

Lecturer: Dr. Sebastian Götz

Prof. Dr. rer. nat. Uwe Aßmann
Institut für Software- und Multimediatechnik
Lehrstuhl Softwaretechnologie
Fakultät für Informatik
TU Dresden
<http://st.inf.tu-dresden.de>
Wintersemester 2019, 27.11.2019

12. Validation Tools

- 1) Code-Centric Test Environments
 - 1) Coverage-based test tools
 - 2) JouleUnit energy test framework
 - 3) Eclipse-based test tools
- 2) Requirements-Driven Test Environments
 - 1) Classification tree method and Tessy
- 3) Model-Driven Test Environments
 - 1) MATE

12.1 Code-Centric Test-Frameworks

12.1.1 Code Coverage Tools

Control-Flow Oriented White-Box Test (Code Coverage)

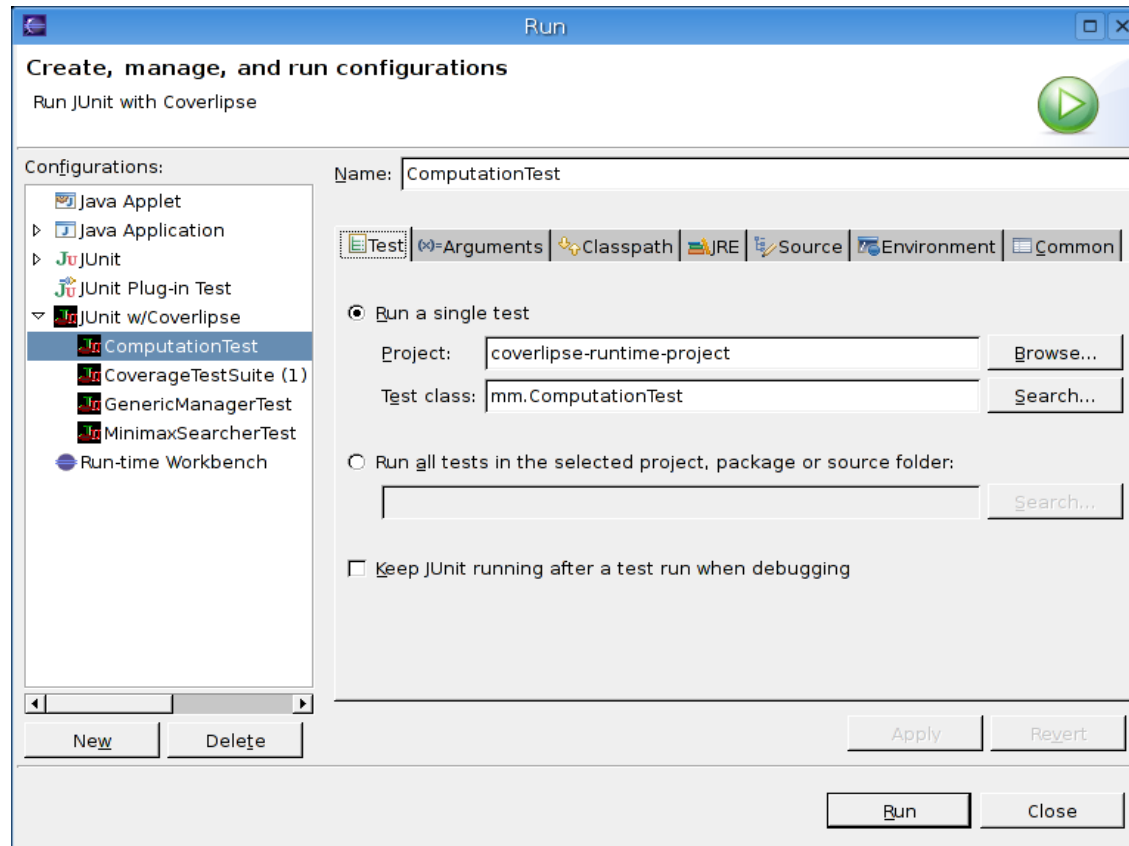
Coverage class	Technique	Purpose
Statement	How many statements are covered by how many test cases?	Discover dead code
Condition (Alternative)	How many branches are covered by how many test cases?	Cover all edges in control-flow graph
Combination of conditions	How many combinations of subsequent branches are covered by how many test cases? Coverage of acyclic paths	Problem: combinatoric explosion
Combination of independent conditions	All combinations of those conditions influencing the result of the program independently	Reduction of the effort
Path	Coverage also of cyclic paths	Im Allgemeinen unmöglich; Einschränkung auf Durchlaufsschranke k
Boundary Test	Coverage of all paths, with at most 1 run through a loop	Loop bound is $k \leq 1$
Interior Test	Coverage of all paths, with at most 2 runs through a loop	Loop bound is $k \leq 2$

Data-Flow Coverage (Datenflussorientierter Test)

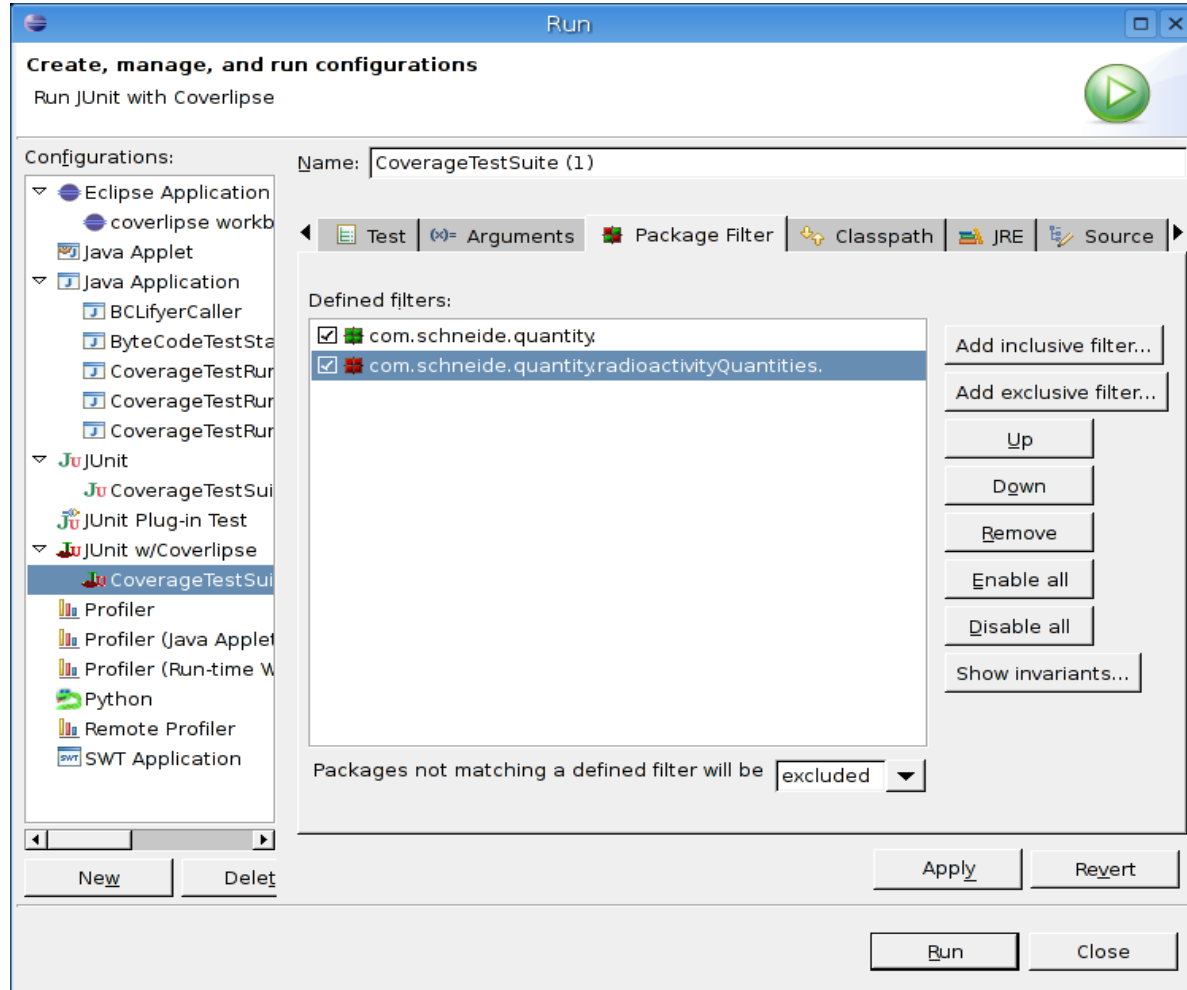
Coverage class	Technique	Purpose
All defs	For all definitions (assignments) of variables: one path to a use has to be tested	Discover dead variable assignments (definitions)
All p-uses	For a definition (assignment) of a variable, test all uses <i>in predicates</i>	Show the influence of the variable assignment to the control flow
All c-uses	For a definition (assignment) of a variable, test all uses <i>outside of predicates</i>	Show the influence of the variable assignment to the data flow

Ex.: Coverlipse based on JUnit

- ▶ Selection of Junit test cases and their path coverage analysis



Coverlipse- Selecting Packages to Analyze



Coverlipse Block Coverage / Statement Coverage

The screenshot displays the Eclipse IDE interface with the Coverlipse plugin. The main editor shows the source code of `BlockVarCombinator.java`. The `Block Coverage` view on the right shows a tree structure of the project, with `BlockVarCombinator` highlighted, indicating 89% coverage. The `Problems` view at the bottom shows a table of coverage markers.

