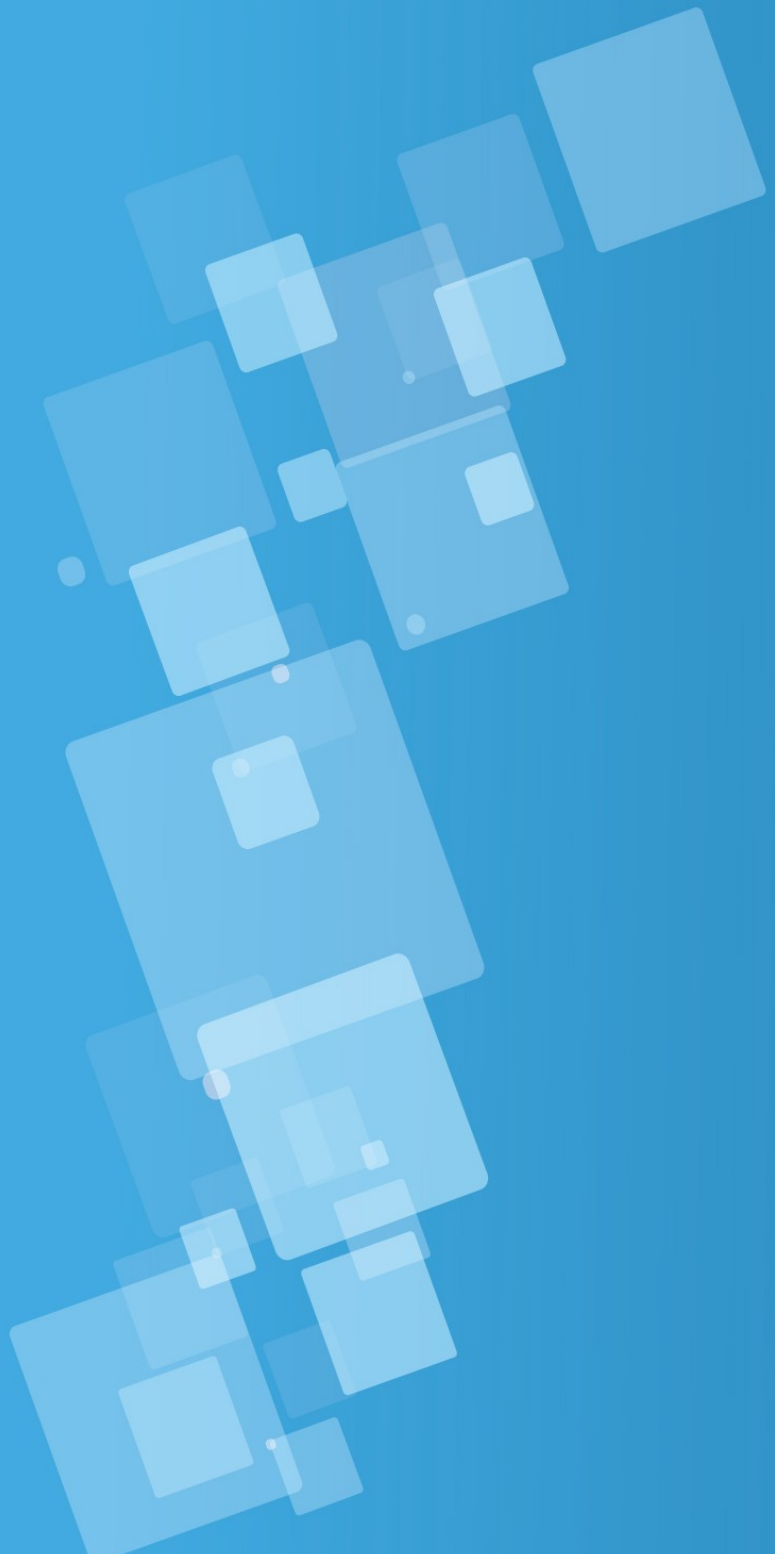


Strategy-driven Test Generation

with Open Source Frameworks

Dimitry Polivaev

UCAAT, 16.09.2014 Munich

About me

Dimitry Polivaev

- Test Engineer and Test Architect by Giesecke & Devrient GmbH working on testing methodology, tool chain and test suites for smart card software,
- Team leader and software developer of non commercial open source software projects hosted by SourceForge,
- Co-founder of Munich Software Craftsmanship Group for practicing TDD, Pair Programming and Clean Code Development.

