BERNER & MATTNER
AN ASSYSTEM COMPANY

# 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 blac g technique that samples inputs, configurati combines them in a systematic fashio
- Creating functional tests de
  Thomas J. Ostrand and Marc J. Balcer. T
- Coverage C
- D
  g
  teraction test coverage, 2001
-
  d Jiangfan Shi. Interaction testing of highly-configurable systems in the presence of

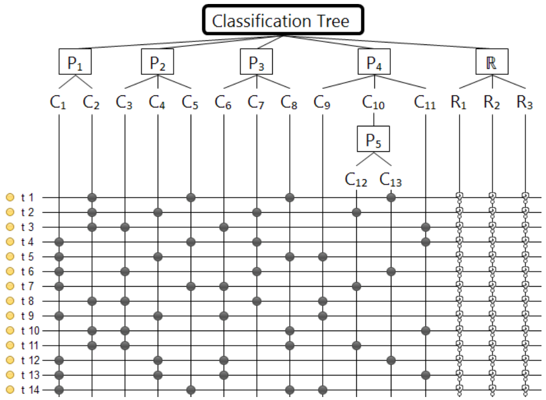
We can skip that here …

# Classification Tree Method

- Grochtmann/Grimm 1993, Daimler Research

Two Steps:

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




- 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

BERNER & MATTNER
AN ASSYSTEM COMPANY

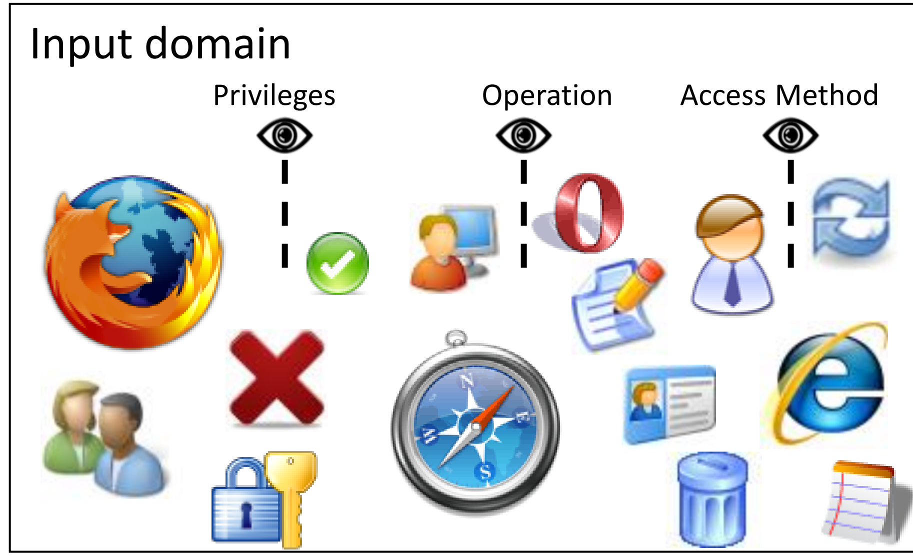
# Example

Input domain

Select a **test object**: decompose

Determine **input data space**

# Example

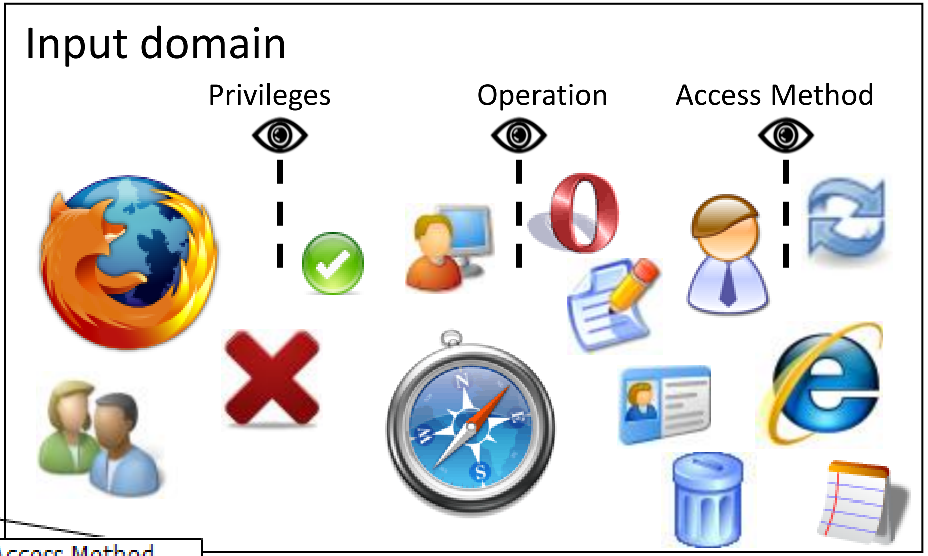


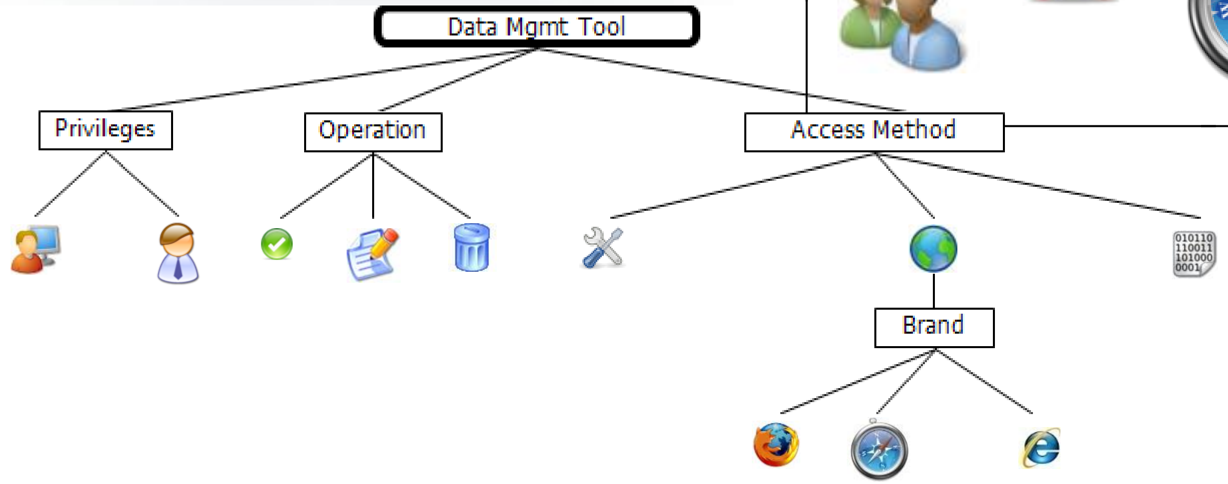

Select a **test object**: decompose

Determine **input data space**