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: All-uses Data-Flow Coverage

The screenshot shows the Eclipse IDE interface with the Coverlipse plugin. The main editor displays the source code for the `Computation` class. The left sidebar shows the 'All-Uses Coverage' view, and the bottom panel shows the 'Coverlipse Markers View' table.

```
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++) {
```

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: Problem Description of a Use of a Variable

The screenshot shows the Eclipse IDE with the Coverlipse plugin. The main editor displays the source code of the `Computation` class. A context menu is open over the line `int result = arg1 + arg2; int meinInt = 0;`, with the option "Show uses of the variable result" selected. A problem description window is open, stating: "Problem description: Definition of the variable result".

Below the editor, a table displays the results of the analysis:

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: all-uses Coverage

The screenshot displays the Eclipse IDE interface with the Coverlipse plugin. The main editor shows the source code of `Computation.java`. The `add` method is highlighted, and the `multiply` method is also visible. The `add` method contains a line with a green checkmark (covered) and a line with a red X (not covered). The `multiply` method has a line with a green checkmark (covered).

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

Message	Line	covered uses	uncovered uses	Resource
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
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

JouleUnit (courtesy Claas Wilke)

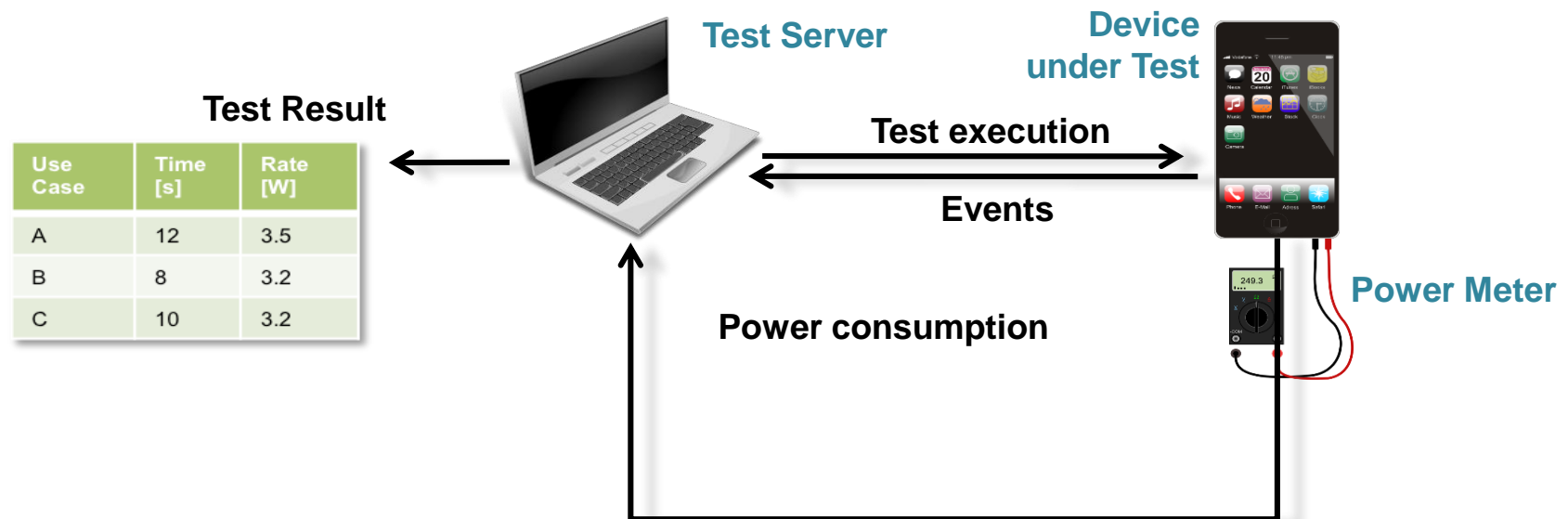
YouTube-Video: <http://is.gd/energyLabel>



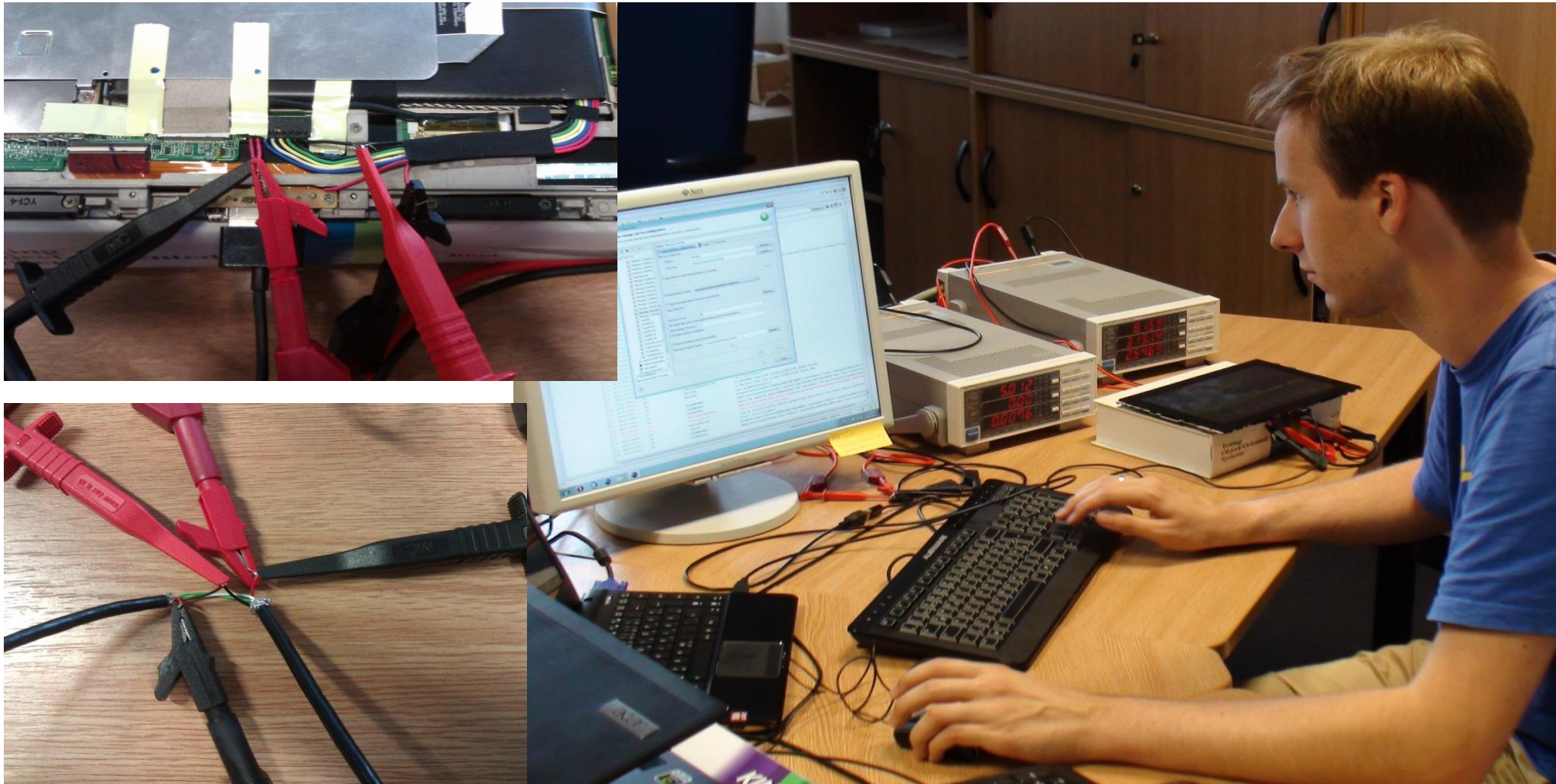
12.1.2 Code-Centric Energy Test-Frameworks

