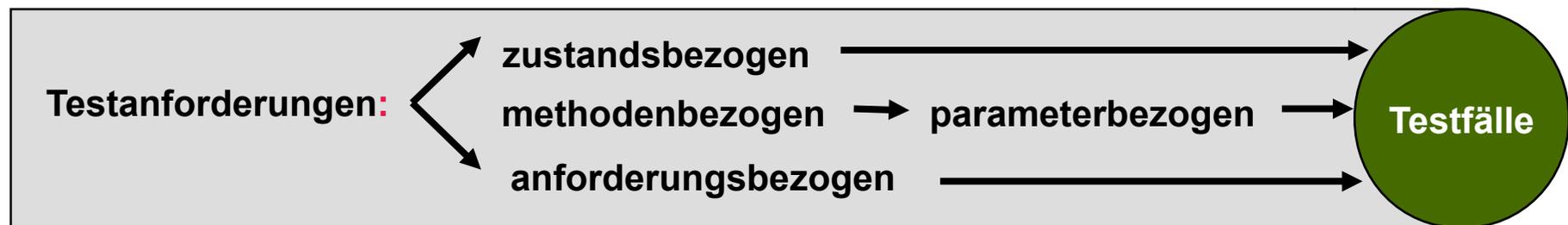
Maßnahmen der Software- Qualitätssicherung (QS) werden differenziert nach:



Der Testfall in objektorientiertem Test und das Testmodell

9

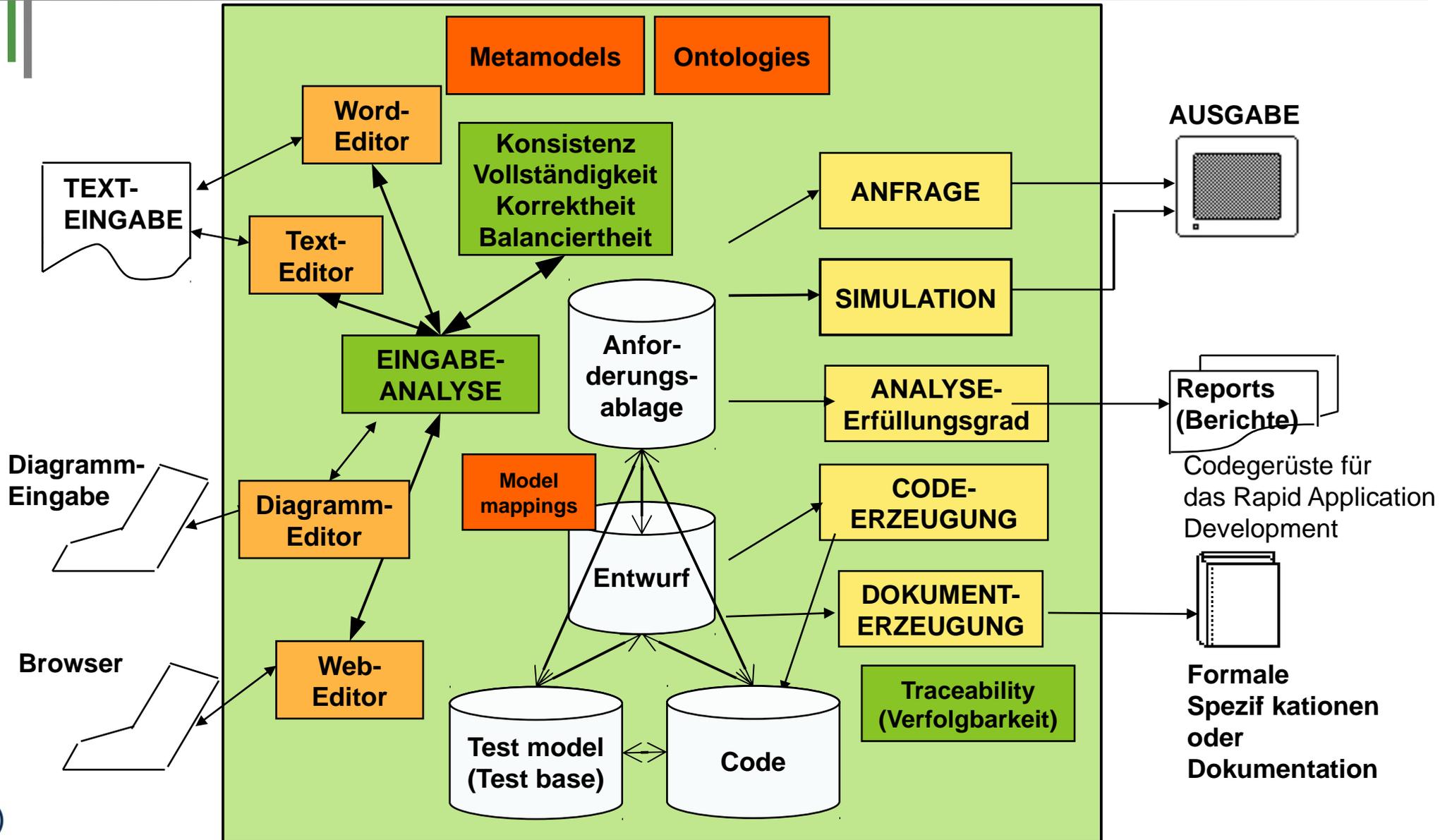
- ▶ Ein **Testfall** besteht aus:
 - ein (mehrere) Test-Objekt in einem gewünschten Ausgangszustand
 - ein (mehrere) Parameter für den Aufruf einer Methode des Objektes
 - Durch den Methodenaufruf ändert sich der Zustand des Objektes
 - einer Prüfung, ob das veränderte Objekt dem erwarteten Endzustand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



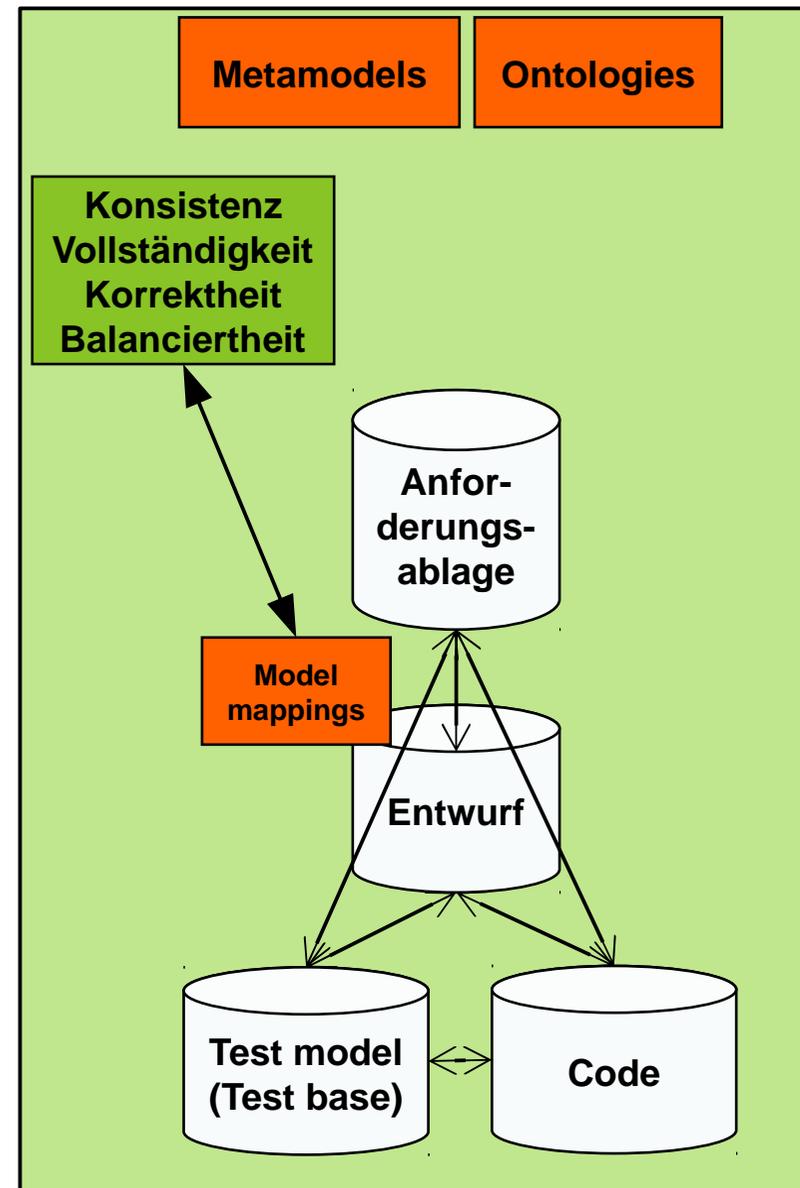
Quelle: Kugel, Thomas: Qualitätssicherung in der Praxis der Softwareerstellung; Vortrag der GI-Regionalgruppe Dresden am 18.10.2001; URL: <http://www.gi-dresden.de/files/181001.pdf>

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

10



- ▶ 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)



Anforderungen an Werkzeuge für den Entwicklertest

12

- ▶ **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

71.2 Werkzeugeinsatz in einzelnen Testaktivitäten



13

Auswahl-Liste von Test-Frameworks

14

- ▶ 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

15

	Unternehmen	URL
SilkTest	Segue Software	www.seguate.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
Tracetrone Framework	Tracetrone, Dresden. Für automotive Tests	www.tracetrone.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>