Agenda

- Motivation
- Terminology
- Test generation by example
- Agile development
- Requirement coverage
- Property combinations by example
- Advanced topics
- Conclusions

Motivation

Terminology

Test generation by example

Agile development

Requirement coverage

Property combinations by example

Advanced topics

Conclusions

Motivation

Test development challenges for complex software systems

- Create thousands of test cases,
- track requirement coverage, tested use cases, used input data combinations etc.,
- avoid redundant test cases and duplicated code,
- simplify development and maintenance.

Test developers work on single test cases and at the same time on the complete test suite as a system of tests.

Agile software development challenges

- Incremental agile and scalable development process,
- Requirements changing at any time,

There is a need of fast, easy and safe identification and modification of all test cases related to the development increments or the requirement changes.

Strategy-driven testing allows to keep control

- It is based on many years of experience in test development for smart cards.
- It allows to implement any kinds of tests for any kinds of test targets and scripting languages.
- It has changed the way and the intensity of testing at my work.
- Now it is implemented as open source project and available for free.

Technology stack (all open source)

- Eclipse (http://www.eclipse.org)
- Xtext (http://www.eclipse.org/Xtext)
- Xbase (http://wiki.eclipse.org/Xbase)
- Test Generator (https://github.com/dpolivaev/test-generation)

Motivation

Terminology

Test generation by example

Agile development

Requirement coverage

Property combinations by example

Advanced topics

Conclusions

Terminology

Test case (specification)

IEEE Standards define test case as follows:

(1) (IEEE Std 610-1990) A set of test inputs, execution conditions, and expected results developed for a particular objective, such as to exercise a particular program path or to verify compliance with a specific requirement.

(2) (IEEE Std 829-1983) Documentation specifying inputs, predicted results, and a set of execution conditions for a test item.

Test procedure, test script files

Test procedure is an implementation of a test case.

- It consists of instructions (test steps) for the set-up, execution, and evaluation of results for a given test case.
- They describe test actions and their input values along with expected results.
- The test steps can be grouped in blocks for set-up (preconditions), test focus, evaluation of results (verification) and tear down (postprocessing).

Test procedures can be grouped in test script files.

Test procedures should be executable in isolation.

Test suite, test categories

- Test suite is a set of test cases (test procedures) providing required test coverage.
- Test cases in a suite can be grouped in test categories.
- The categorization can base on test objectives, covered requirements, input values, expected results etc.

Test driver

Test driver is a component containing definition of test steps used in test case procedures.

It communicates with the test target.

Test drivers are specific to the test target and the test suite.

Test oracle

Test oracle is a (software) component that calculates expected results to be compared with the actual results coming from the test target.

Test case properties, test strategies, strategy rules

Test case properties are elements describing test cases. For example, test steps, test inputs, expected results as well as test categories, test scenarios and even tested requirements are test case properties.

Test strategies are algorithms for systematic or randomized exploration of test property space.

They can be expressed as sets of rules.

Test generation, test generator, test specifications

Test generation is a process of generating test script files with test procedures from test specifications.

Test generator is a program for test generation.

The test specifications are defined as a set of test strategies.

The strategies can be implemented in a test specification DSL.