Energy Test with JouleUnit

- ▶ Generic profiling framework [WGR13]
 - Based on unit tests with jUnit: Test cases define workloads
- ▶ Reusable for different platforms, e.g., Android, NAO robots



Energy Test



JouleUnit + QMark

- ▶ Extension of JUnit for Energy tests of Android apps
- ▶ Reuse of Junit functional tests
- ▶ Execution on Smart Phone
 - Feedback on energy bugs
- Remote execution on Qmark test server
 - Hardware energy profiling



JouleUnit Workbench (Eclipse)

The screenshot displays the Eclipse IDE with the JouleUnit Workbench. The interface is divided into several panes:

- Project Explorer:** Shows a project structure with various test packages like 'asl-demo', 'Chrome', 'com.maildroid.test', etc.
- Test Run Details:** A chart showing power consumption metrics over time (0 to 35 seconds). The chart includes a legend for Power Rate [mW], CPU Frequency [MHz], WiFi Traffic [Byte], and LCD Brightness [0..255].
- Test Run Results:** A table showing individual test case results.
- Console:** Displays the execution log for the test run.

Test Run Results 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

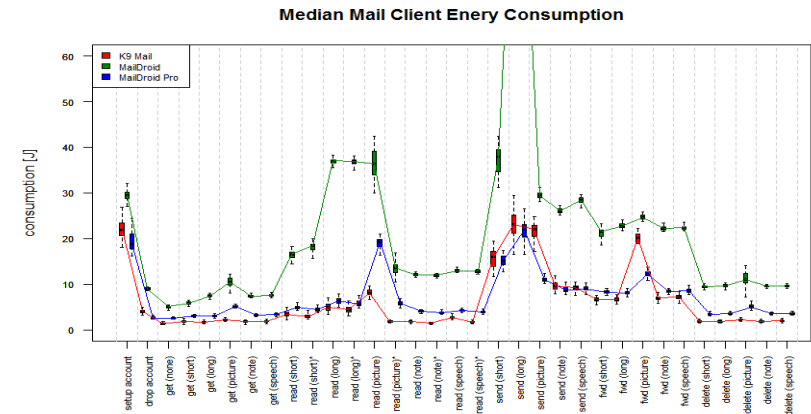
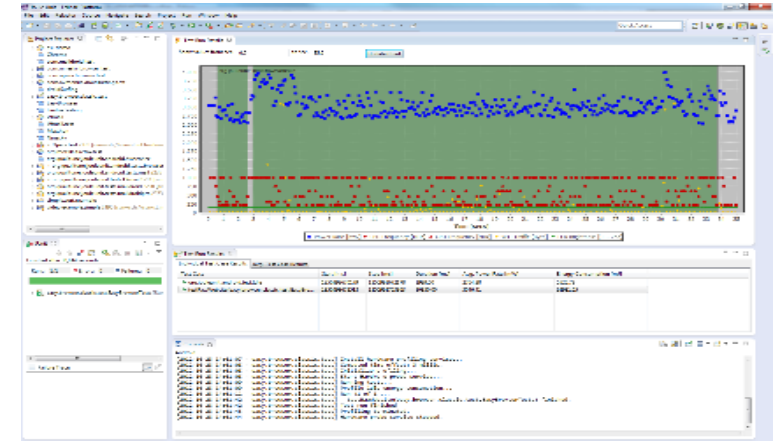
Console Log:

```

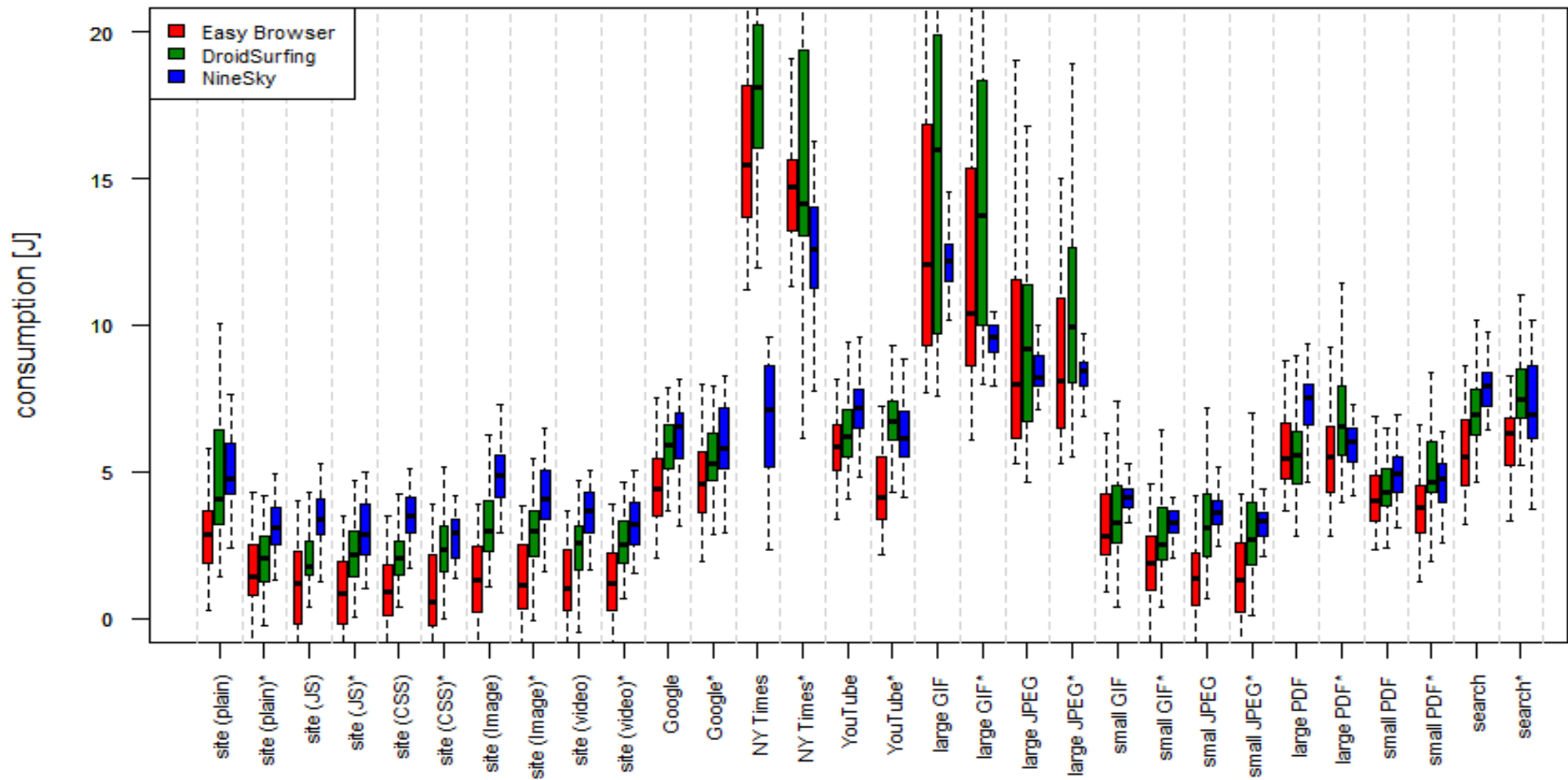
[2012-10-23 14:01:07 - easy.browser.classic.test] Install hardware profiling service...
[2012-10-23 14:01:08 - easy.browser.classic.test] Computed time offset: 3 millis.
[2012-10-23 14:01:08 - easy.browser.classic.test] Initialize profiling...
[2012-10-23 14:01:08 - easy.browser.classic.test] Start Hardware 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.
    
```


Compare „Simliar“ Apps

- ▶ Definition of benchmarks
 - Web browsing
 - Emailing
- ▶ Apps:
 - EasyBrowser, DroidSurfing, NineSky
 - K9 Mail, MailDroid, MailDroid Pro