71.2.1 Die Klassifikationsbaum-Methode

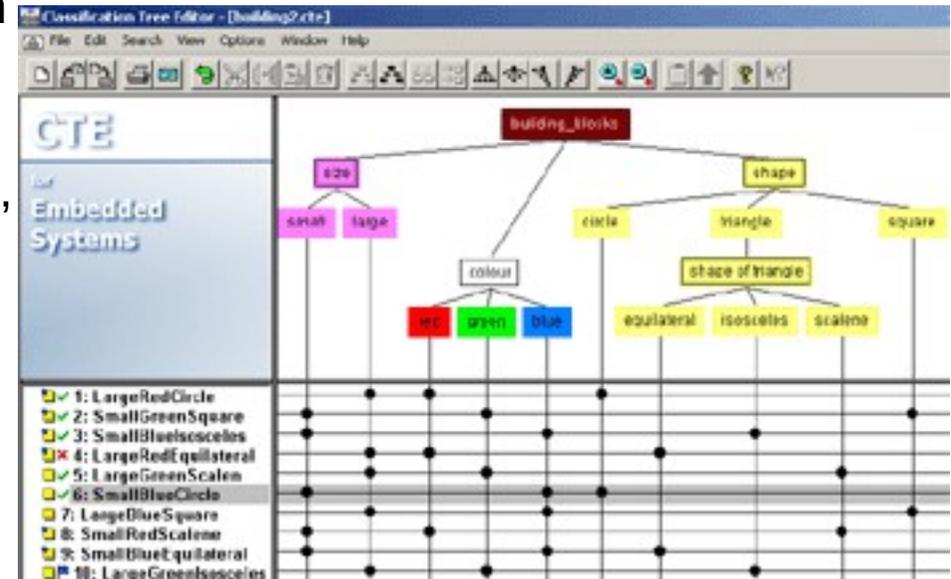
16

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

Klassifikationsbaum (Classification Tree)

17

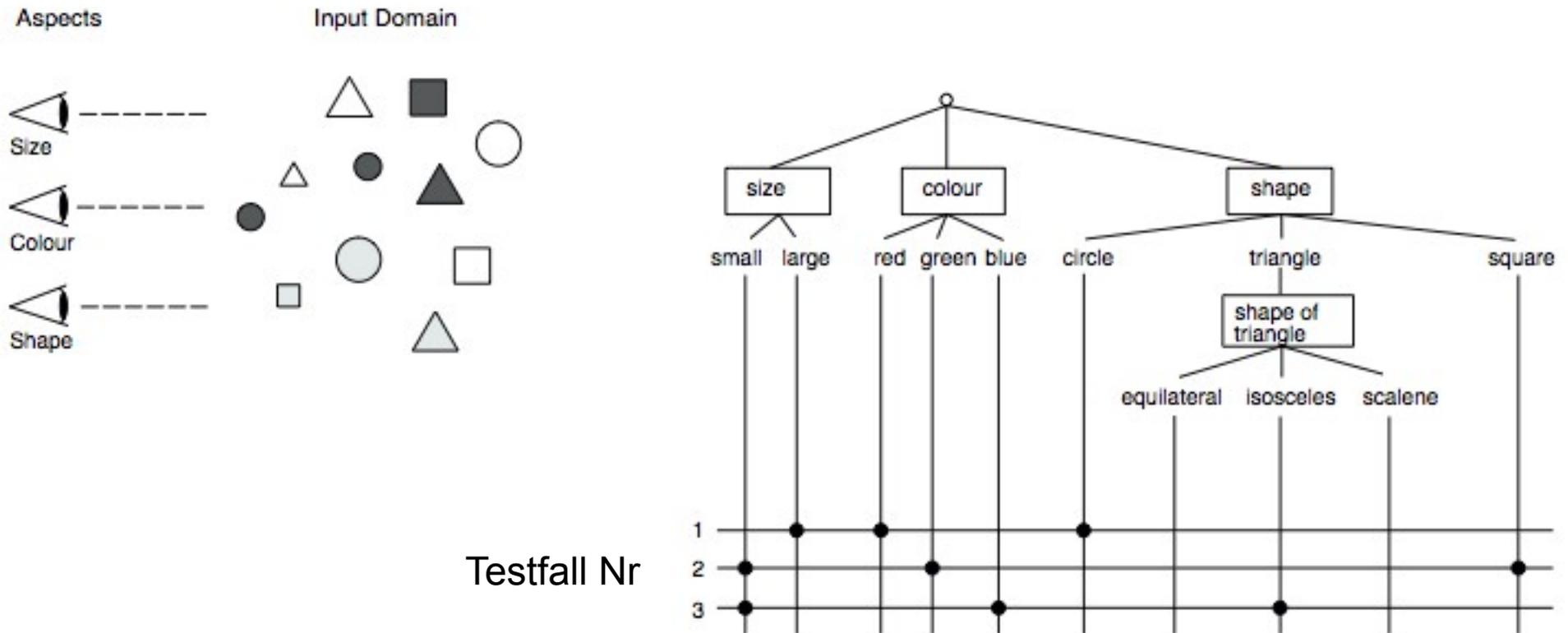
- ▶ Einteilung der Testdaten in Kategorien
- ▶ Einteilung in ein Modell basierend auf dem Classification Tree Metamodel
 - http://de.wikipedia.org/wiki/Klassifikation_sbaummethode
 - Grochtmann, M., Grimm, K.: Classification Trees For Partition testing, Software testing, Verification & Reliability, Vol. 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>



Kategorien (Facetten, Aspekte) der Testfalldaten

18

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



Vorteile der Klassifikationsbaum-Methode

19

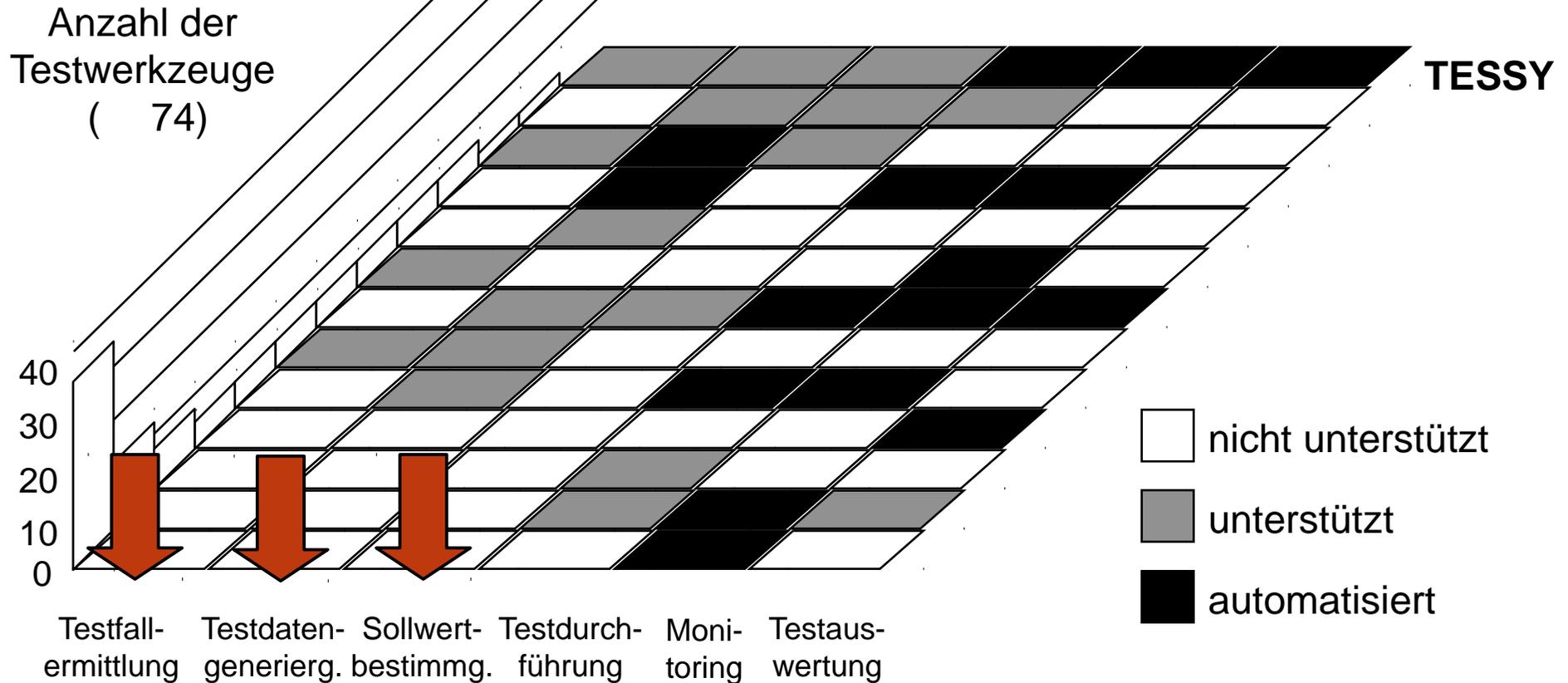
- ▶ Aspektorientierung reduziert die Komplexität
 - mehrere Klassifikationen ermöglichen es, das Problem in Dimensionen aufzuteilen
 - Visualisierung, auch gerade für Manager und Gutachter
- ▶ Abdeckungsgrad
 - Wohlüberlegte Testfallkonstruktion deckt die meisten Fehlerfälle ab
- ▶ Test-Ende-Kriterium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

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)

21



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



Werkzeugpalette von *TESSY*

22

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

23

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



Steuerflussorientierter Test (code coverage)

2	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
	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 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)