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

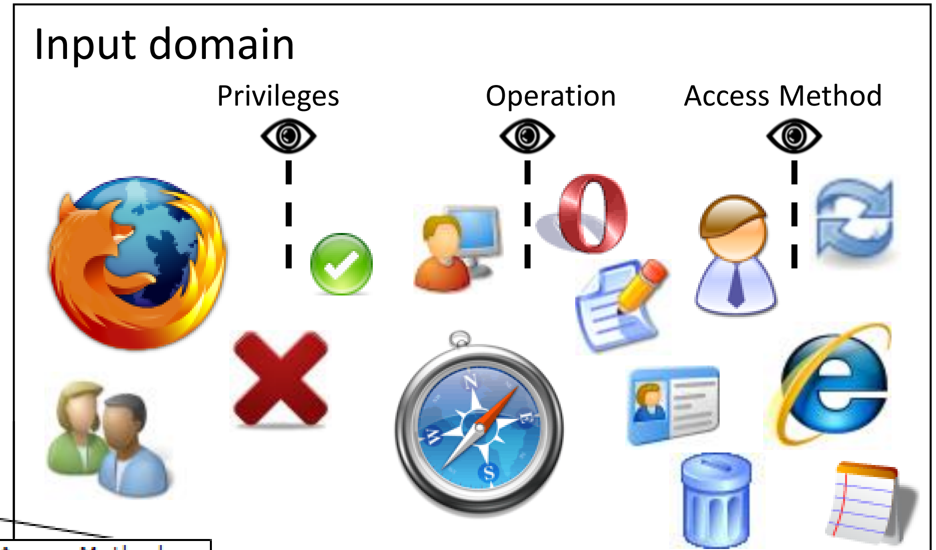
# Example

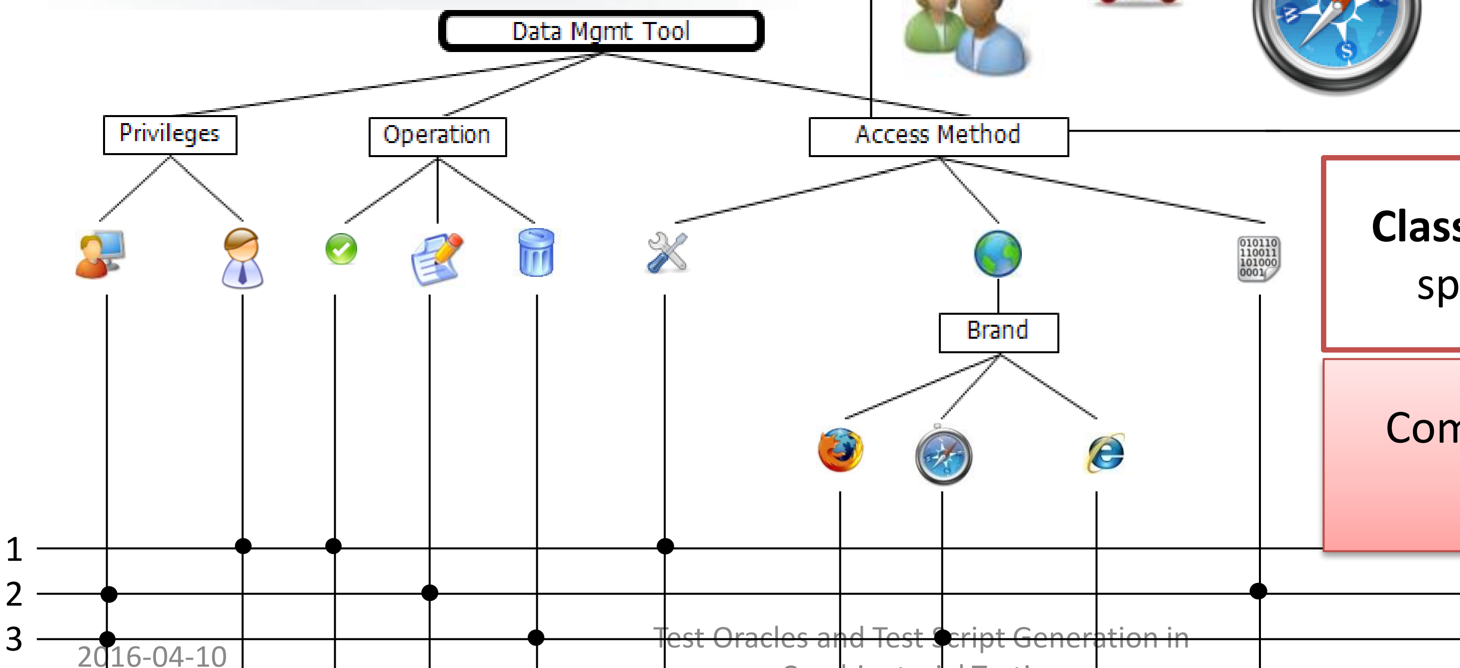



**Classify** the input data space into classes

# Example




**Classify** the input data space into classes

Combine classes into **test cases**

# 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

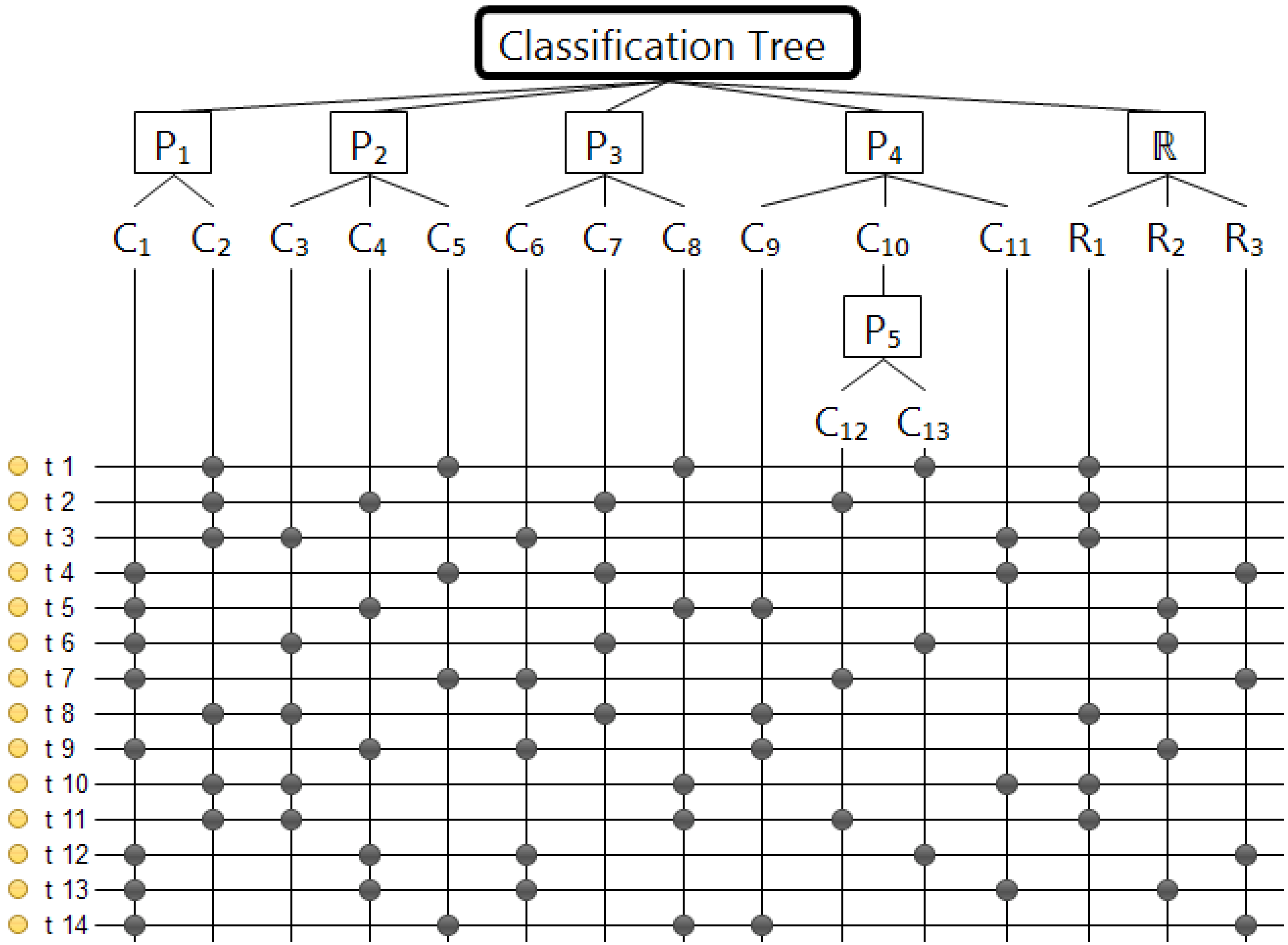
- A posteriori (e.g. Regression Test, back-to-back Test)
- Mapping function $f(t_q) = R_q$ not needed