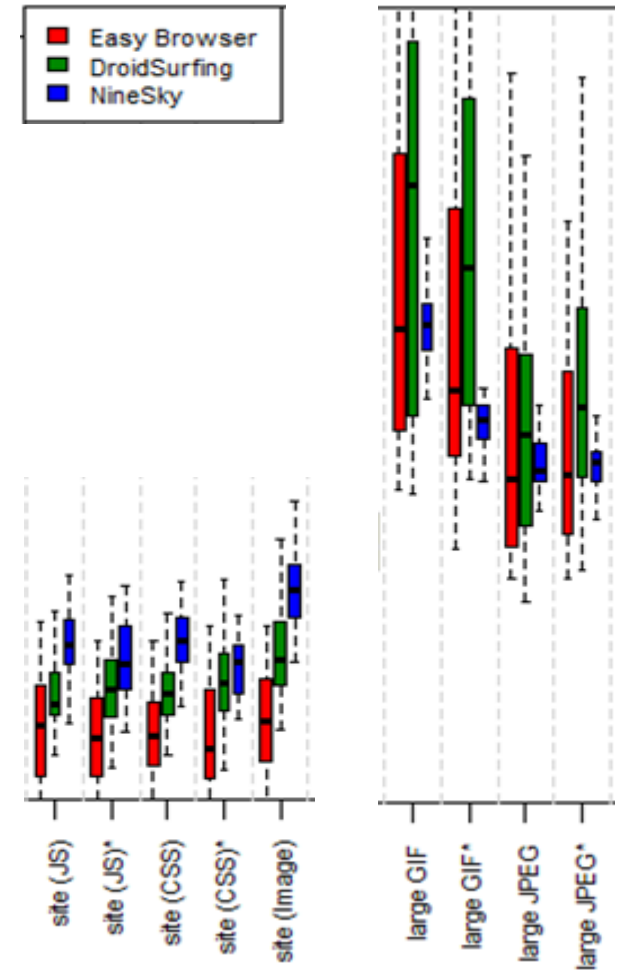


Median Web Browser Energy Consumption

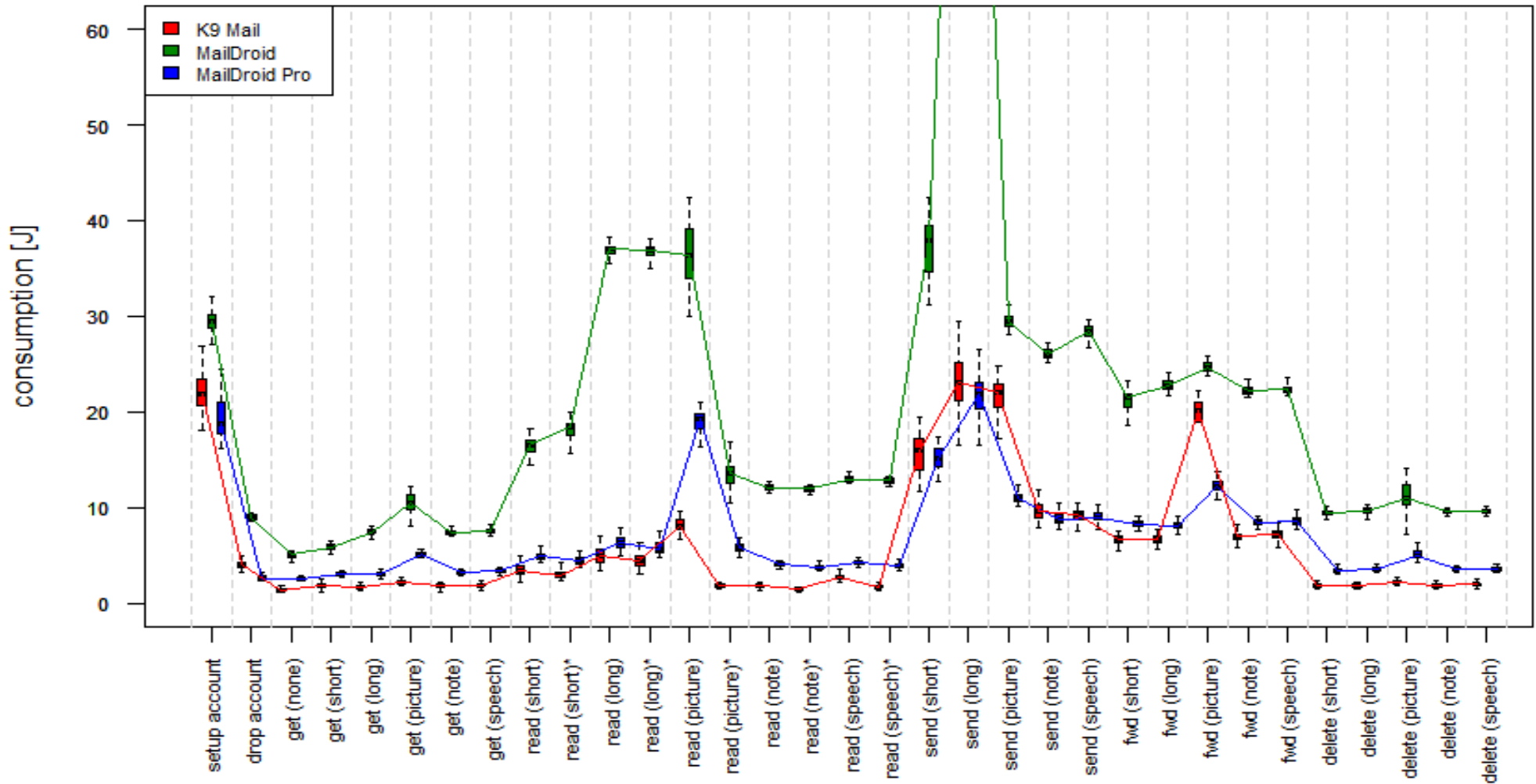


Interesting Issues with Web Browsers

- ▶ High variance in measurements (by high variance of response times)
 - But: comparable trends
- ▶ NineSky worst for small pages
 - ▶ But better for big images (because faster)
- ▶ Advertisement in EasyBrowser, DroidSurfing has negative influence only during long load times
- ▶ Different browsers are optimal for different use cases

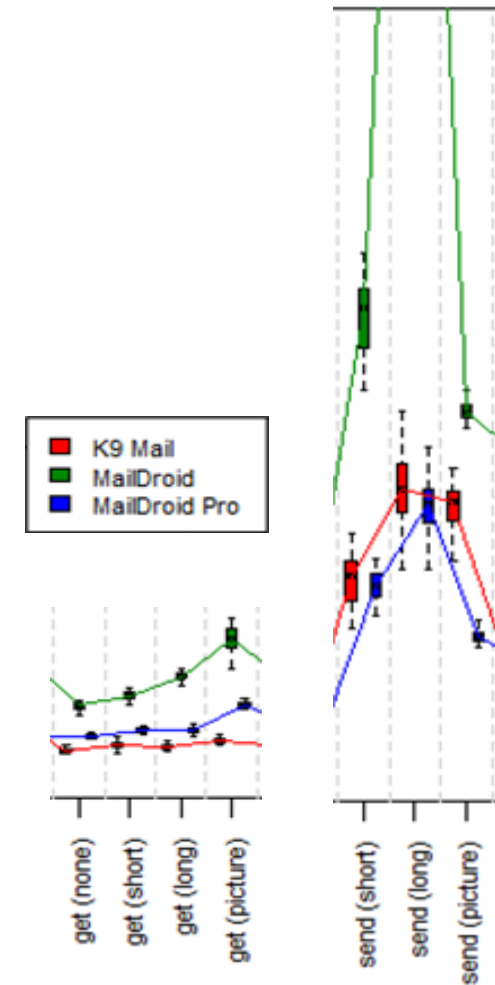


Median Mail Client Energy Consumption



Interesting Issues with Email Clients

- ▶ Low variance of measurements
- ▶ MailDroid worst for all scenarios
 - Reason: Advertisement
 - Negative influence grows for long scenarios
 - MailDroid pro & K9 Mail are similar
- ▶ Differences in energy consumption
- ▶ Avoid advertisement



12.1.3 Eclipse-Based Test Platforms

TPTP

TPTP Platform Project <http://www.eclipse.org/tptp/>

- Covers the common infrastructure in the areas of user interface, EMF based data models, data collection and communications control, remote execution environments and extension points