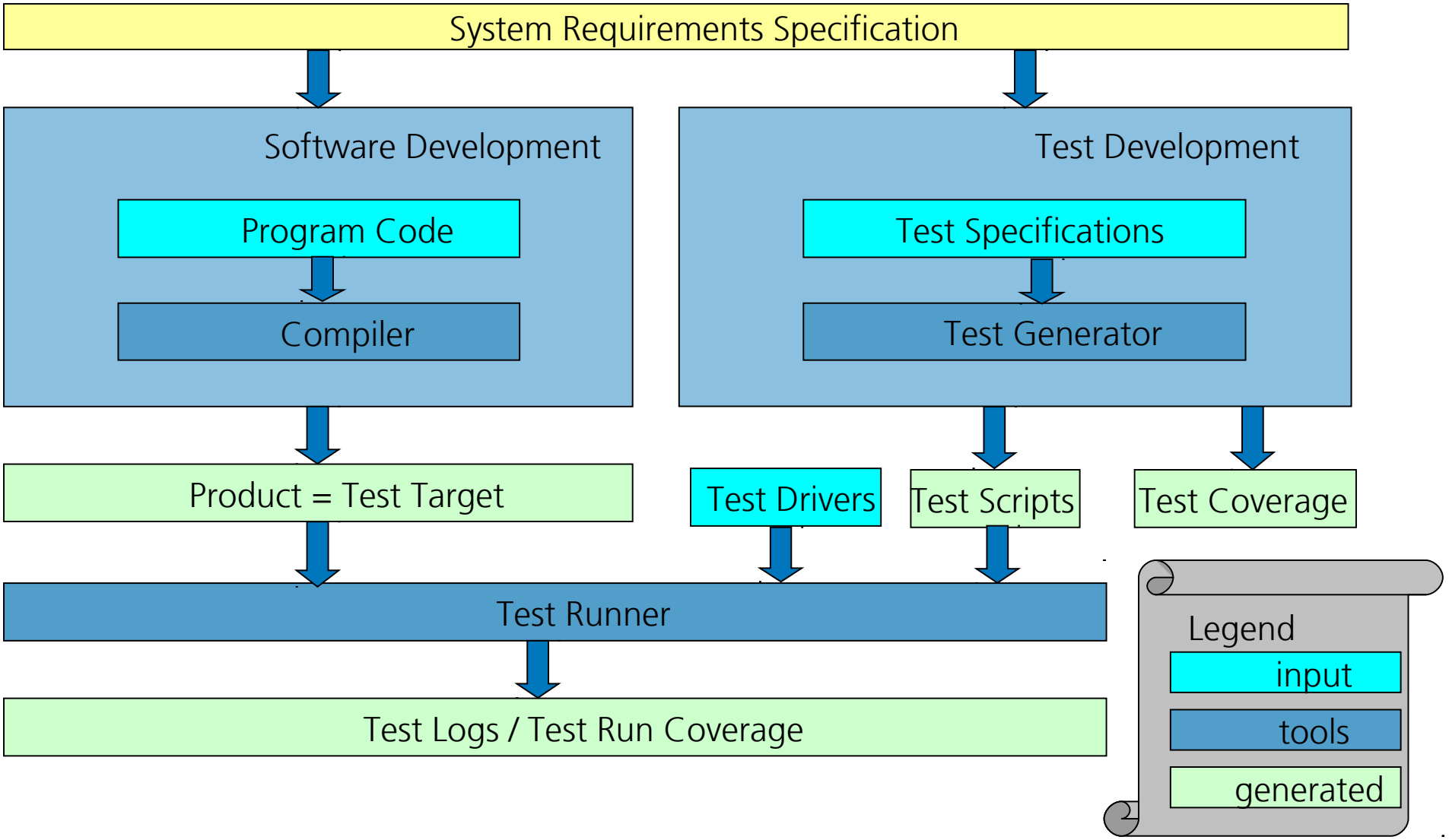
The strategies can use test oracles.

