Test Oracles and Test Script Generation in Combinatorial Testing

Peter M. Kruse
Berner & Mattner Systemtechnik GmbH
Berlin, Germany
Overview

• Classification Tree Method
• Expected results
• Executable test scripts
Combinatorial Testing

Background

- **Combinatorial Interaction Testing (CIT)** is a black-box testing technique that samples inputs, configurations, and parameters and combines them in a systematic fashion.


- Coverage Criteria: Minimum, Maximum, Pairwise, N-Wise
  - Pairwise NP Complete. Yu Lei, Kuo-Chung Tai. In-parameter order: a test generation strategy for pairwise testing, 1998

- Constraints. Myra B. Cohen, Matthew B. Dwyer, and Jiangfan Shi. Interaction testing of highly-configurable systems in the presence of...
Classification Tree Method

- Grochtmann/Grimm 1993, Daimler Research

Two Steps:

1. **1st Design Classification Tree**
   - One *Classification* per test aspect (Parameter)
   - One *Class* for each parameter value
   - Resulting in a Tree of Classifications

2. **2nd Compose Test Cases**
   - Can be automated using TESTONA tool
     (formerly *Classification Tree Editor*)
Example

Select a test object: decompose

- Test object: Database Management System
Example

Input domain

Select a **test object**: decompose

Determine **input data space**
Example

Input domain

Privileges
Operation
Access Method

Select a test object: decompose

Determine input data space

Identify relevant aspects (e.g. from specification)
Example

Classify the input data space into classes.
Example

Classify the input data space into classes

Combine classes into test cases

Input domain
Privileges | Operation | Access Method
---|---|---
Classify the input data space into classes
Combine classes into test cases
Test Oracle

• Howden (1978)
  
  *non-trivial challenge of deciding whether a test case has passed or failed*

• Categorization by Barr, Harman, McMinn, Shahbaz, and Yoo (2015)
  
  – non-automated
  – implicit
  – derived
  – specified
Non-automated Oracles
• Trivial case
• Mapping function \( f(t_q) = R_q \) unknown

Implicit Oracles
• Indirect evaluation (e.g. no exception)
• Mapping function \( R_q = R_1 = R_2 = \ldots = R_n \)

Derived Oracles
• A posteriori (e.g. Regression Test, back-to-back Test)
• Mapping function \( f(t_q) = R_q \) not needed

Specified Oracles
• Based on (formal) Specification
• Mapping function \( f(t_q) = R_q \) known
Non-automated Oracle Example

• Manual assignment for each test case
Implicit Oracle Example

• Monitor system

No exceptions at runtime, ...
Derived Oracle Example

- Run once, record results, assign to tests
Specified Oracle Example

\[ C_2 \leftrightarrow R_1 \]

\[ C_1 \land (C_3 \lor (C_4 \land \neg C_{10})) \leftrightarrow R_2 \]
Specified Oracle Example

Using constraints can be problematic

1. Makes computation of output more difficult, solver must be used
   
   TESTONA has build in Solver

2. Possible to write incomplete/inconsistent mappings (e.g. $C_1 \leftrightarrow R_1$ and $C_1 \leftrightarrow R_2$)

3. Allows non-determinism

Task of test designer
Test Script Generation

- Problem: How to execute test specification?
import org.junit.Assert.*;
public class Example {

@Test
public void %TEST% throws Exception {

  setEnabled(%CLASS%);
  setDelay(%CLASS%);
  setUser( "%CLASS%" );
  getValue(%CLASS%);
  setValue(%CLASS%);
  update();
  assertTrue(%CLASS%);

  Include these per mark in test case

  Include this per test case

  Include this once at start

}
import org.junit.Assert.*;
public class Example {

@Test
public void %TEST% throws Exception {

    setEnabled(%CLASS%);
    setDelay(%CLASS%);
    setUser("%CLASS%"),
    getStatus(),
    setValue(%CLASS%);
    update();
    assertTrue(%CLASS%);

    Include this once at start

    Include this per test case

    Include these per mark in test case

    import org.junit.Assert.*;
    public class Example {

        @Test
        void t1 throws Exception {
            setEnabled(false);
            setDelay(0);
            setUser("Chris");
            update();
            assertFalse(getEnabled());

        }

    }

}
Effort Considerations

• Instead of implementing each test case
• Implement each parameter

• Less effort
• Reusable

Formalization in the paper

To be evaluated large scale
Related Work

• Most work on CT focuses on the calculation of minimal size test suites, mostly pairwise.
• Some CT approaches consider test oracles.
• Reports on test implementation and test execution are limited.
Related Work – Oracles

Non-automated Oracles
• All approaches

Implicit Oracles
• AETG, ACTS

Derived Oracle
• ACTS

Specified Oracles
• AETG, PICT, ACTS, ATGT, FoCuS

“Oracle-free Testing”
IWCT’15
Related Work
Test Implementation

• Integrated Approach also for
  – Category Partition Method (as part of specification language)
  – FoCuS (using post-pressing with templates)

• External Solutions
  – AETG (use of perl script reported)
  – PICT
  – ACTS
  – ...

Test Oracles and Test Script Generation in Combinatorial Testing
Future Work

• Large scale evaluation
• Effort for different test script generation approaches with focus on test suite maintenance
  – costs of adapting existing test suites, test scripts, post-processors when parameters are added, modified or removed
• What about model import?
  – UML, Statecharts, ...
Conclusion

• Post-processing is avoided in our approach
• Test implementation allows for the direct execution of combinatorial test suites
• Independent of different SUT types
• Several kinds of oracles are supported

Peter M. Kruse
Berner & Mattner Systemtechnik GmbH

Test Oracles and Test Script Generation in Combinatorial Testing