TPTP Monitoring Tools Project

- Collects, analyzes, aggregates and visualizes data that can be captured in the log and statistical models

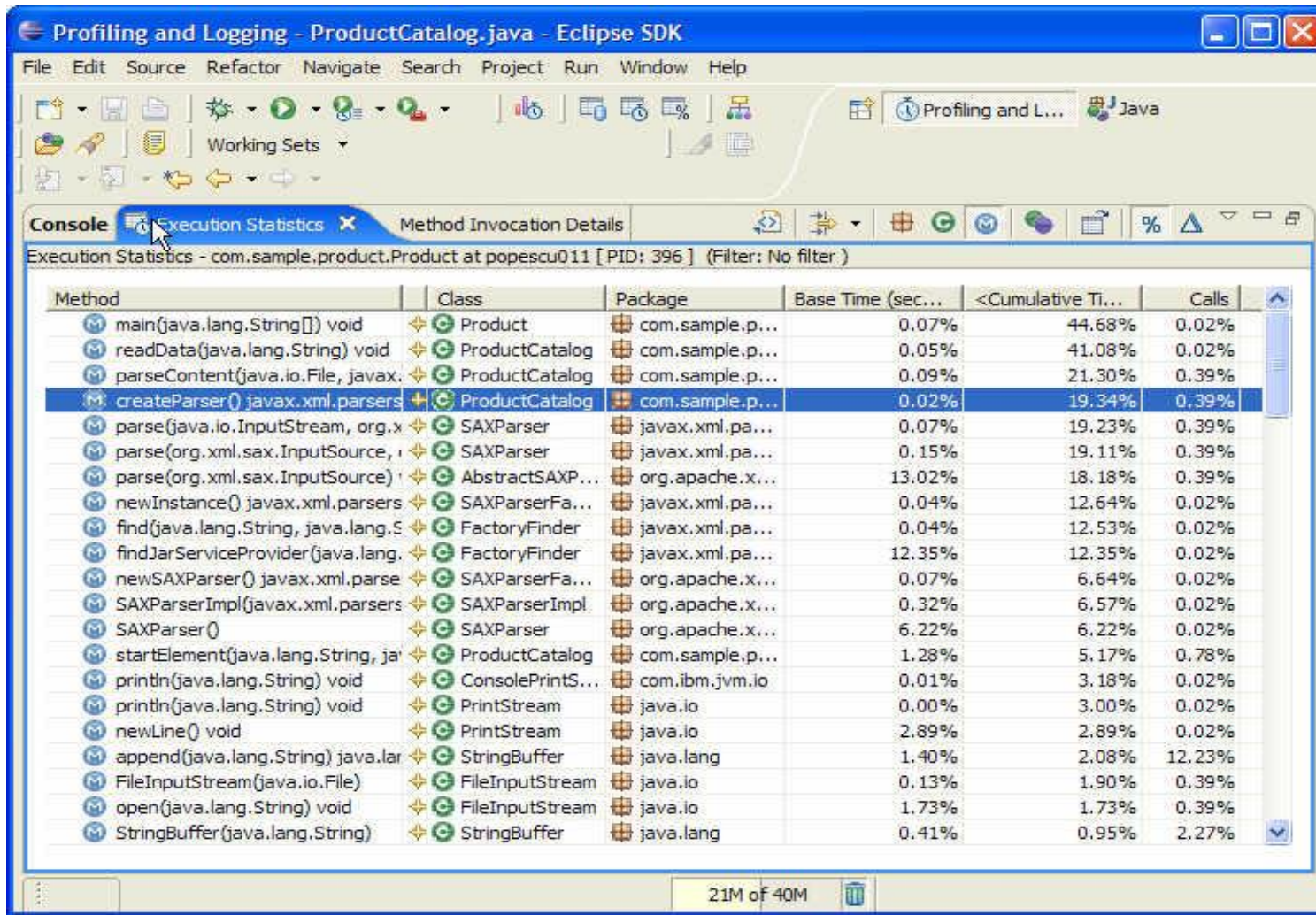
TPTP Testing Tools Project

- Provides specializations of the platform for testing and extensible tools for specific testing environments
- 3 test environments: JUnit, manual and URL testing

TPTP Tracing and Profiling Tools Project

- Extends the platform with specific data collection for Java and distributed applications that populate the common trace mode, also viewers and analysis services

TPTP Profiling Tool



Profiling and Logging - ProductCatalog.java - Eclipse SDK

File Edit Source Refactor Navigate Search Project Run Window Help

Working Sets

Console Execution Statistics Method Invocation Details

Execution Statistics - com.sample.product.Product at popescu011 [PID: 396] (Filter: No filter)

Method	Class	Package	Base Time (sec...)	<Cumulative Ti...	Calls
main(java.lang.String[]) void	Product	com.sample.p...	0.07%	44.68%	0.02%
readData(java.lang.String) void	ProductCatalog	com.sample.p...	0.05%	41.08%	0.02%
parseContent(java.io.File, javax...	ProductCatalog	com.sample.p...	0.09%	21.30%	0.39%
createParser() javax.xml.parsers	ProductCatalog	com.sample.p...	0.02%	19.34%	0.39%
parse(java.io.InputStream, org.x...	SAXParser	javax.xml.pa...	0.07%	19.23%	0.39%
parse(org.xml.sax.InputSource, i...	SAXParser	javax.xml.pa...	0.15%	19.11%	0.39%
parse(org.xml.sax.InputSource) i...	AbstractSAXP...	org.apache.x...	13.02%	18.18%	0.39%
newInstance() javax.xml.parsers	SAXParserFa...	javax.xml.pa...	0.04%	12.64%	0.02%
find(java.lang.String, java.lang.S...	FactoryFinder	javax.xml.pa...	0.04%	12.53%	0.02%
findJarServiceProvider(java.lang...	FactoryFinder	javax.xml.pa...	12.35%	12.35%	0.02%
newSAXParser() javax.xml.parse	SAXParserFa...	org.apache.x...	0.07%	6.64%	0.02%
SAXParserImpl(javax.xml.parsers	SAXParserImpl	org.apache.x...	0.32%	6.57%	0.02%
SAXParser()	SAXParser	org.apache.x...	6.22%	6.22%	0.02%
startElement(java.lang.String, ja...	ProductCatalog	com.sample.p...	1.28%	5.17%	0.78%
println(java.lang.String) void	ConsolePrintS...	com.ibm.jvm.io	0.01%	3.18%	0.02%
println(java.lang.String) void	PrintStream	java.io	0.00%	3.00%	0.02%
newLine() void	PrintStream	java.io	2.89%	2.89%	0.02%
append(java.lang.String) java.lar	StringBuffer	java.lang	1.40%	2.08%	12.23%
FileInputStream(java.io.File)	FileInputStream	java.io	0.13%	1.90%	0.39%
open(java.lang.String) void	FileInputStream	java.io	1.73%	1.73%	0.39%
StringBuffer(java.lang.String)	StringBuffer	java.lang	0.41%	0.95%	2.27%

21M of 40M

12.2 Requirements-Oriented Testing

- A requirements-oriented test environment allows for
 - Tracing test cases to requirement specifications
 - Measuring the maturity of test with regard to the acceptance tests