25

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

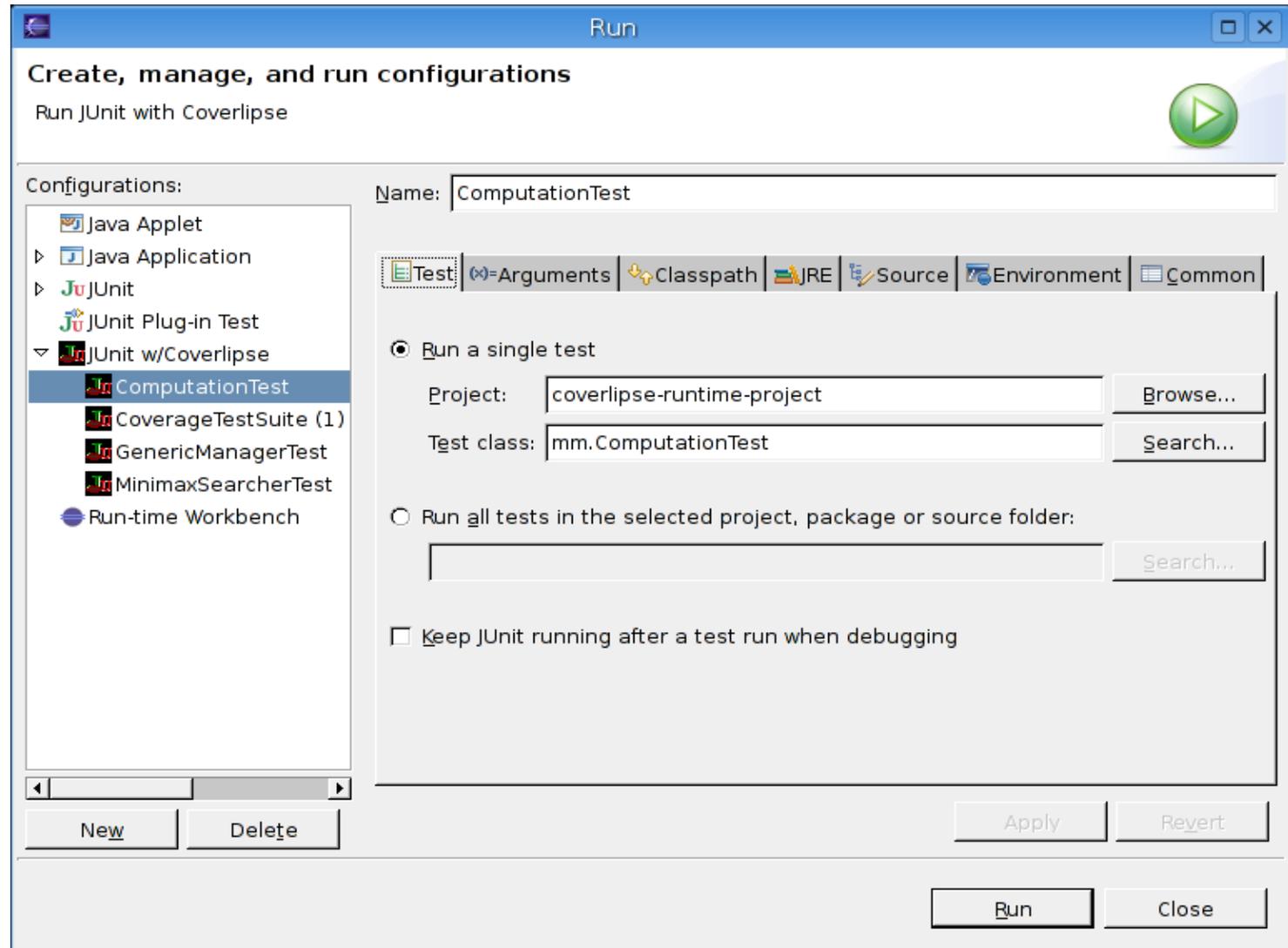
Pro



Bsp.: Coverlipse basiert auf JUnit

26

- ▶ Selektion von Junit-Testfällen und deren Pfadabdeckungsanalyse



Coverage Tools – Werkzeuge zur Pfadabdeckung

27

- ▶ Paketfilterung stellt die Pakete zur Pfadabdeckungsanalyse ein

The screenshot shows the Eclipse 'Run' dialog box for a configuration named 'CoverageTestSuite (1)'. The 'Package Filter' tab is selected, displaying a list of 'Defined filters':

- com.schneide.quantity
- com.schneide.quantity.radioactivityQuantities

Buttons for managing filters include 'Add inclusive filter...', 'Add exclusive filter...', 'Up', 'Down', 'Remove', 'Enable all', 'Disable all', and 'Show invariants...'. At the bottom, a dropdown menu indicates that 'Packages not matching a defined filter will be excluded'. The 'Run' button is visible at the bottom right.



Coverlipse

28

- ▶ block coverage / statement coverage

de.uka.ipd.coverage.ssa.helpers.BlockVarCombinator

Message	Li	cove	unco	Resource
✓ This line was fully covered	25			BlockVarCombinator.java
✓ This line was fully covered	28			BlockVarCombinator.java
! This line was not covered	29			BlockVarCombinator.java
✓ This line was fully covered	31			BlockVarCombinator.java
✓ This line was fully covered	32			BlockVarCombinator.java



Coverlipse

29

► All-uses-coverage

The screenshot shows the Eclipse IDE with the Coverlipse plugin. The 'All-Uses Coverage' view is active, showing a tree structure with 'simpletests' and 'Computation'. The code editor displays the following Java code:

```
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 j = 0; j < m; j++) {
```

The 'Coverlipse Markers View' at the bottom displays 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 meinInt	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

30

- Problembeschreibung aus einem Use

Java - Computation.java - Eclipse Platform

File Edit Source Refactor Navigate Search Project Run Window Help

Coverlip... All-Uses Coverage
Clear results

simpletests
Computation

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

Show all definitions
Show all definitions
Show uses of the variable meinInt
Show uses of the variable result

Problem description: Definition of the variable result

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 meinInt	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

31

- ▶ all-uses-coverage information

The screenshot shows the Eclipse IDE with the Coverlipse plugin. The main editor displays the source code of the `Computation` class. The `add` method is highlighted, and the `mainInt` variable is marked as uncovered. The `multiply` method is also visible.

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

The **Coverlipse Markers View** at the bottom shows the following table:

Message	Line	covered uses	uncovered uses	Resource
Definition of the variable mainInt	13	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



71.3 Testautomatisierung mit Test-Frameworks

32

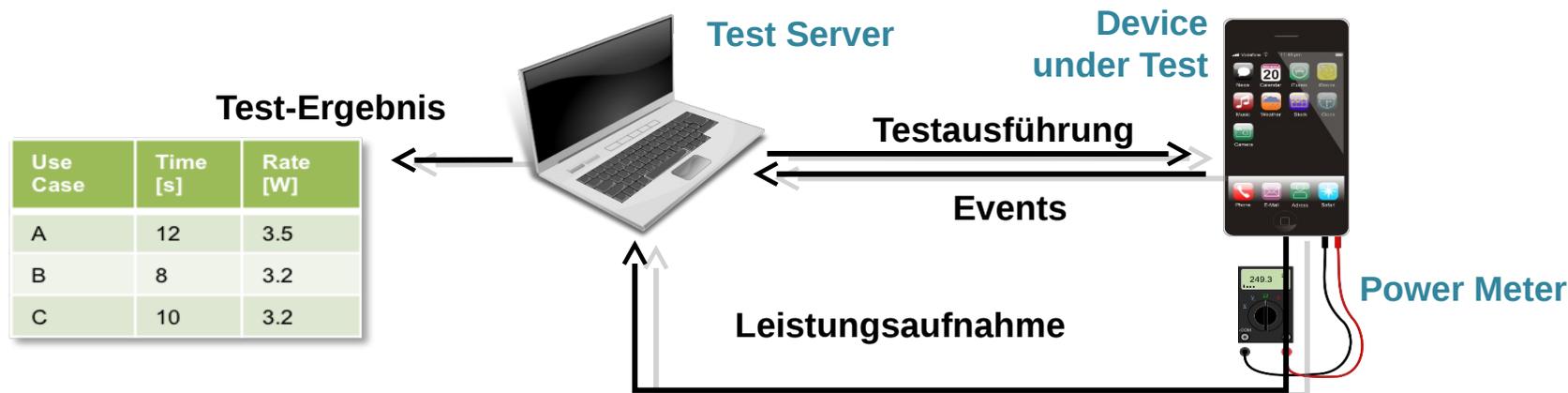
- JouleUnit (courtesy Claas Wilke)
YouTube-Video zum Thema: <http://is.gd/energyLabel>
Mehr Infos zum Projekt:
<http://www.qualitune.org/>
<http://www.jouleunit.org/>
claas.wilke@tu-dresden.de



EnergieTest mit JouleUnit

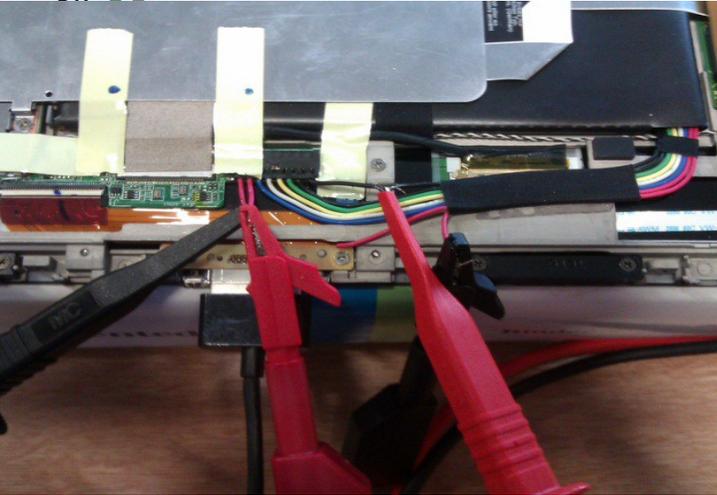
33

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



Energie-Test

24



JouleUnit + QMark

35

- ▶ 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)

36

Prof. U. Alßmann, Softwareentwicklungswerkzeuge (SEW)

The screenshot displays the JouleUnit Eclipse Platform interface during a test run. The main window is titled "Test Run Details" and shows a scatter plot of various metrics over time (0 to 35 seconds). The metrics include Power Rate [mW], CPU Frequency [MHz], WiFi Traffic [Byte], and LCD Brightness [0..255]. The plot shows a significant increase in power rate and CPU frequency during the test execution phase, which is highlighted by a green shaded area.

