12. Validation Tools

**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](http://st.inf.tu-dresden.de)

Wintersemester 2017, 15.11.2017

1) **Code Analysis Tools**
   1) Programmanalyse

2) **Code-Centric Test Environments**
   1) Coverage-based test tools
   2) JouleUnit energy test framework
   3) Eclipse-based test tools

3) **Requirements-Driven Test Environments**
   1) Classification tree method and Tessy

4) **Model-Driven Test Environments**
   1) MATE
   2) Andere
For metrics, control-flow and value-flow analysis

12.1 CODE ANALYSIS TOOLS
Functionality of Kalimetrix Logiscope

- Different vendors: Telelogic North America Inc., Irvine, USA (Hersteller des Requirement Management Systems DOORS), Rational, IBM, since 2012 Kalimetrix
  - Windows, Linux
- Coverage testing
- Test delivers
  - Trace protocols
  - Untested branches
  - Visualization of control flow graphs and call graphs
- Computation of metrics
- Test definition:
  - Instrumentation of code
  - Graphic analysis of test results
  - Generation of test documentation
Logiscope Tools

- Quality Checker: definition and checking of software metrics
  - Quality characteristics

- Rule Checker:
  - Definition of programming guidelines and rules in Tcl or Perl
  - Check the rules of standards for regulation (MISRA)

- Test Checker: Coverage checker
  - Check acceptance of diverse standards such as IEC 61508-3, DO178B (Arial)
  - Generation of test reports

- Code Reducer:
  - Code clone detection

- Viewer: Visualizer for graphs
Sonargraph Metrics Tool

Hello2Morrow  [http://www.hello2morrow.com](http://www.hello2morrow.com)


Static and metrics-based analyses for the ASG:

- Architectural analysis (package dependencies, layering,...)
- Monitoring the compliance to architectural rules
- Finding “bad smells”, opportunities for refactoring

Trend computations for metrics

- 3D visualisation of results
- Specification of architecture in an architectural DSL
12.2 CODE-CENTRIC TEST-FRAMEWORKS
12.2.1 CODE COVERAGE TOOLS
# Control-Flow Oriented White-Box Test (Code Coverage)

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

<table>
<thead>
<tr>
<th>Coverage class</th>
<th>Technique</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>All defs</td>
<td>For all definitions (assignments) of variables: one path to a use has to be tested</td>
<td>Discover dead variable assignments (definitions)</td>
</tr>
<tr>
<td>All p-uses</td>
<td>For a definition (assignment) of a variable, test all uses <em>in predicates</em></td>
<td>Show the influence of the variable assignment to the control flow</td>
</tr>
<tr>
<td>All c-uses</td>
<td>For a definition (assignment) of a variable, test all uses <em>outside of predicates</em></td>
<td>Show the influence of the variable assignment to the data flow</td>
</tr>
</tbody>
</table>
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
Coverlipse: All-uses Data-Flow Coverage
Coverlipse: Problem Description of a Use of a Variable

Java - Computation.java - Eclipse Platform

public class Computation {
    public int add(int arg1, int arg2) {
        int result = arg1 + arg2; int meinInt = 0;
        // Problem description: Definition of the variable result
    }

<table>
<thead>
<tr>
<th>Message</th>
<th>Line</th>
<th>Covered Uses</th>
<th>Uncovered Uses</th>
<th>Ressource</th>
</tr>
</thead>
<tbody>
<tr>
<td>Definition of the variable arg1</td>
<td>12</td>
<td>13 15</td>
<td></td>
<td>Computation.java</td>
</tr>
<tr>
<td>Definition of the variable arg2</td>
<td>12</td>
<td>13</td>
<td></td>
<td>Computation.java</td>
</tr>
<tr>
<td>Definition of the variable meinInt</td>
<td>13</td>
<td>19</td>
<td></td>
<td>Computation.java</td>
</tr>
<tr>
<td>Definition of the variable result</td>
<td>14</td>
<td>18</td>
<td></td>
<td>Computation.java</td>
</tr>
<tr>
<td>Definition of the variable result2</td>
<td>14</td>
<td>18</td>
<td></td>
<td>Computation.java</td>
</tr>
<tr>
<td>Definition of the variable result3</td>
<td>15</td>
<td>18</td>
<td></td>
<td>Computation.java</td>
</tr>
</tbody>
</table>
Coverlipse: all-uses Coverage
12.2.2 CODE-CENTRIC ENERGY TEST-FRAMEWORKS

- JouleUnit (courtesy Claas Wilke)
- YouTube-Video: http://is.gd/energyLabel
  http://www.qualitune.org/
  http://www.jouleunit.org/
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

<table>
<thead>
<tr>
<th>Use Case</th>
<th>Time [s]</th>
<th>Rate [W]</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>12</td>
<td>3.5</td>
</tr>
<tr>
<td>B</td>
<td>8</td>
<td>3.2</td>
</tr>
<tr>
<td>C</td>
<td>10</td>
<td>3.2</td>
</tr>
</tbody>
</table>
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)
Compare „Similar“ Apps