## Specified Oracles

- Based on (formal) Specification
- Mapping function $f(t_q) = R_q$ known

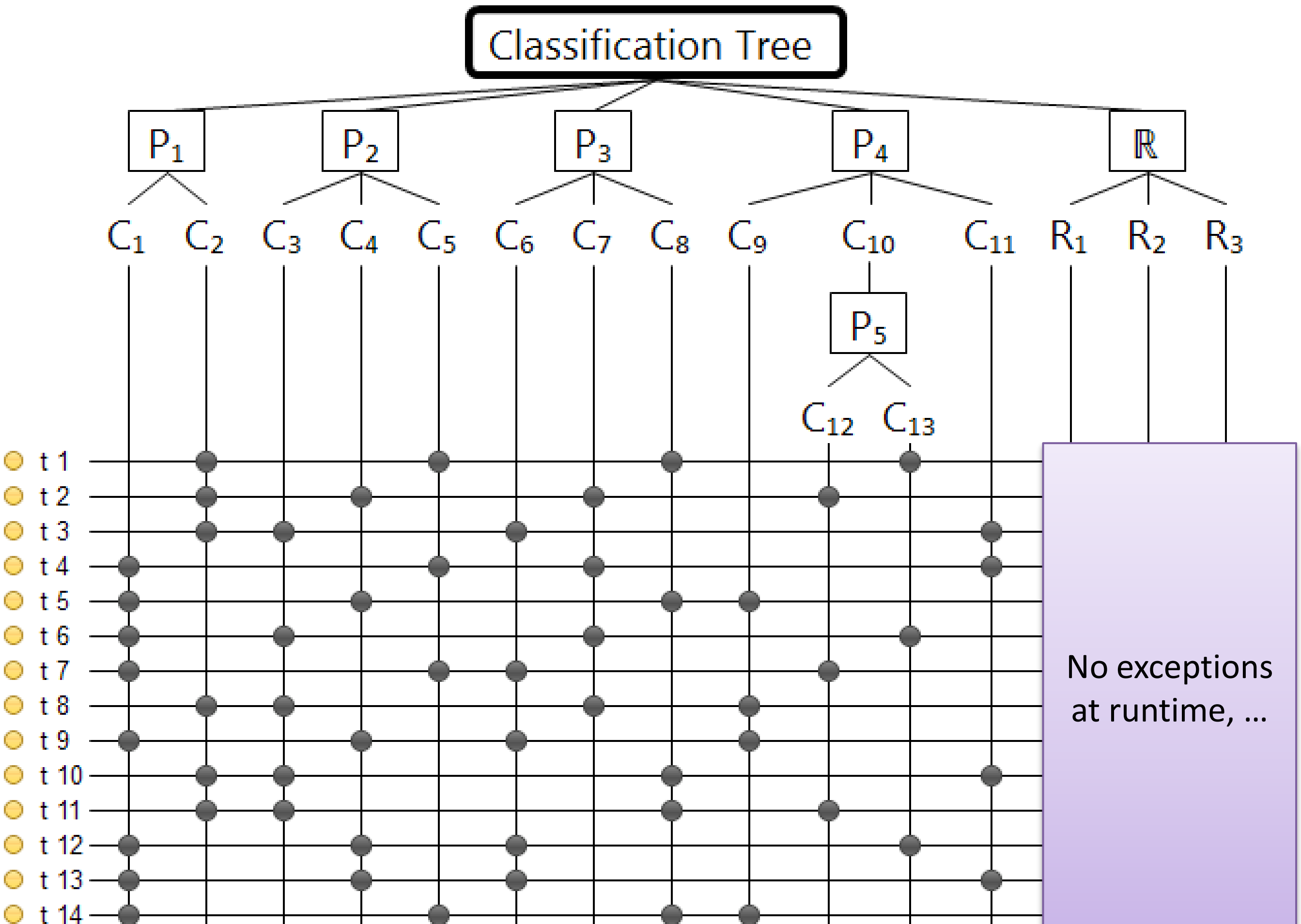
# Non-automated Oracle Example

- Manual assignment for each test case