The "Test Run Results" window shows the following table:

Test Case	Start [ms]	Stop [ms]	Duration [ms]	Avg. Power Rate [mW]	Energy Consumption [mJ]
org.jouleunit.android.test.Idle	1350993670179	1350993672178	1999,00	2764,28	5525,79
testRawWebsite(easy.browser.classic.test.EasyBro...	1350993672513	1350993703363	30850,00	3046,41	93981,65

The Console window shows the following log output:

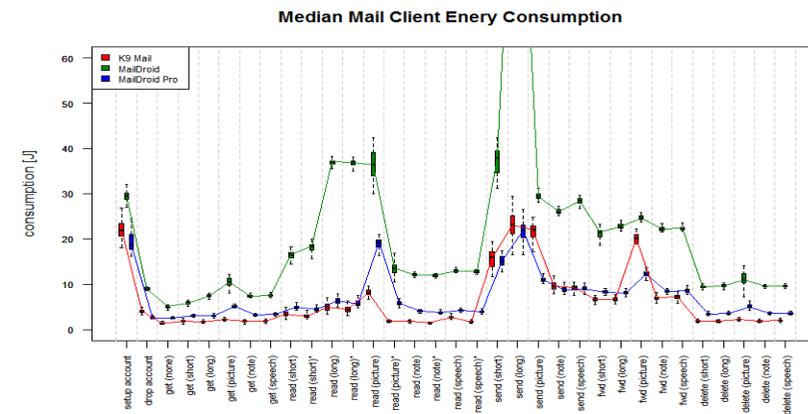
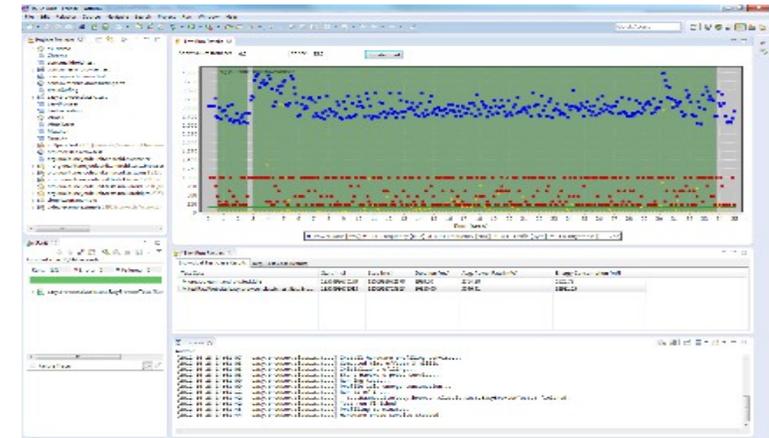
```
Android
[2012-10-23 14:01:07 - easy.browser.classic.test] Install hardware profiling service...
[2012-10-23 14:01:08 - easy.browser.classic.test] Computed time offset: 3 millis.
[2012-10-23 14:01:08 - easy.browser.classic.test] Initialize profiling...
[2012-10-23 14:01:08 - easy.browser.classic.test] Start Hardware probe service...
[2012-10-23 14:01:09 - easy.browser.classic.test] Running tests...
[2012-10-23 14:01:09 - easy.browser.classic.test] Profile idle energy consumption...
[2012-10-23 14:01:11 - easy.browser.classic.test] Run #1 of 1 ...
[2012-10-23 14:01:42 - easy.browser.classic.test] testRawWebsite(easy.browser.classic.test.EasyBrowserTests) finished.
[2012-10-23 14:01:42 - easy.browser.classic.test] Test run finished
[2012-10-23 14:01:43 - easy.browser.classic.test] Profiling terminated.
[2012-10-23 14:01:44 - easy.browser.classic.test] Hardware probe service stopped.
```