Test Environments

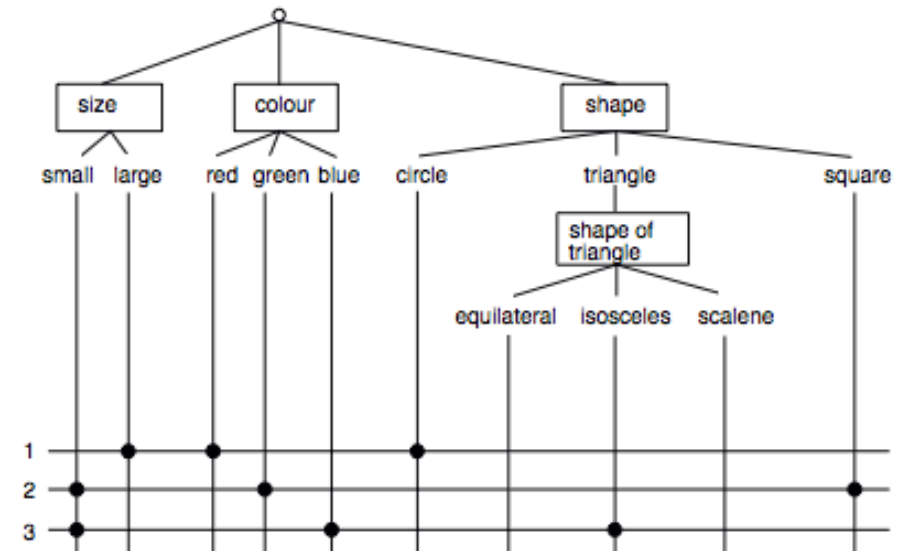
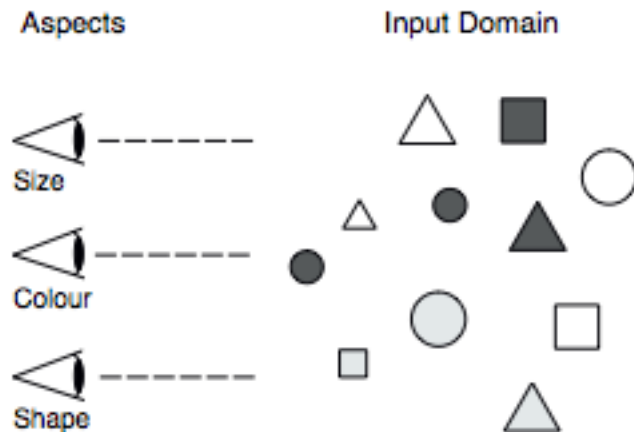
More: <https://www.testtoolreview.de/de/>

	Enterprise	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
Tracetronic ECUTest	Tracetronic, Dresden. For motor tests	www.tracetronic.de

12.2.1 Classification Tree Method and Tessy

Categories (Facets, Aspects) of the Test Case Data

- ▶ Test cases for testee objects and testee procedures are worked out in different categories (aspects, facets)
- ▶ Values of the parameters of testees are recursively divided into
 - ▶ intervals with one representant
 - ▶ Atomic value
- ▶ The values are grouped to test cases for a *test case table*

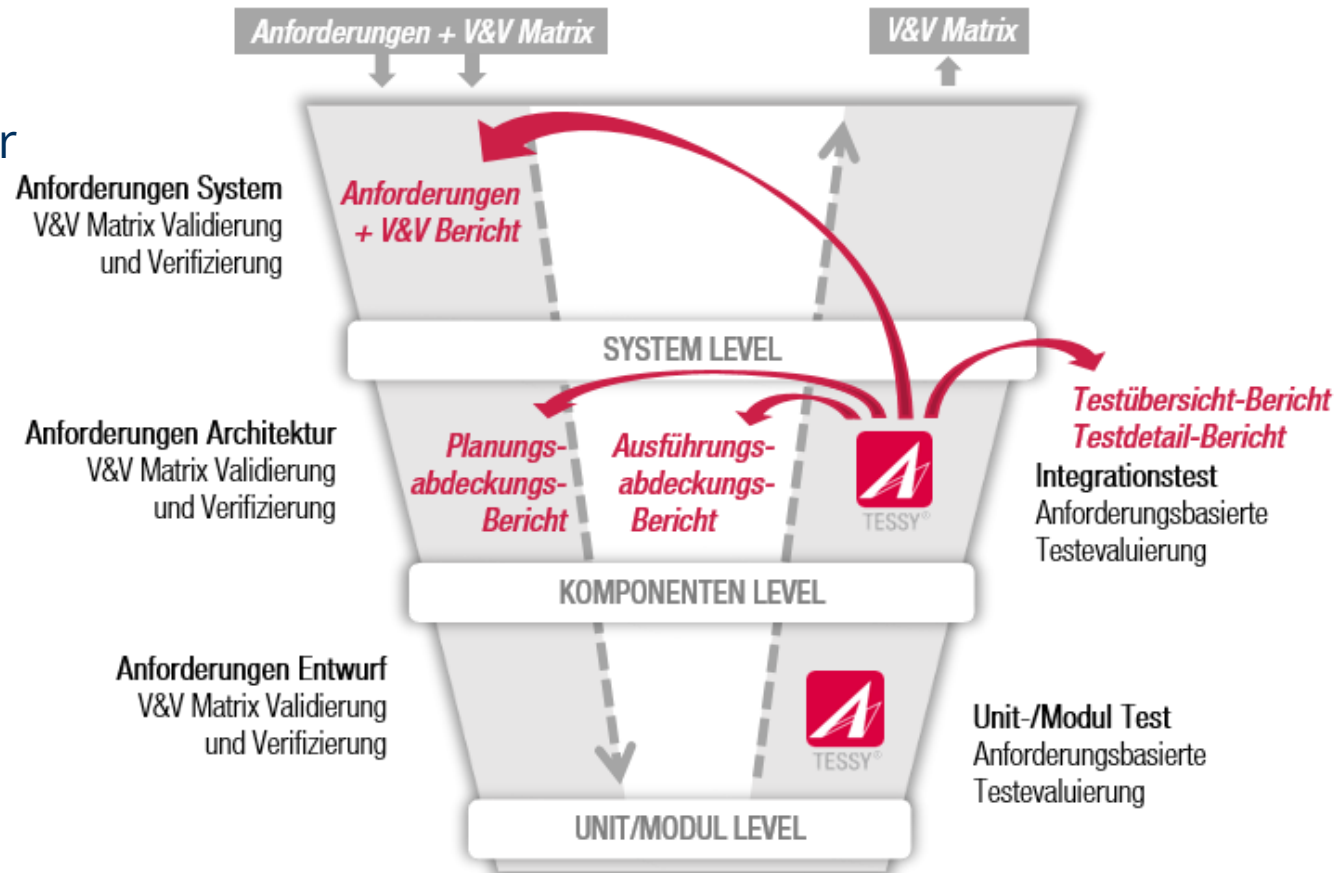


Advantages of Classification Tree Method

- ▶ http://www.hitex.com/fileadmin/pdf/products/tessy/white-papers/WP-TESSY-0102-CTE_e.pdf
- ▶ Division into categories reduces complexity of testing
 - Dimensional decomposition of parameter value space
 - Good visualization
- ▶ Representants of intervals should be chosen as boundary values of the intervals
- ▶ Coverage
 - ▶ A good combination of parameter values should cover the most important error cases
 - ▶ Generation of test cases

TESSY of HITEX

- ▶ Defining test cases with the classification tree method for regression tests
- ▶ Combination with coverage test



<http://www.hitex.com/>

<https://www.razorcat.com/de/tessy.html>

TESSY of HITEX

The screenshot displays the TESSY software interface for a project named 'Example-Project'. The main window is the 'Classification Tree Editor (CTE)', which shows a hierarchical tree structure for the 'MEAS_CalcAverage' component. The tree is divided into 'invalid' and 'valid' states. The 'valid' state contains a 'Recorded values (cRecords[])' node, which is further divided into 'Min.' and 'Max.' values, with a range of -128 to 127. The 'Count of recorded values (ucl.length)' node is also present, with a range of 0 to 5. The left panel shows the 'Test Project' tree structure, including 'AutoStartStop', 'Plant-Monitor', and 'MEAS' components. The bottom-left panel shows the 'Properties' window for the 'cRecords' parameter, indicating its type is 'char *' and its scope is 'parameter'. The right panel shows the 'Test Data of MEAS_CalcAverage' window, which contains a table of test data for the 'target_cRecords' parameter.

Parameter	Value
target_cRecords[0]	21
target_cRecords[1]	32
target_cRecords[2]	25
target_cRecords[3]	36
target_cRecords[4]	30

12.3.2 Imbus TestBench

imbus TestBench

Imbus TestBench is a test environment supporting

- Test planning
- Test analysis (connection of test cases to requirements)
- Test design
- Test automation (realization)
- Test metrics and reports

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

Test-Status eines Requirements (rot, gelb, grün)

The screenshot shows a software interface for requirements management. The title bar reads "Anforderungsverwaltung von Car Konfigurator (Version 2.1, Abnahmetest)".