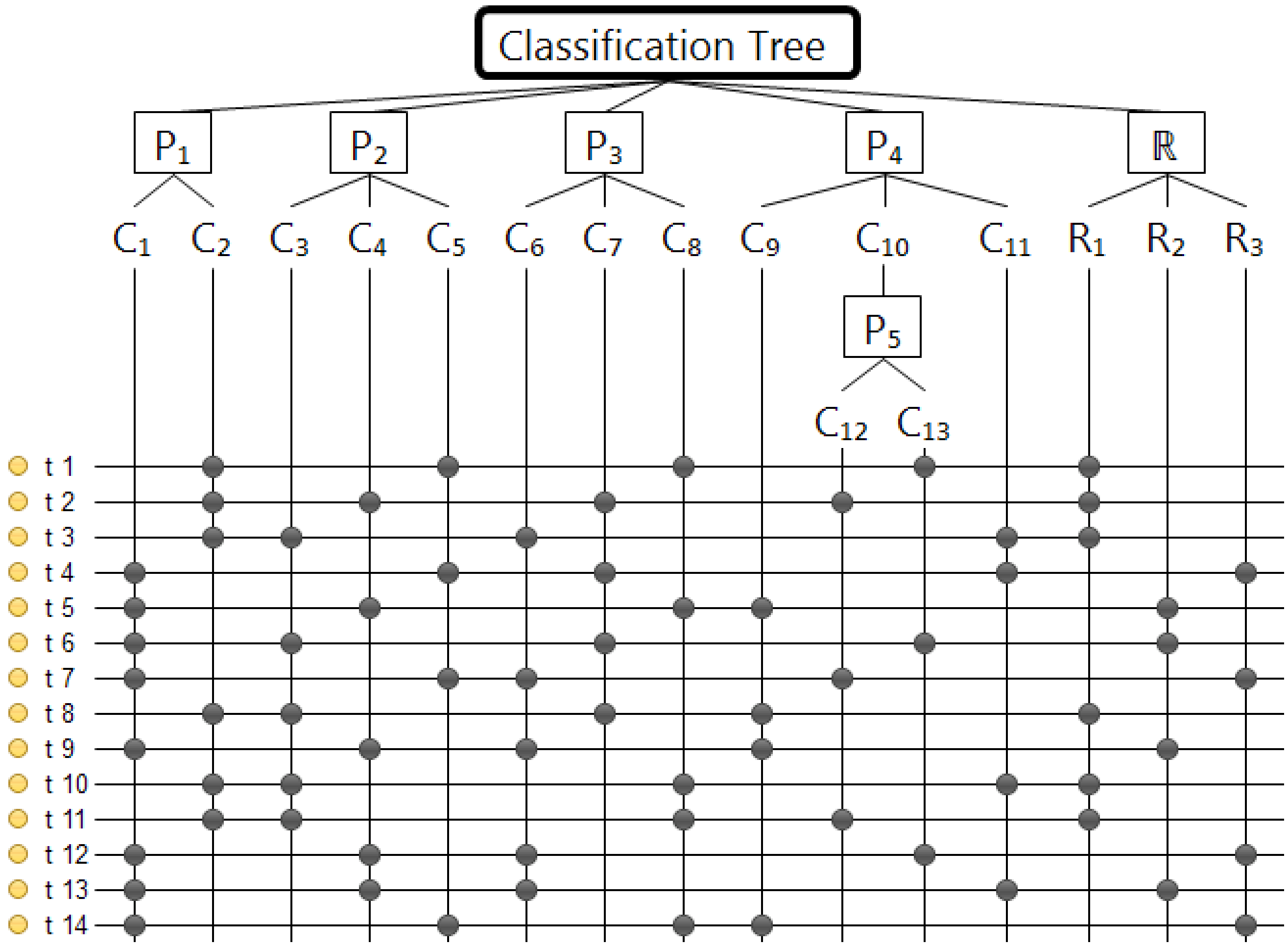
# Implicit Oracle Example

- Monitor system



# Derived Oracle Example

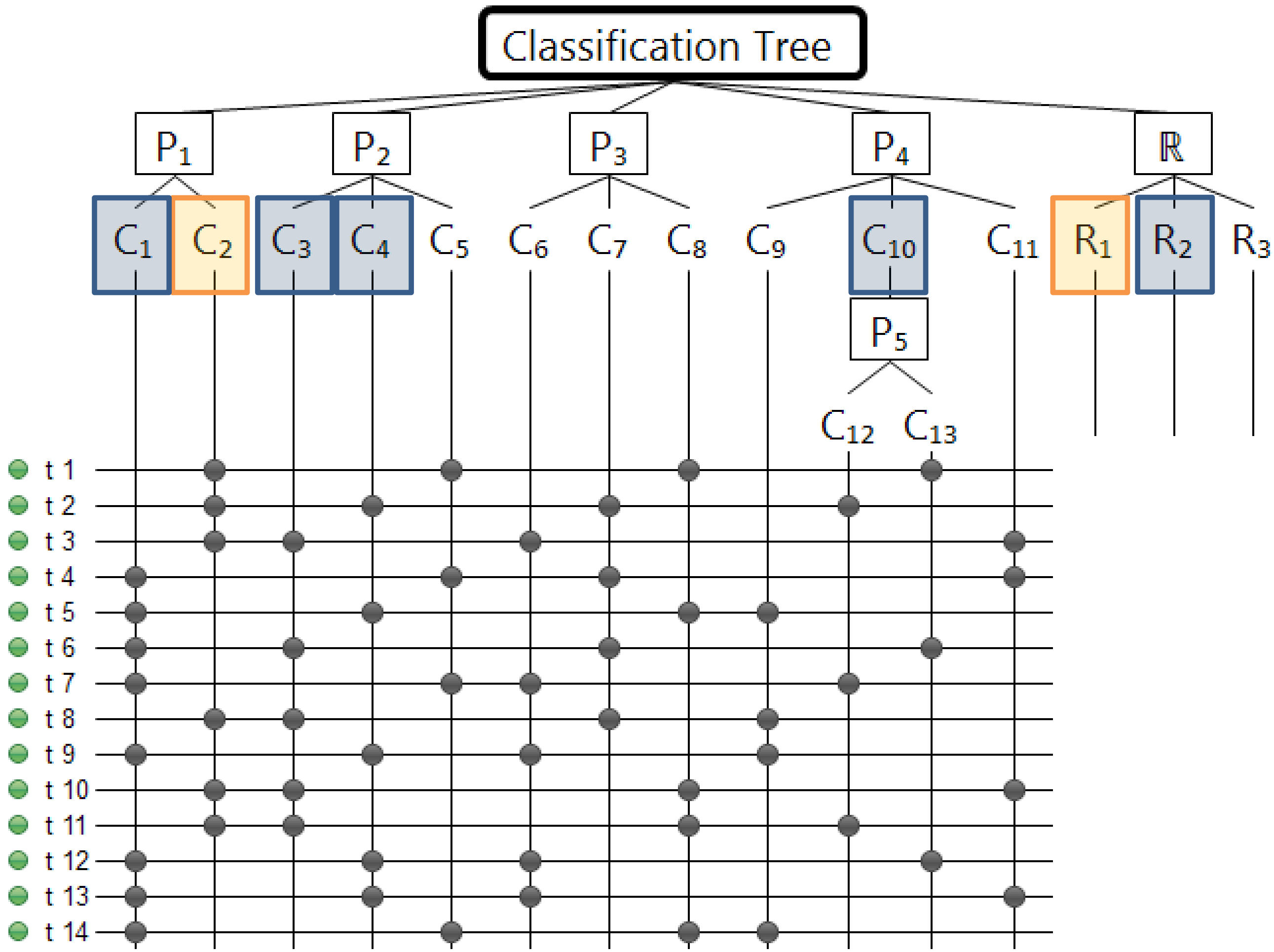
- Run once, record results, assign to tests