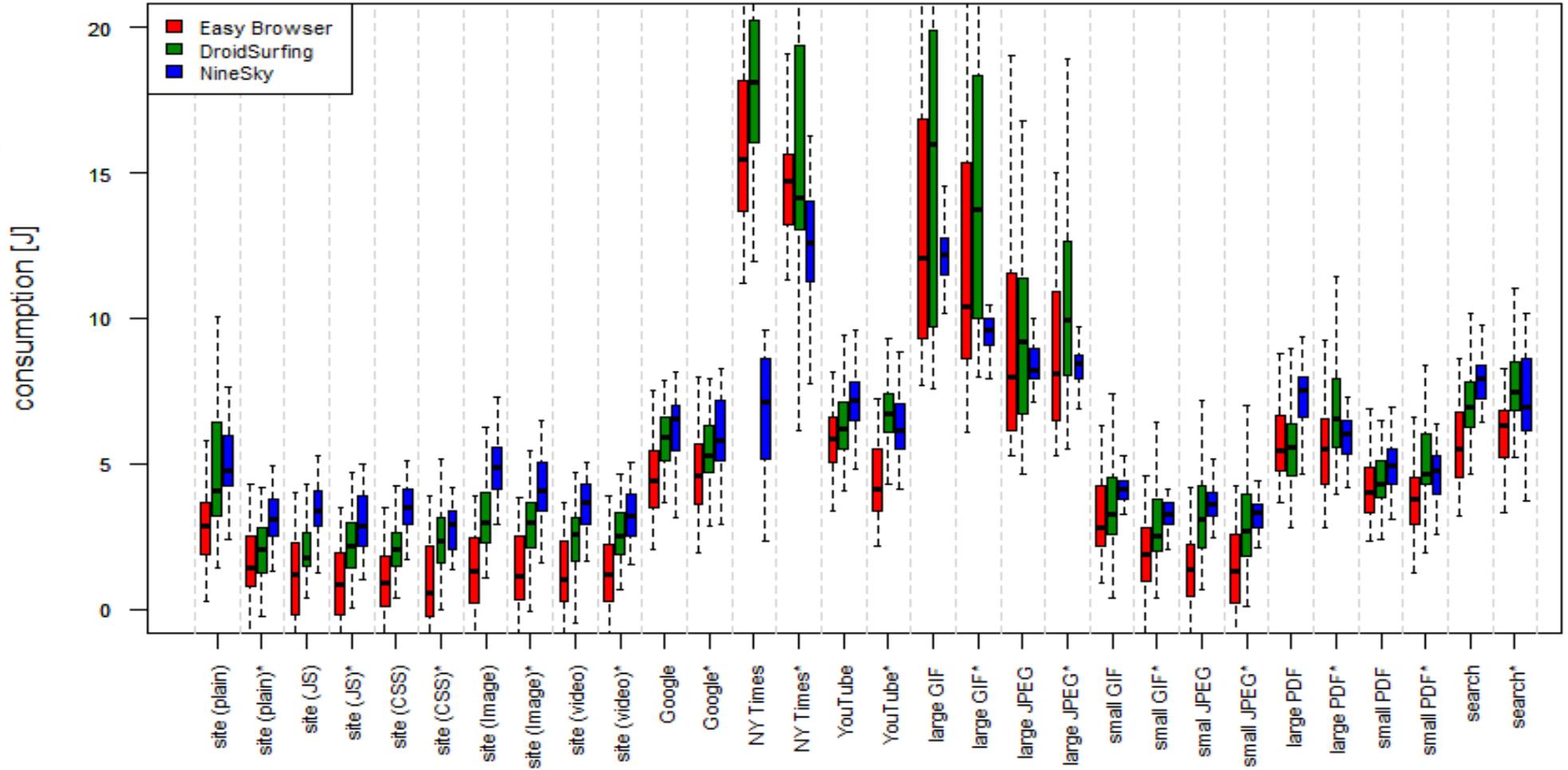
„Ähnliche“ Anwendungen vergleichen

37

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



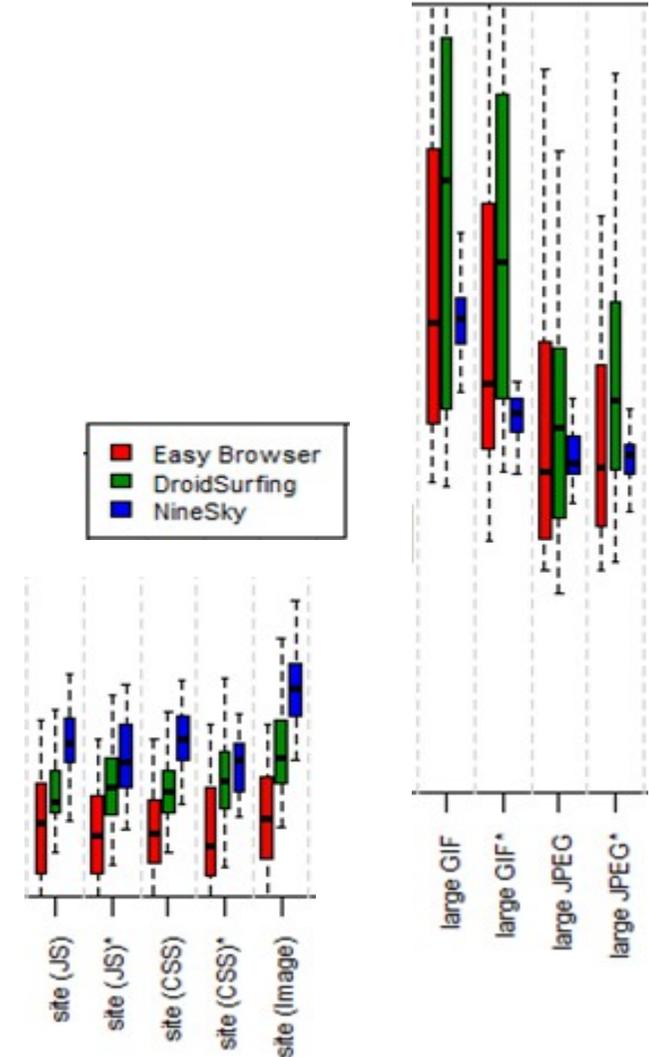
Median Web Browser Energy Consumption



Vergleich von Web Browsern

39

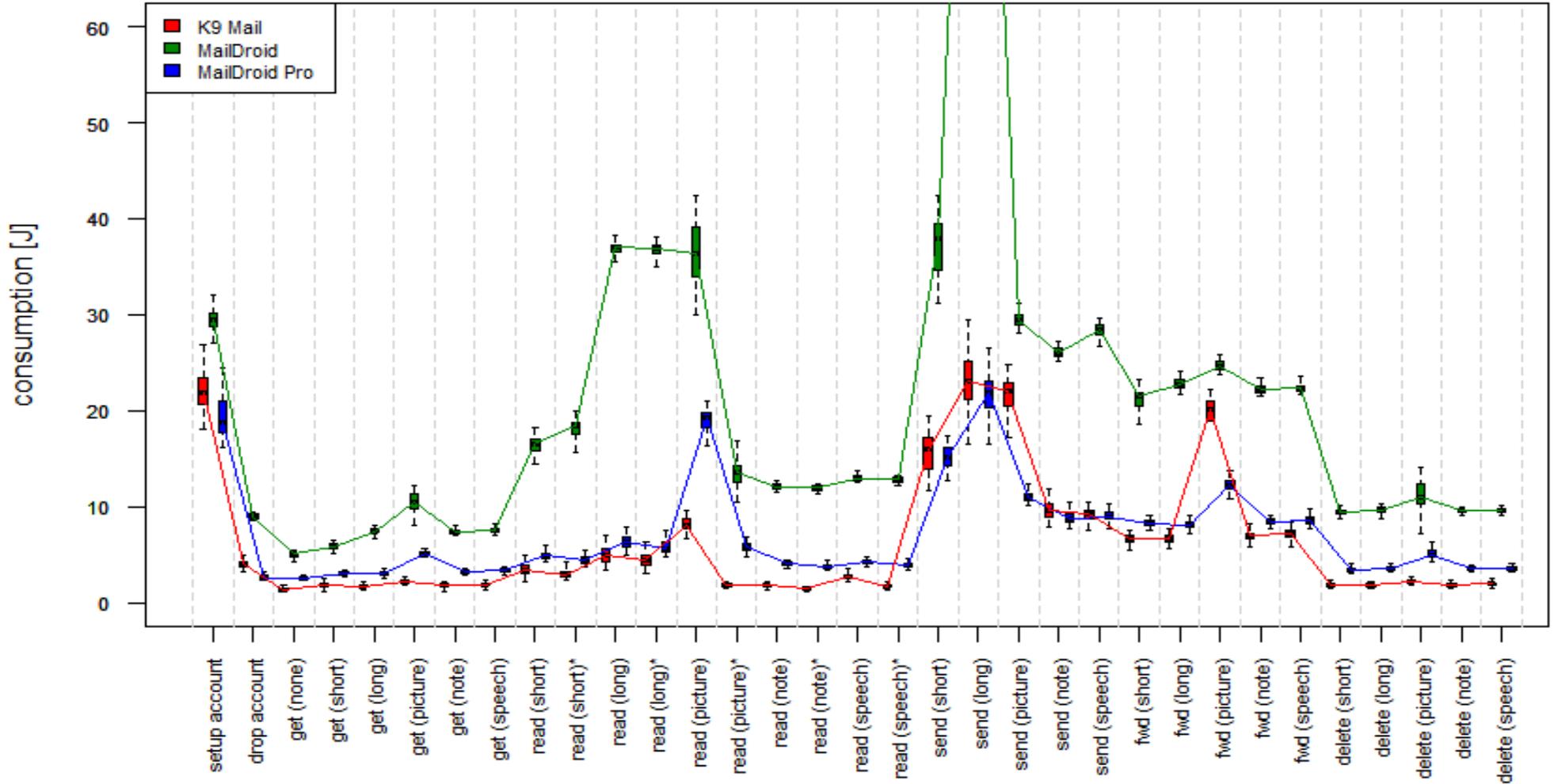
- ▶ Hohe Varianz in Messungen (durch hohe Varianz in Antwortzeiten)
 - Aber: vergleichbare Trends
- ▶ NineSky schlecht für kleinere Seiten
 - Aber: besser für große Bilder (da schneller)
- ▶ Advertisement in EasyBrowser, DroidSurfing negative Auswirkung nur bei langen Ladezeiten
- ▶ Verschiedene Browser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





Median Mail Client Energy Consumption

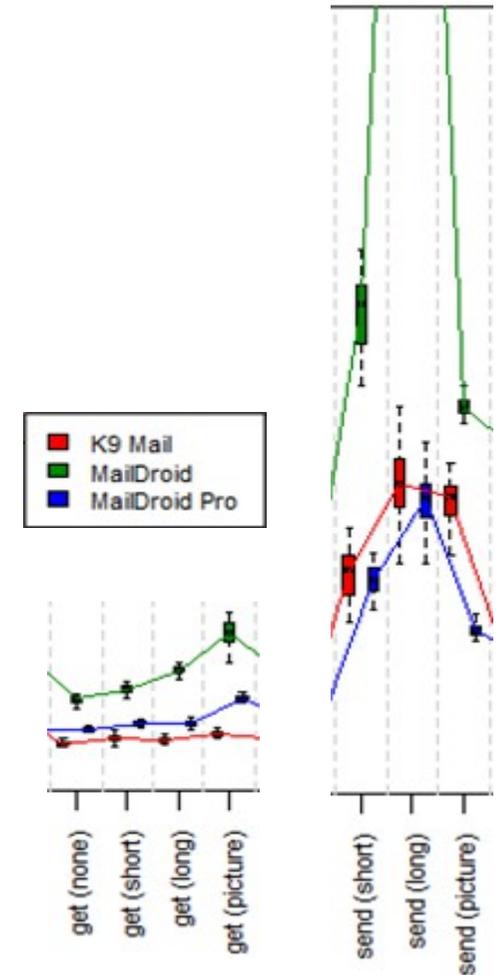
Prof. Dr. A. Özcan, Çukurova University, Adana, Turkey (oazcan@cu.edu.tr)



Vergleich von Email Clients

41

- ▶ 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

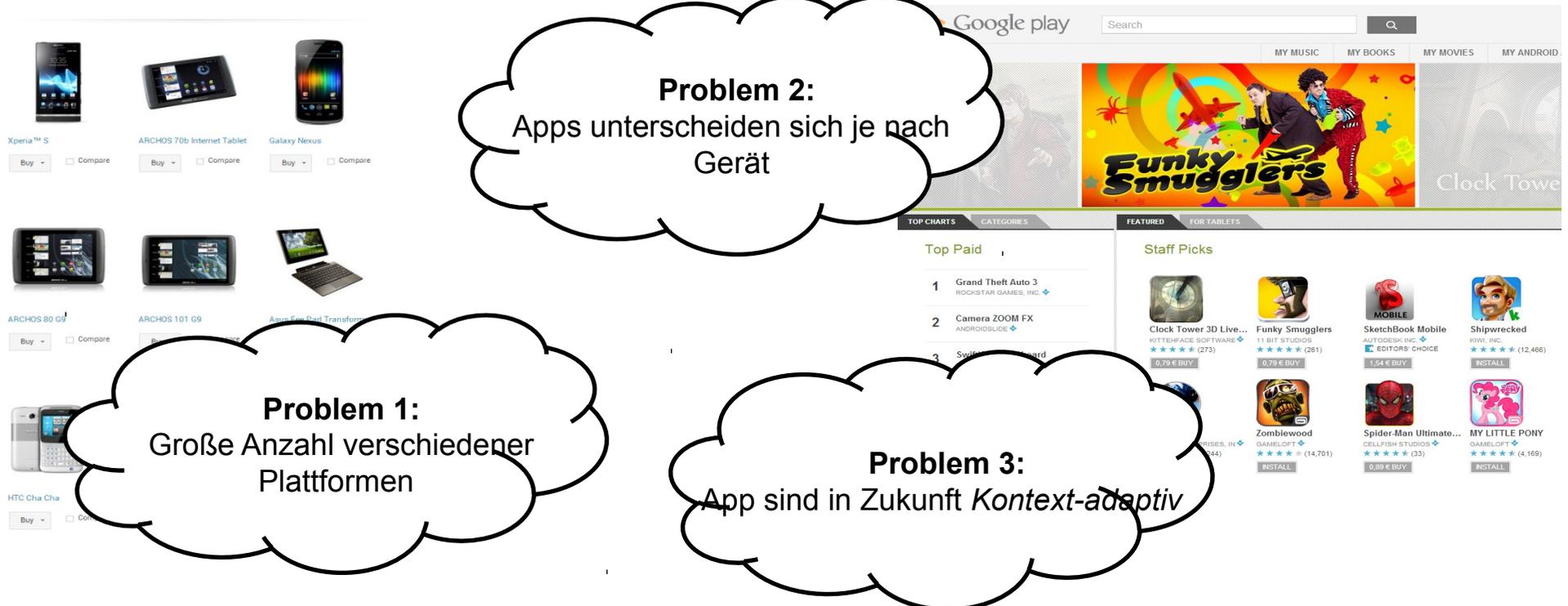


71.3.2 Modellgetriebenes Testen mit dem Werkzeug MATE

42

- Georg.Pueschel@tu-dresden.de

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



The screenshot shows the Google Play Store interface. On the left, there are sections for different device types: smartphones (Xperia™ S, ARCHOS 70b Internet Tablet, Galaxy Nexus) and tablets (ARCHOS 80 G9, ARCHOS 101 G9, Asus EeePC Transformer). On the right, there are sections for 'Top Paid' and 'Staff Picks' featuring various apps like 'Grand Theft Auto 3', 'Camera ZOOM FX', 'Clock Tower 3D Live...', 'Funky Smugglers', 'SketchBook Mobile', 'Shipwrecked', 'ZombieWood', 'Spider-Man Ultimate...', and 'MY LITTLE PONY'.