Anforderungsbaum:

- CarKonfigurator - Version 1.1 (caliber)
 - 1. Business Requirements
 - Konfiguration zusammenstellen (Yellow)
 - Rabatt gewähren (Green)
 - automatische Rabatte (Green)
 - Händler gewährt Rabatt (Green)
 - 2. User Requirements
 - ständige Preisanzeige (Green)
 - keine erzwungene Bedienerfolge (Yellow)
 - 3. Functional Requirements
 - sofortige Preisberechnung (Red)
 - Quelle der Basisdaten (Green)
 - Import einer Datei (Green)
 - Import vom OEM-Host (Green)
 - 4. Design Requirements
 - gültige Konfiguration (Grey)
 - Eingabe der Basisdaten (Red)

Details Panel:

- 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

Menü ? - x

Ansicht >>

Details >>

✓

⊘ Mehr

Imbus

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>

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) 00:00:00.000

Aktuelle Durchführungszeit (DD:HH:MM:SS.SSS) 00:00:00.000

Liste der Anforderungen

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

12.4 Model-Driven Test Environment MATE

Model-Driven Testing



Model-driven testing (MDT, MBT) generates black-box test cases from models, e.g., statecharts, petrinets, activity diagrams, sequence diagrams

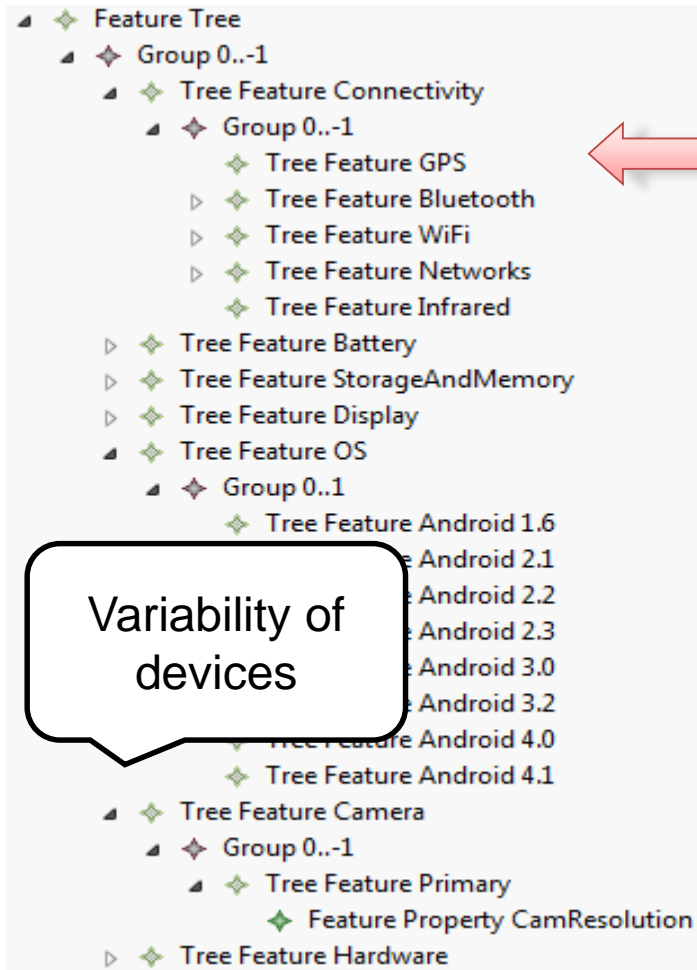


Problem 2:
Apps differ on different devices

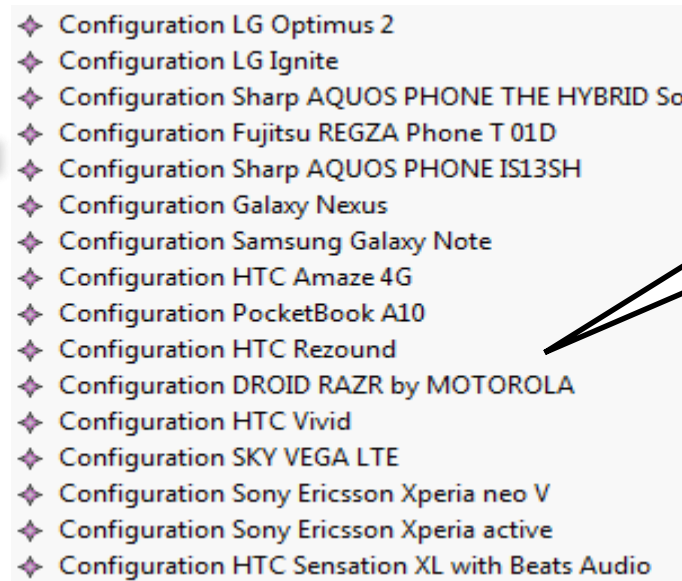
Problem 1:
Different platforms

Problem 3:
Apps are context-adaptive

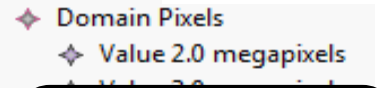
Plattform Management with Attributed Feature-Models (And-Or-Trees)



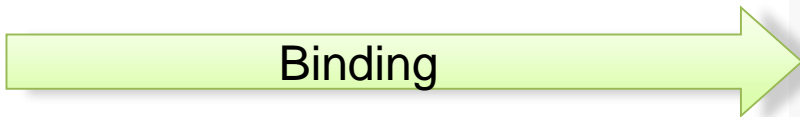
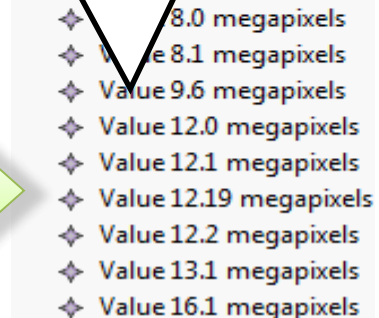
Variability of devices