# Specified Oracle Example

$C_2 \leftrightarrow R_1$

$C_1 \wedge ( C_3 \vee (C_4 \wedge \neg C_{10})) \leftrightarrow R_2$



# Specified Oracle Example

Using constraints can be problematic

1. Makes computation of output more difficult, solver must be used

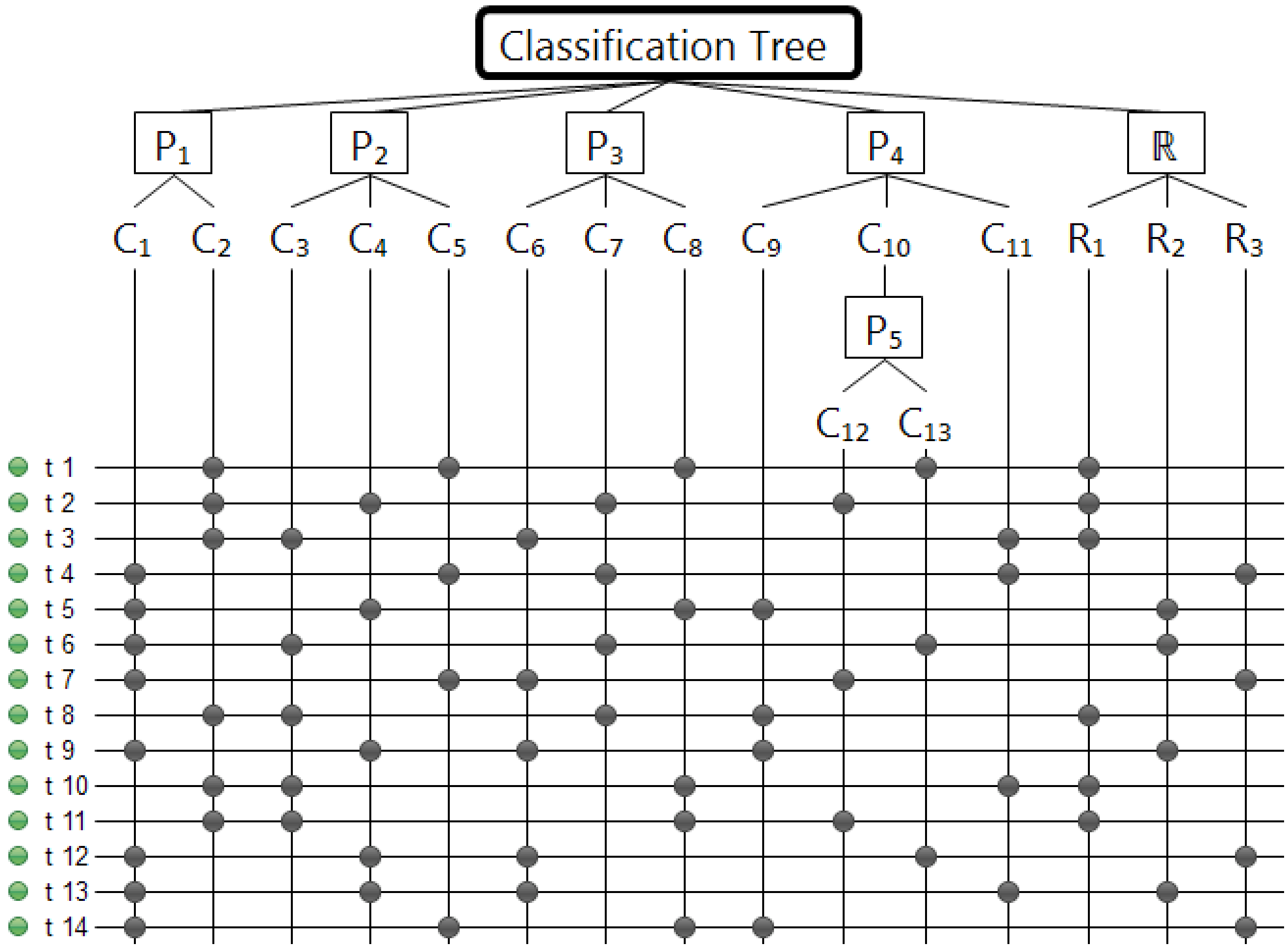
TESTONA has build in Solver

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?