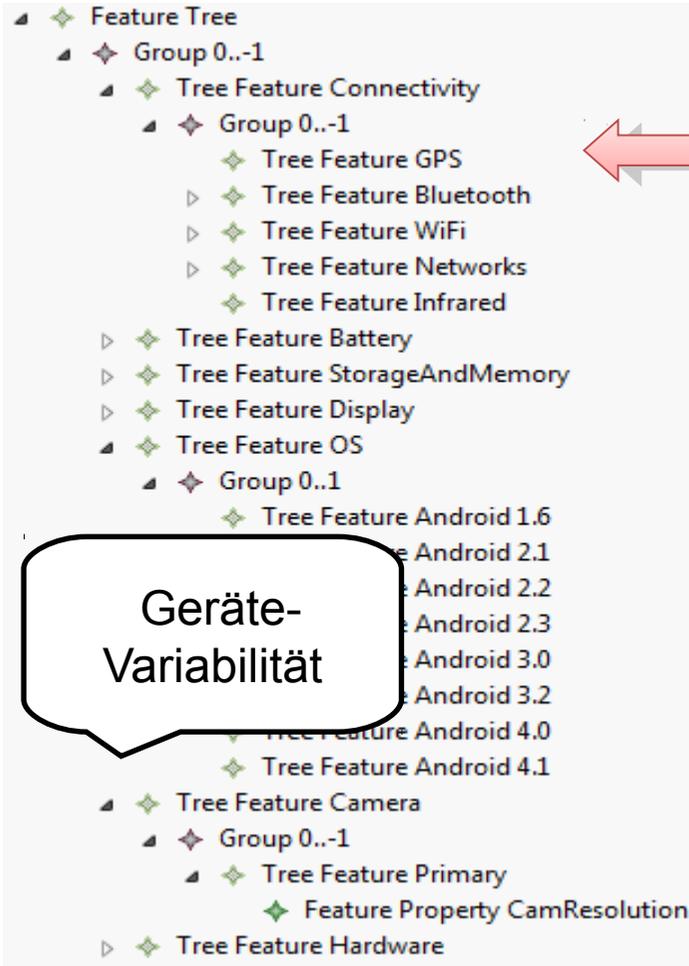
Problem 2:
Apps unterscheiden sich je nach
Gerät

Problem 1:
Große Anzahl verschiedener
Plattformen

Problem 3:
Apps sind in Zukunft *Kontext-adaptiv*

Plattformmanagement durch attributierte Feature-Modelle

44



Geräte-Variabilität

- Configuration LG Optimus 2
- Configuration LG Ignite
- Configuration Sharp AQUOS PHONE THE HYBRID So
- Configuration Fujitsu REGZA Phone T 01D
- Configuration Sharp AQUOS PHONE IS13SH
- Configuration Galaxy Nexus
- Configuration Samsung Galaxy Note
- Configuration HTC Amaze 4G
- Configuration PocketBook A10
- Configuration HTC Rezound
- Configuration DROID RAZR by MOTOROLA
- Configuration HTC Vivid
- Configuration SKY VEGA LTE
- Configuration Sony Ericsson Xperia neo V
- Configuration Sony Ericsson Xperia active
- Configuration HTC Sensation XL with Beats Audio

Konkrete Geräte

- Domain Pixels
 - Value 2.0 megapixels

Finite Attributwert-mengen

- Value 8.0 megapixels
- Value 8.1 megapixels
- Value 9.6 megapixels
- Value 12.0 megapixels
- Value 12.1 megapixels
- Value 12.19 megapixels
- Value 12.2 megapixels
- Value 13.1 megapixels
- Value 16.1 megapixels

Binding

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

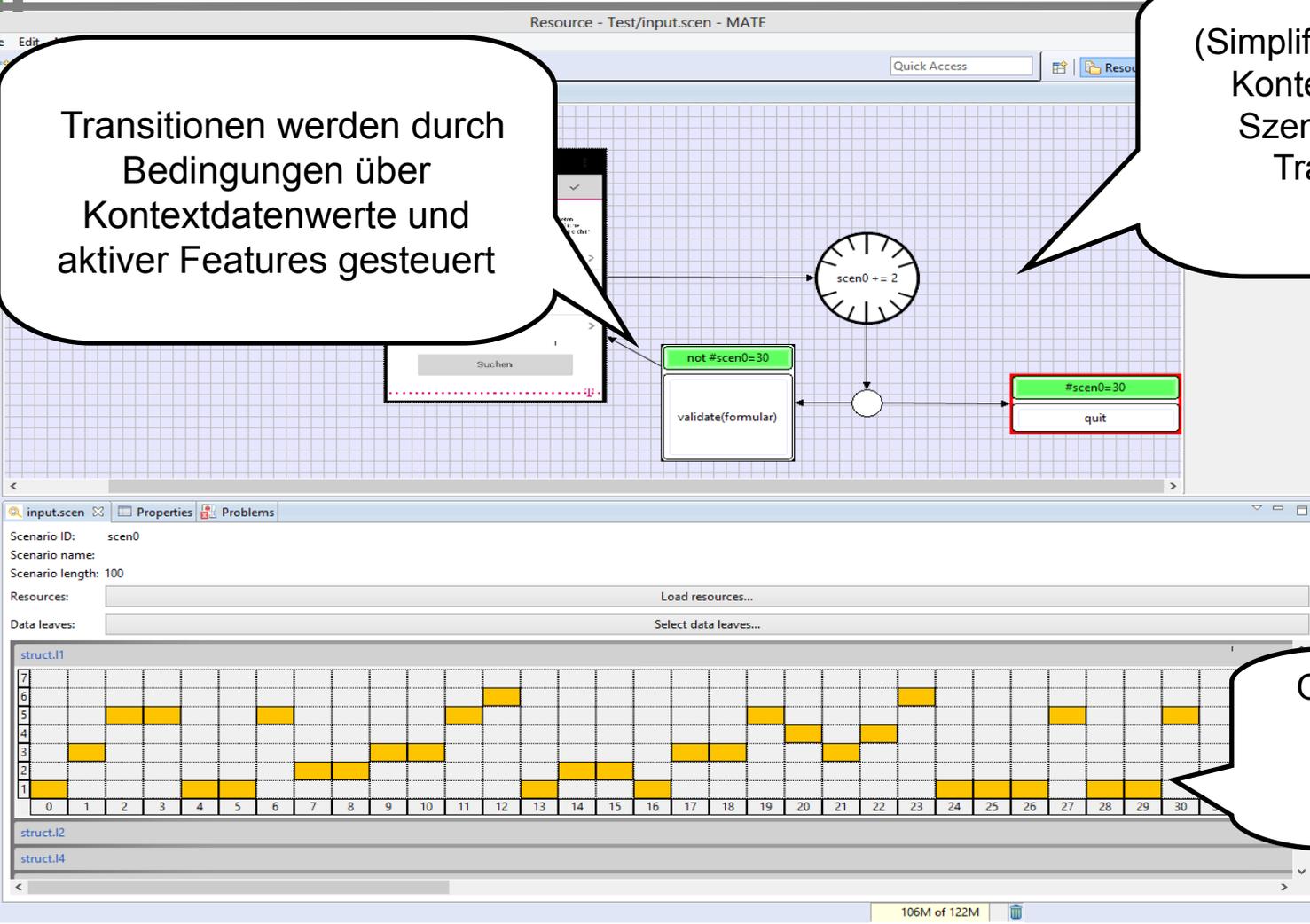


Adaptionsbeschreibung

46

Transitionen werden durch Bedingungen über Kontextdatenwerte und aktiver Features gesteuert

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



OCM-Alternative: Szenarien beschreiben Werteänderungen von Kontextdaten

Im Ergebnis erzeugt ein Generator aus durch Erreichbarkeitsanalyse Testfälle.

71.4 Funktionalität und Werkzeuge ausgewählter Test-Umgebungen



47

Funktionalität von *LOGISCOPE*

48

- ▶ 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 Wartung 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.

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

Werkzeuge von LOGISCOPE (2)

50

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).

imbus TestBench

51



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

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
- Endpreis berechnen "ohne" Rabatt
 - CarConfig Starten
 - Fahrzeug konfigurieren
 - Fahrzeug wählen CBR
 - Sondermodell wählen

Interaktion

Fahrzeug wählen CBR

Parameter	Wert
Fahrzeug	15

Fehler Fehler hinzufügen

Interaktion: Fahrzeug wählen CBR Bemerkungen

-Beschreibung

Fahrzeug aus der Liste der Fahrzeuge wählen

-Bemerkungen zur Durchführung

-Bemerkungen zur Spezifikation

Benutzerdefinierte Felder der Durchführung

<für diesen Knotentyp können Benutzerdefinierte Felder nicht definiert werden>

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

Aufgezeichnete Attribute

Tester

Aktueller Benutzer

Tester

Letzte Änderung des Ergebnisses

Aktuelles Ergebnis Zu prüfen

Ergebnis-Datum (DD.MM.YYYY)

Ergebnis-Zeit (HH:MM:SS)

Zeitmessung

Geplante Durchführungszeit (DD:HH:MM:SS.SSS)

Aktuelle Durchführungszeit (DD:HH:MM:SS.SSS)



- **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



55

71.4.1 In-Vitro-Testläufe mit Debuggern



56

Entwanzer (Debugger)

57

- ▶ 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)

58

- ▶ 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

The screenshot displays the DDD interface for debugging a C program. The top window shows a graph of a linked list with three nodes. Each node contains a 'value' field and a 'next' pointer. The first node has value 85 and points to the second node (value 86). The second node points to the third node (value 86). The third node's 'next' pointer is null. The interface includes a menu bar (File, Edit, View, Program, Commands, Status, Source, Data, Help), a toolbar with various debugging actions, and a command prompt where 'list->self' is entered. Below the graph is a code editor showing C code for a linked list. A 'DDD Tip of the Day #5' dialog box is overlaid on the code editor, advising the user to use 'Edit->Undo' to revert to the previous state. The bottom status bar shows '(gdb) graph display *(list->next->next->self) dependent on 4' and '(gdb) |'.

ddd Settings

59

The screenshot displays the DDD (Data Display Debugger) interface. The main window shows the source code of a C program named 'cxxtest.C'. The code is as follows:

```
int main(int /* argc */, char ** /* argv */)
{
}

Copy
Copy
(gdb) file /public/source/programming/ddd-3.2/ddd/cxxtest
Reading symbols from /public/source/programming/ddd-3.2/ddd/cxxtest...done.
(gdb) |
```

The 'DDD: Debugger Settings' dialog box is open, showing the following GDB settings, all of which are currently unchecked:

- 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 '\nnn'

Buttons at the bottom of the dialog include 'Apply', 'Reset', 'Close', and 'Help'. The status bar at the bottom of the debugger window reads: 'Look up a line, file, function, variable or breakpoint in the source'.



ddd Registerwerte

60

DDD: /public/source/programming/ddd-3.2/ddd/cxxtest.C

File Edit View Program Commands Status Source Data Help

0: 'info all-registers'

int main(int /*
{
int i = 42;
tree_test();
i++;
list_test(i);
i++;
array_test(i);
i++;
string_test(i);
i++;
plot_test(i);
i++;
type_test(i);
--i;
cin_cout_test(i);
return 0;
}