Concrete devices



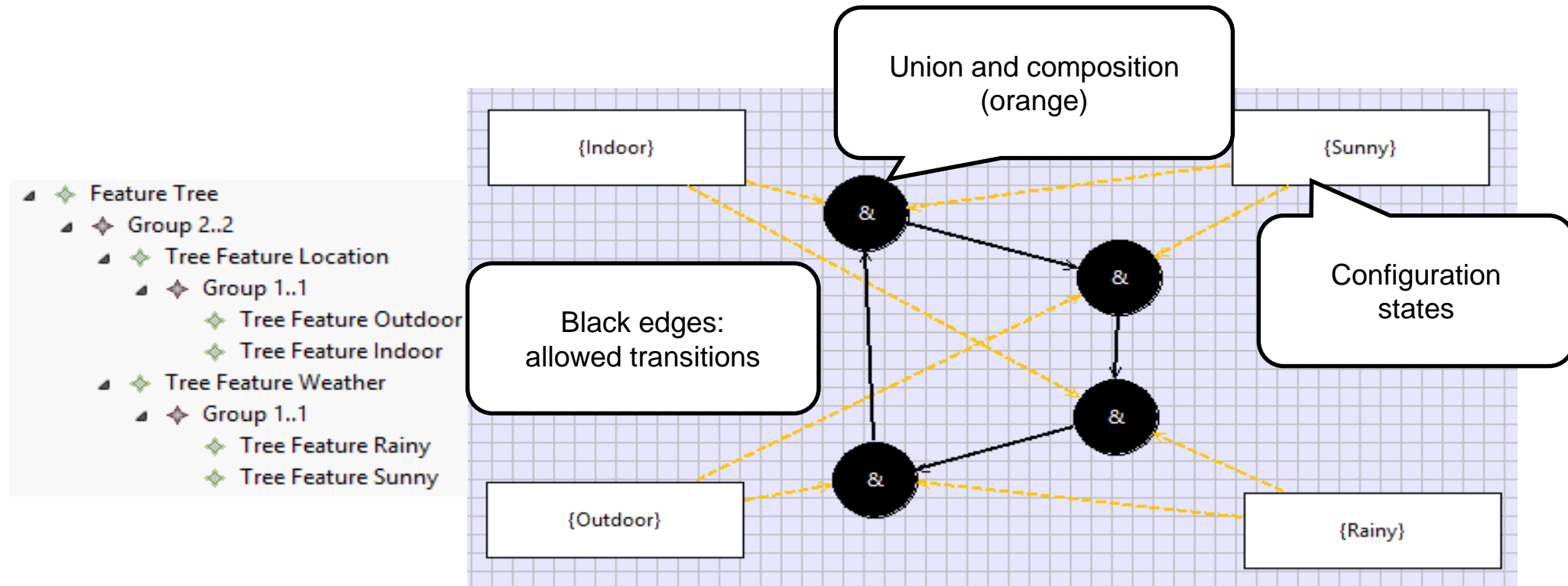
Finite Attribute-value-sets



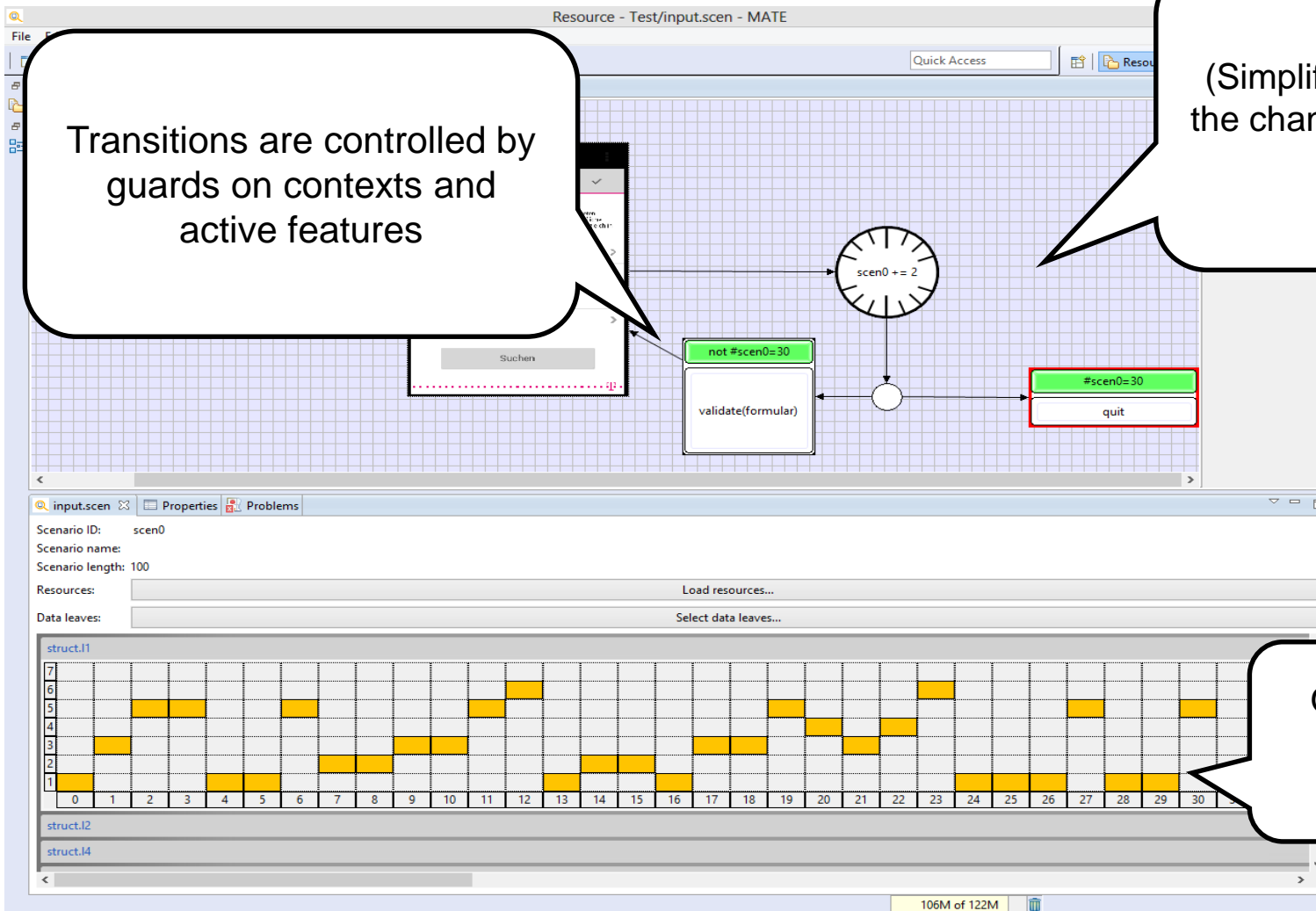
Variability in applications can be described by feature models

Dynamic Change of Features

- ▶ Features can be activated or deactivated at run time (**dynamic reconfiguration**)
 - ▶ Dynamic change of contexts based on a feature transition Petrinet
- ▶ Legal reconfigurations are described in a **operational configuration model (OCM)**



Specification of Legal Adaptations



Transitions are controlled by guards on contexts and active features

(Simplified) Timed Petrinet controls the changes of contexts in Scenarios or OCM alternatives

The MATE generator produces test cases by reachability analysis on the timed petrinet.

OCM alternative: Szenarios describe value changes of context data

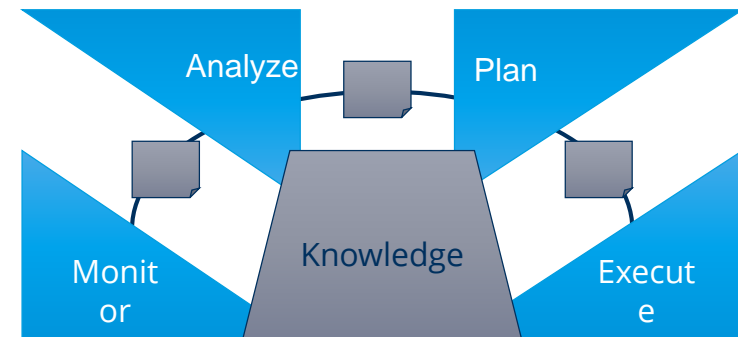
Test of SAS must be Model-Based

Self-adaptive Systems (SAS) reconfigure themselves

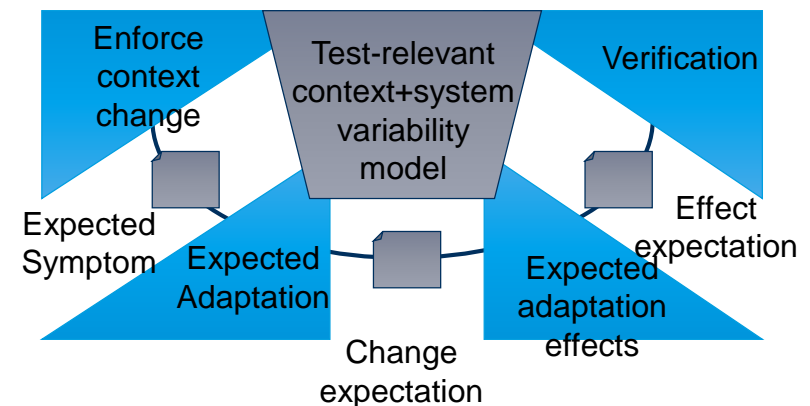
- at runtime
- according to requirements
- And dependent on the „context“, which may change over time.

In order to test SAS, one must

- stress the system with different contexts,
- alter contexts over time,
- model the system's expected adaptation
- and the expected effects of the adaptation on the system's behavior.



„counter feedback loop“ (CFL)



Automation: Coping with Complexity

Solution A: Model-driven Testing (MDT)

- Black-box testing with automatic test design (in contrast to automatic test execution)
- Test data, cases, and expected outcomes are generated from models
- The models' expressiveness hides complexity
- Adequacy criteria control generation

Solution B: SAS in the loop (ITL Simulation)

- MDT models are executed while the simulation state is compared to the real system one's for verification
- ITL is more explorative as only one path through state space is considered during a single simulation
- Enables testing reactions on non-controllable events (physical events that cannot be enforced, e.g. imprecise navigation of robots)

MATE testing a self-adaptive production system transport system with robots

The image displays two windows from the MATE (Model-based Adaptivity Test Environment) software. The left window, titled "Plant Overview - Operating mode - 'Demo-01'", shows a complex network of nodes and edges representing a production system. Nodes include "Goods in north 01", "Storage 01", "Storage 02", "Working station 01", "Working station 02", "Working station 03", and "Goods out 01". Edges represent transitions between these states, with labels like "Point-0027" and "Recharge 01". The right window, titled "Resource - Test.gg - Model-based Adaptivity Test Environment", shows a Petri net graph. The graph consists of several nodes: three yellow rectangular nodes labeled "produce('P1')", "produce('P2')", and "sleep(1000); produce('P3')"; two blue oval nodes labeled "gui2450244258190865810" and "gui813787..."; and one blue diamond node labeled "<2>gui245537...". The graph is connected by directed edges with weights. Below the graph, the "Progress" and "Simulation View" panels are visible. The "Simulation View" panel shows a list of simulation events, including "sleep(1000)", "produce('P3')", and "sleep(1000); produce('P3')". The "Properties" panel shows the current state of the simulation, including the number of tokens in various places and the current time.

The End