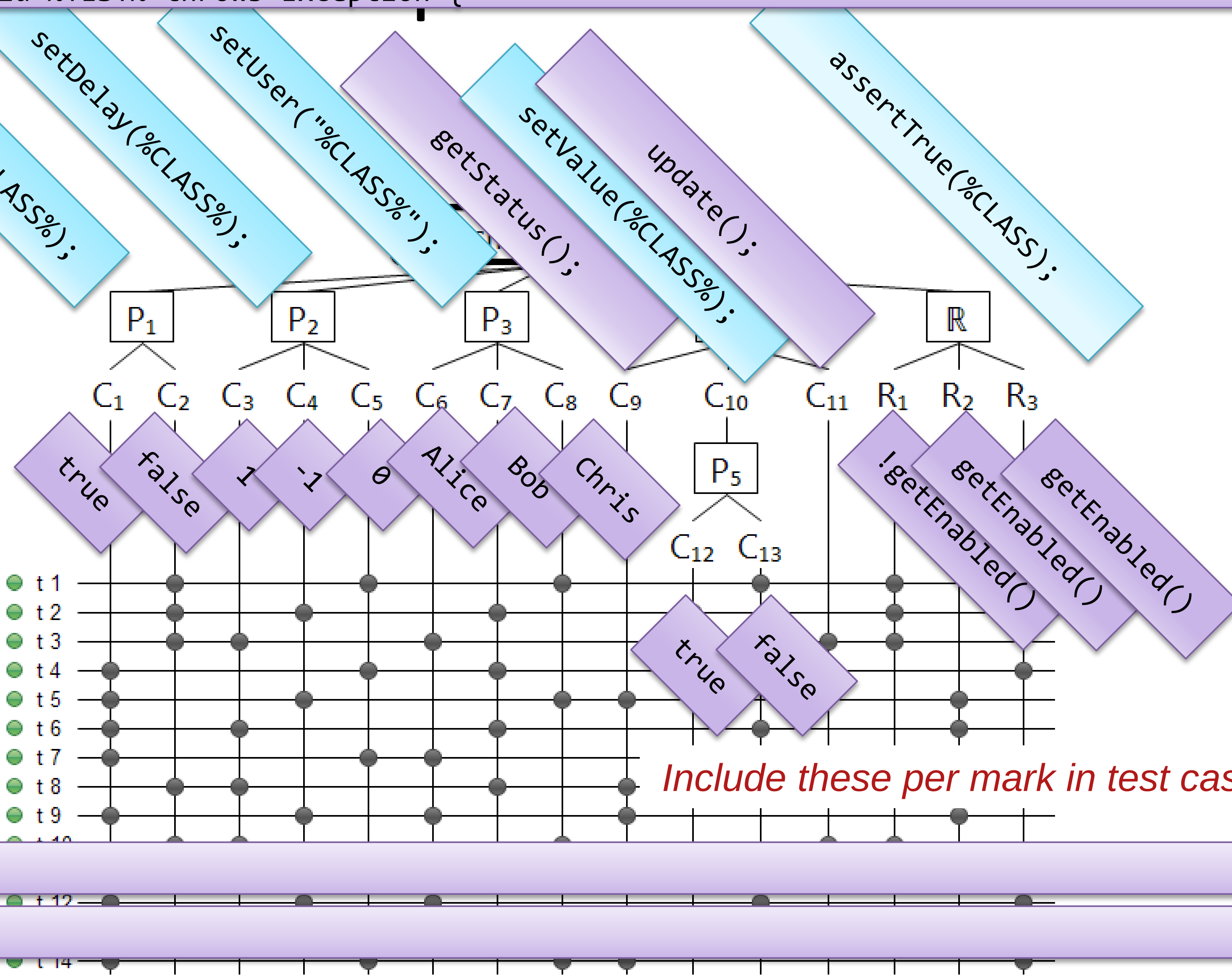


```
import org.junit.Assert.*;
public class Example {
```

*Include this once at start*

```
    @Test
    public void %TEST% throws Exception {
```