Registers

eax:	0x401a2db8	1075457464
ecx:	0x8049c94	134519956
edx:	0x401a1234	1075450420
ebx:	0x401a41b4	1075462580
esp:	0xbffef30	-1073746128
ebp:	0xbffef48	-1073746104
esi:	0xbffef94	-1073746028
edi:	0x1	1
eip:	0x8049ca1	134519969
eflags:	0x286	IOPL: 0
flags:	PF SF IF	
orig_eax:	0xffffffff	-1
cs:	0x23	35

Integer registers All registers

Close Help

0x8049c9a <
0x8049ca1 <
0x8049ca6 <
0x8049ca9 <
0x8049cac <
0x8049caf <
0x8049cb0 <
0x8049cb5 <
0x8049cb8 <main+36>: incl 0xffffffffc(%ebp)
0x8049cbb <main+39>: call 0x8049428 <array_test(void)>
0x8049cc0 <main+44>: incl 0xffffffffc(%ebp)
0x8049cc3 <main+47>: call 0x8049404 <string_test(void)>

(gdb) I





Appendix

61

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

Fehlerhafte Software produziert Kosten

62

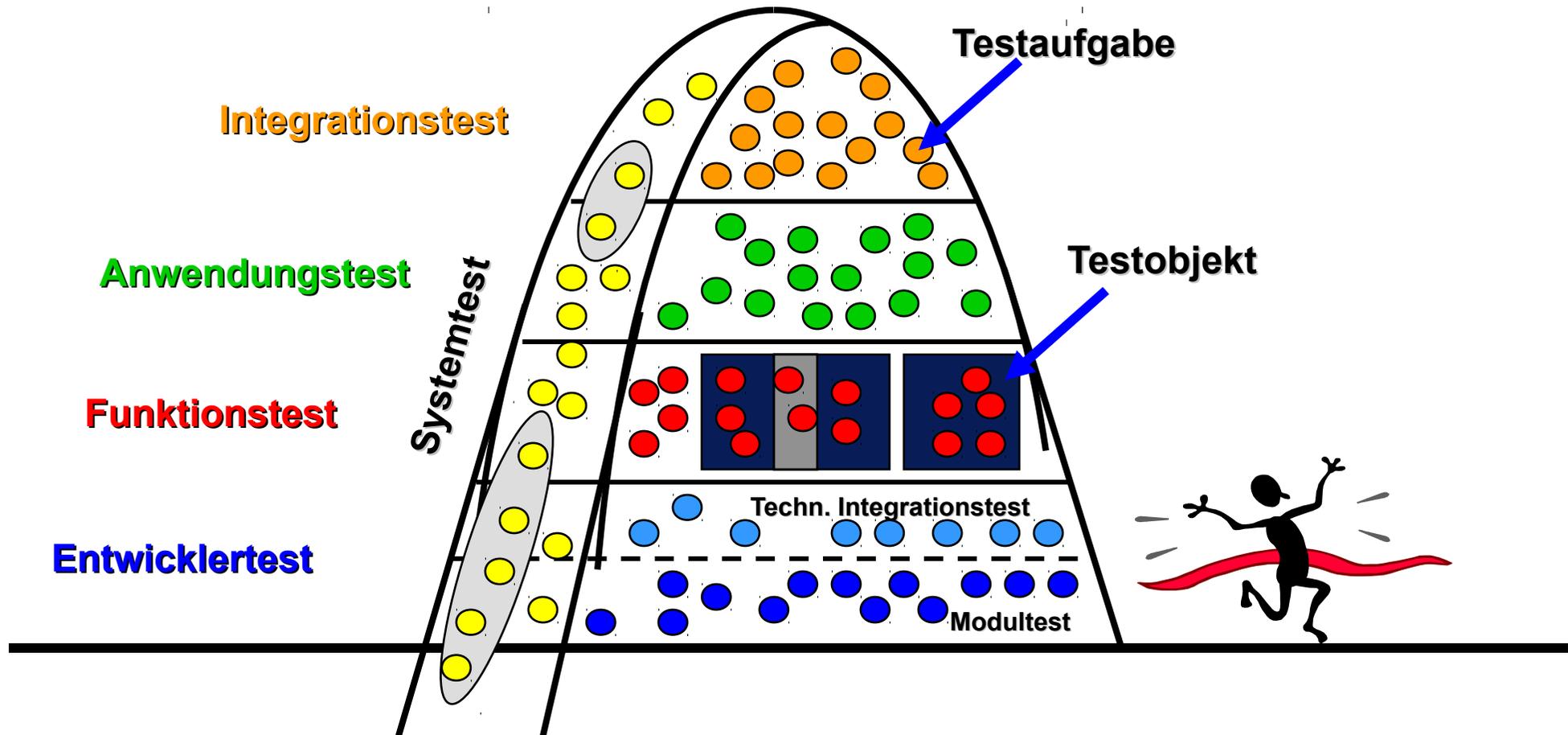
- ▶ SW-Fehler pro Jahr in Deutschland verursachen Kosten in Höhe von 80 Milliarden EUR (Studie von Lot-Consulting bei 922 deutschen Unternehmen)
- ▶ Produktivitätsverlust aufgrund stillstehender Computer kostet 70 Milliarden EUR (Handelsblatt)
- ▶ Großteil der Fehler tritt bereits in der SW-Entwicklungsphase 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/files/181001.pdf>

Der Testberg führt zu Angstgefühlen im Entwickler

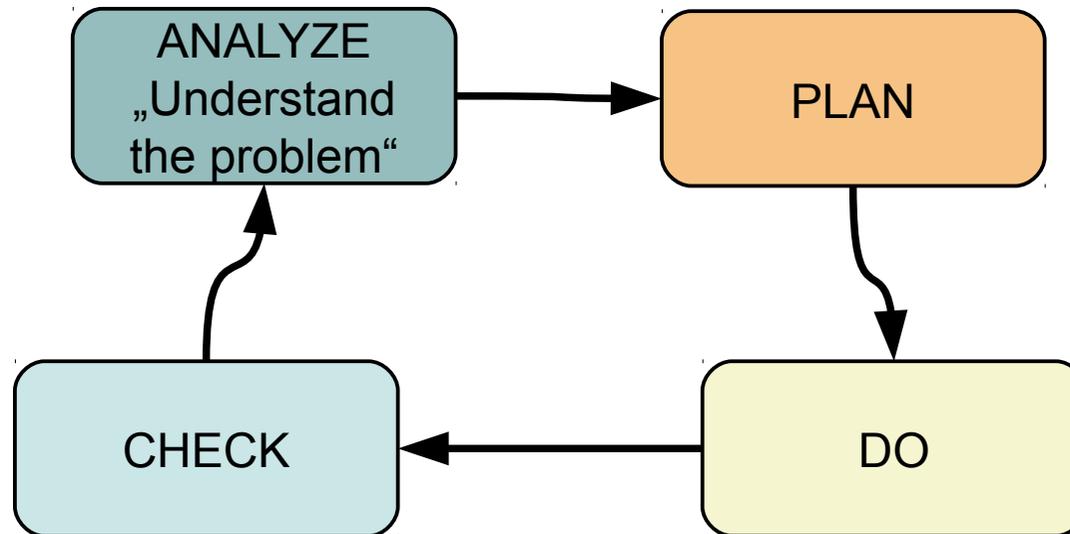


- **Entwurfstest:**
Testobjekt sind alle Dokumente (und Modelle), in denen die Fachlichkeit der Anwendung beschrieben ist. Sie werden durch die Prüf-Werkzeuge der CASE auf Korrektheit, Konsistenz, Kohärenz und Abgeschlossenheit getestet.
- **Entwicklertest (Einheitentest, unit test):**
 - **Klassentest** 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.