Definition of benchmarks

- Web browsing
- Emailing

Apps:

- EasyBrowser, DroidSurfing, NineSky
- K9 Mail, MailDroid, MailDroid Pro
Median Web Browser Energy Consumption

- Easy Browser
- DroidSurfing
- NineSky

Consumption [J]
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

- K9 Mail
- MailDroid
- MailDroid Pro

consumption [J]
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
12.2.3 ECLIPSE-BASED TEST PLATFORMS
Eclipse Supports Many Test Platform Plugins

- **Hyades** [www.eclipse.org/test-and-performance](http://www.eclipse.org/test-and-performance)
  - Test Capture-Replay framework for web and other applications
  - Http requests can be recorded, generated into JUnit test case classes, afterwards replayed
  - Uses UTP to specify test cases
  - A Remove-Access-Controller mediates between Eclipse and the SUT
  - Test data can be stored in data pools
  - Log-file analysis based on the Common-Base-Event format of IBM

- **Solex** http proxy logger [www.sf.net/projects/solex](http://www.sf.net/projects/solex)

- **Scapa stress test** [www.scapatech.com](http://www.scapatech.com)

- **HttpUnit, htmlUnit** extensions of JUnit for test of web applications
  - [httpunit.sf.net](http://httpunit.sf.net)
  - [htmlunit.sf.net](http://htmlunit.sf.net)
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
Test Plugins for Eclipse


<table>
<thead>
<tr>
<th>Source</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coverlipse</td>
<td><a href="https://coverlipse.sourceforge.net">coverlipse.sourceforge.net</a></td>
</tr>
<tr>
<td>EclEmma</td>
<td><a href="http://eclemma.org/userdoc/coverageview.html">http://eclemma.org/userdoc/coverageview.html</a></td>
</tr>
</tbody>
</table>

[https://confluence.atlassian.com/display/CLOVER/Comparison+of+code+coverage+tools](https://confluence.atlassian.com/display/CLOVER/Comparison+of+code+coverage+tools)
A requirements-oriented test environment allows for
- Tracing test cases to requirement specifications
- Measuring the maturity of test with regard to the acceptance tests

12.3 REQUIREMENTS-ORIENTED TESTING
## Test Environments

<table>
<thead>
<tr>
<th>Enterprise</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>SilkTest</td>
<td>Segue Software</td>
</tr>
<tr>
<td>TestBench</td>
<td>Imbus</td>
</tr>
<tr>
<td>Cantata++</td>
<td>IPL, Bath, UK</td>
</tr>
<tr>
<td>ClickTracks</td>
<td>ClickTracks Analytics, Inc.CA</td>
</tr>
<tr>
<td>Tracetronic</td>
<td>Tracetronic, Dresden.</td>
</tr>
<tr>
<td>ECU Test</td>
<td>For motor tests</td>
</tr>
</tbody>
</table>

More: https://www.testtoolreview.de/de/
12.3.1 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

![Diagram](image)
Advantages of Classification Tree Method

- 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
12.3.2 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)

CarKonfigurator - Version 2.1 (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

Name: Händler gewährt Rabatt
ID: WHY162
Version: 1.1
Eigentümer:
Status: Review Complete
Priorität: Essential
Test-Status: Getestet PASS
12.4 MODEL-DRIVEN TEST ENVIRONMENT MATE

Georg.Pueschel@tu-dresden.de
Model-Driven Testing

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

**Problem 1:** Different platforms

**Problem 2:** Apps differ on different devices

**Problem 3:** Apps are context-adaptive
Plattform Management with Attributed Feature-Models (And-Or-Trees)

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

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

Transitions are controlled by guards on contexts and active features.

OCM alternative: Szenarios describe value changes of context data.

The MATE generator produces test cases by reachability analysis on the timed petrinet.
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)
Model-based Adaptivity Test Environment (MATE)

- implements both MDT and ITL for SAS
- provides SAS-specific test metamodels, interpretors, tooling, and a test adaptor framework

**Context & System variability**

Weather

- @illuminaton:[0..100]
- @wind:[0..12]
- @rainfall:[0..100]

Safety

- Cautious
- Unbound

**Adaptation Mode**

- Camera-Profile
- ShortExposure
- LongExposure

@wind > 4 and @rainfall > 20 ⇒ Cautious
@illuminaton < 60 ⇒ LongExposure

**Context dynamics**

**Adaptation dynamics**

- ShortExposure + LongExposure
- LongExposure + ShortExposure

**Adaptation effects on behavior**

- LowIllumination
- HighIllumination

B

- @wind := 10
- 5/@wind := 2
- 3/@wind := 4

C

- @wind := 0
- 4/@wind := 0

D

- alterCt(x());
- assertTrue(onGround());

E

- provideWaypoint();
- assertTrue(tookPhotos() == &childOf(CameraProfile));

F

- waitForBatteryLoading();
- assertTrue(batteryFull());

A

- waitForDrone();
- tearDown();
- assertTrue(batteryFull());
MATE testing a self-adaptive production system transport system with robots
The End