*Include this per test case*




*Include these per mark in test case*

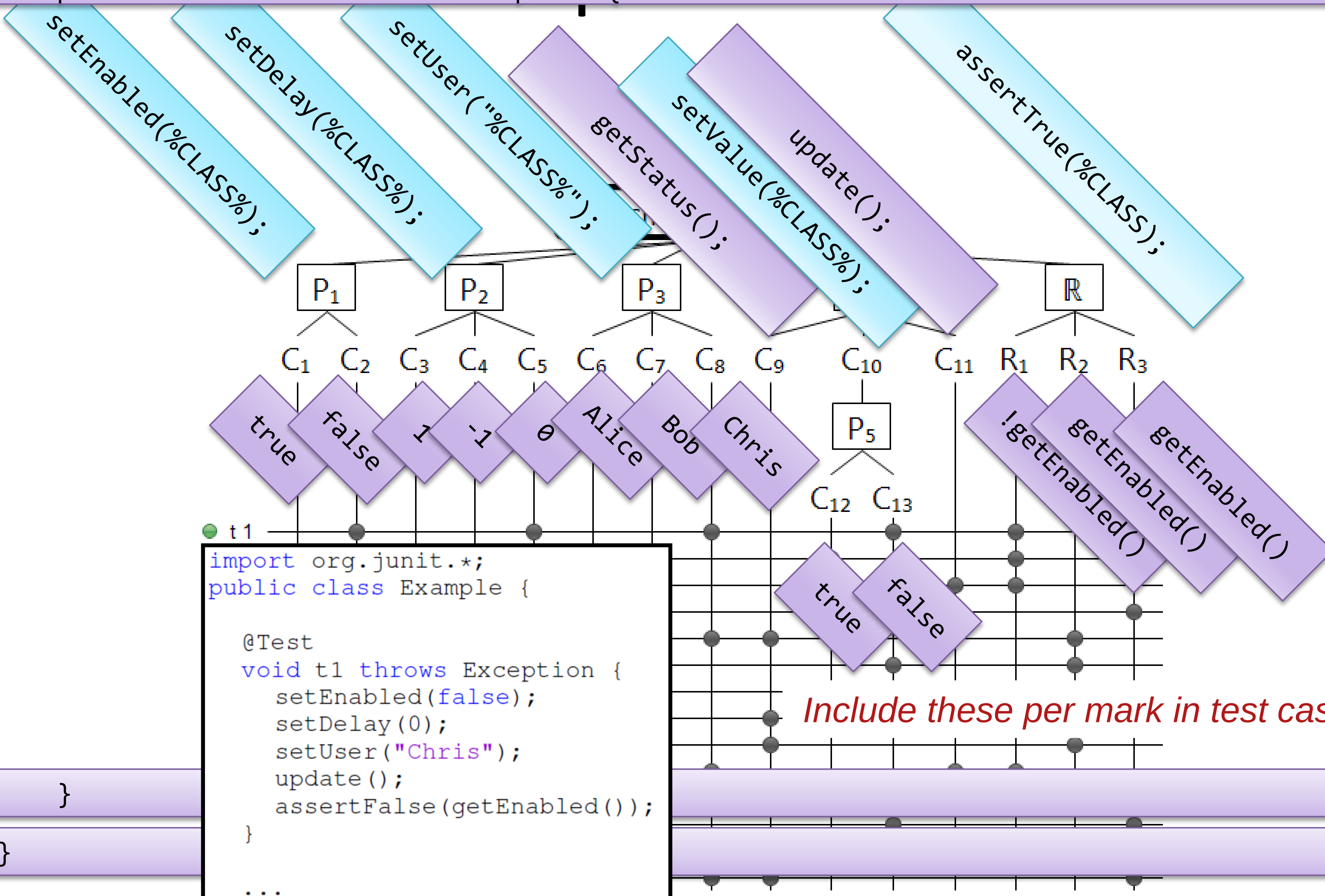
```
    }
}
```

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

*Include this once at start*

```
    @Test
    public void %TEST% throws Exception {
```

*Include this per test case*




*Include these per mark in test case*

```
import org.junit.*;
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

"Oracle-free Testing"
IWCT'15

## Specified Oracles

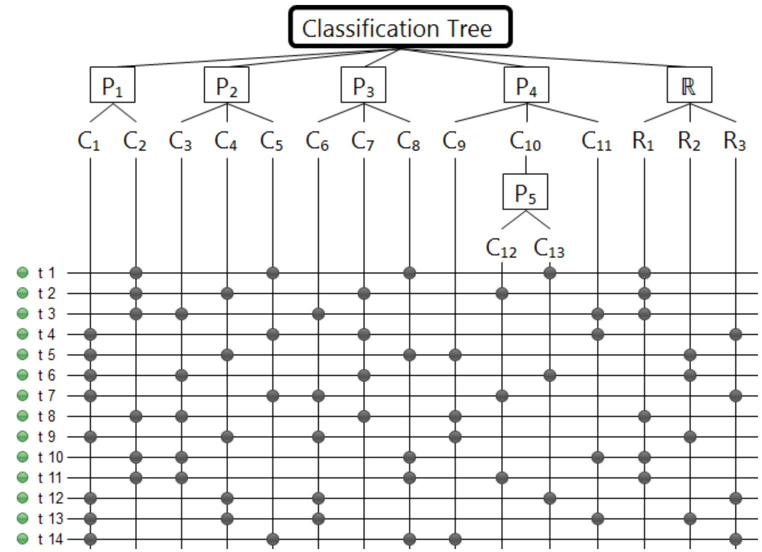
- AETG, PICT, ACTS, ATGT, FoCuS

# 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
  - …

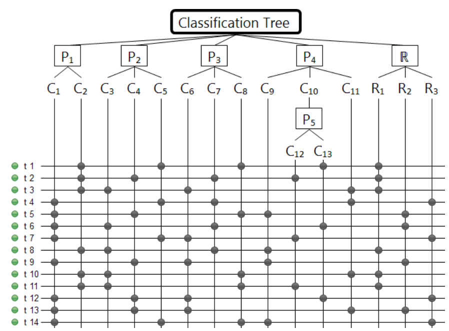
# Future Work




- 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




```java
import org.junit.*;
public class Example {

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

  ...

}
```

- Independent of different SUT types
- Several kinds of oracles are supported