Testing zur Qualitätssicherung wendet den Feedbackzyklus “Polya Cycle” an

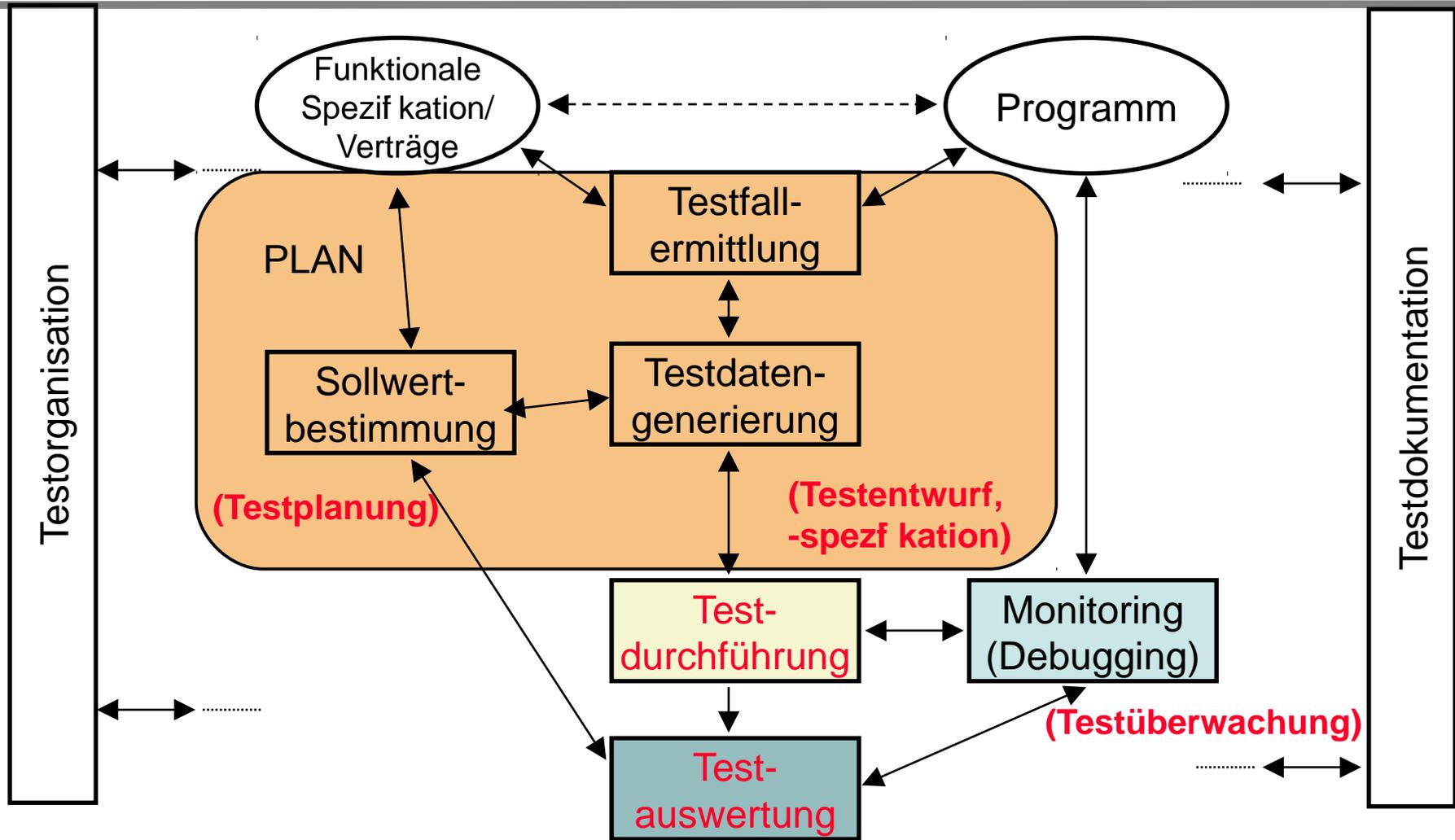
65

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



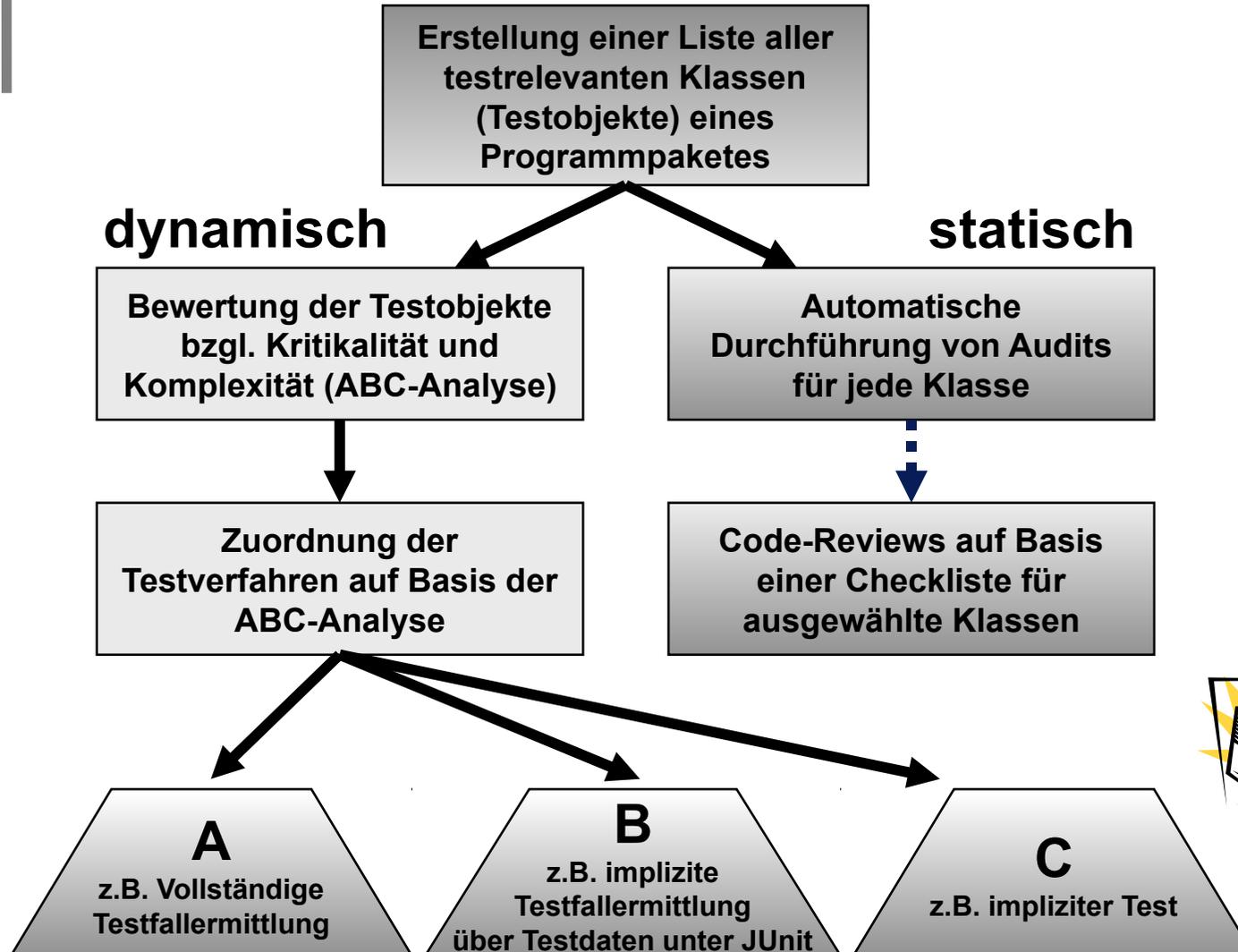
Test-Management

66



Roadmap für den Entwicklertest

67



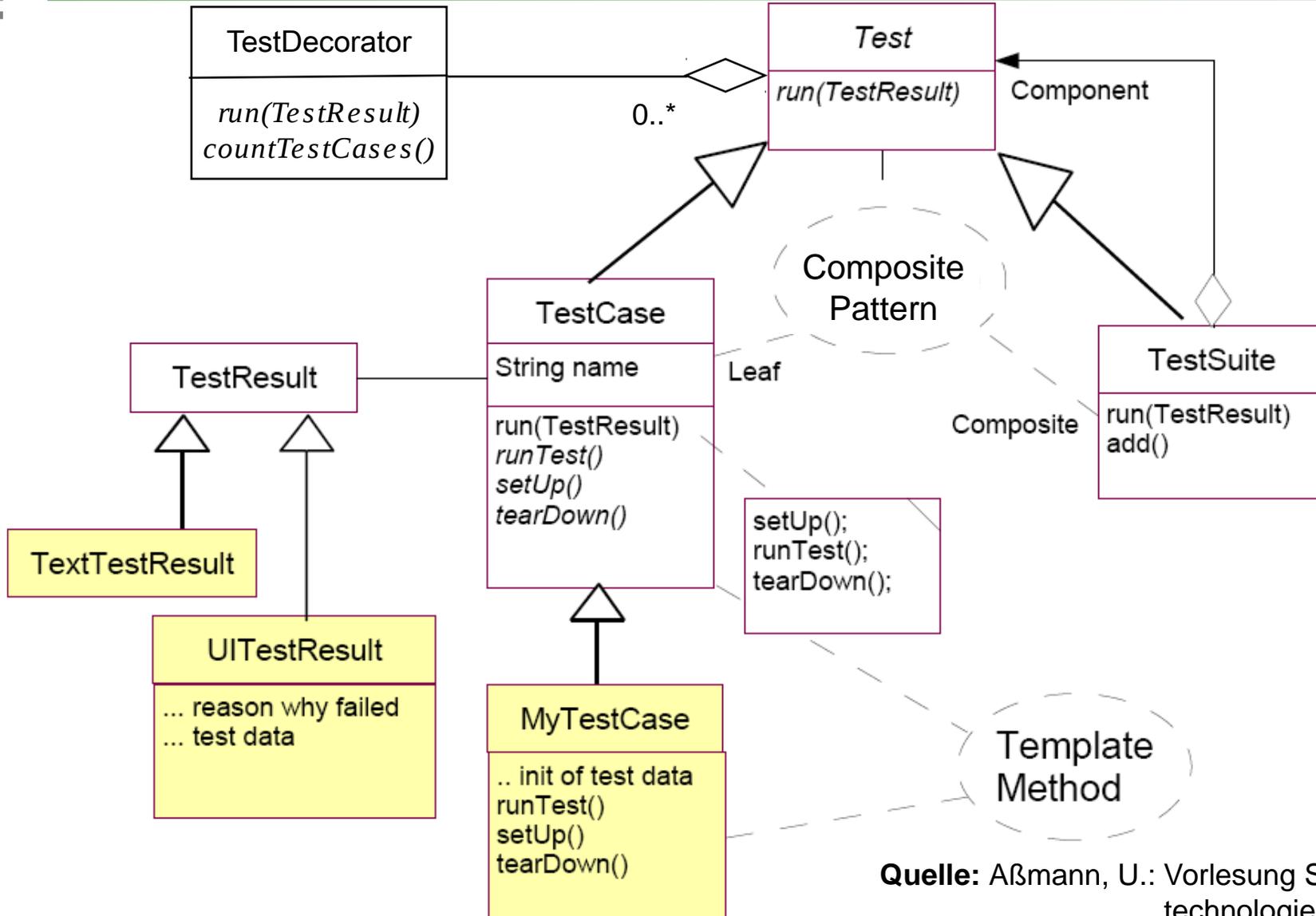
Framework *JUnit* für Komponententest

68

- **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)

Teststruktur von JUnit

69



Quelle: Aßmann, U.: Vorlesung Software-technologie II SS06



Testklassen („Werkzeuge“) von JUnit

70

Test	Schnittstellenklasse, die es nach dem <i>Composite Pattern</i> 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	<ul style="list-style-type: none">- 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>



Entwicklertest: FURPS Qualitätsziele

71

Prof. U. Aßmann, Softwareentwicklungswerkzeuge (SEW)

Produkteigenschaften

statisch

dynamisch

Modelle

Code

Testlauf

In-Vitro-Lauf

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

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

Testendekriterien

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 findet im Rahmen einer SW-Entwicklungsmethode statt

73

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