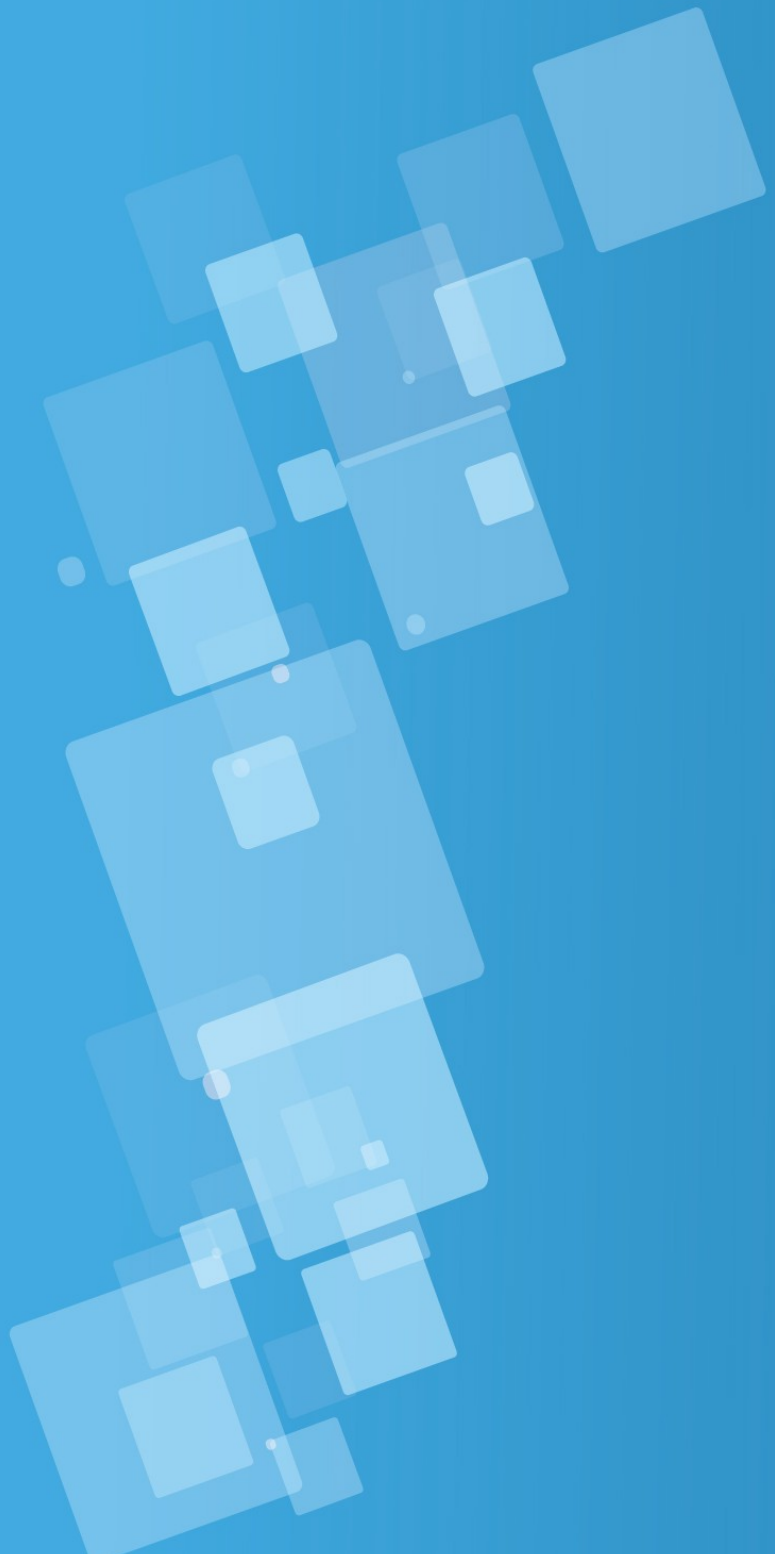
For execution of generated test scripts also test drivers should be implemented.

Test runner

Test runner is an application that executes test instructions contained in test procedures and controls test targets given required test drivers.

Testing vs. software development





Motivation

Terminology

Test generation by example

Agile development

Requirement coverage

Property combinations by example

Advanced topics

Conclusions

Test generation by example

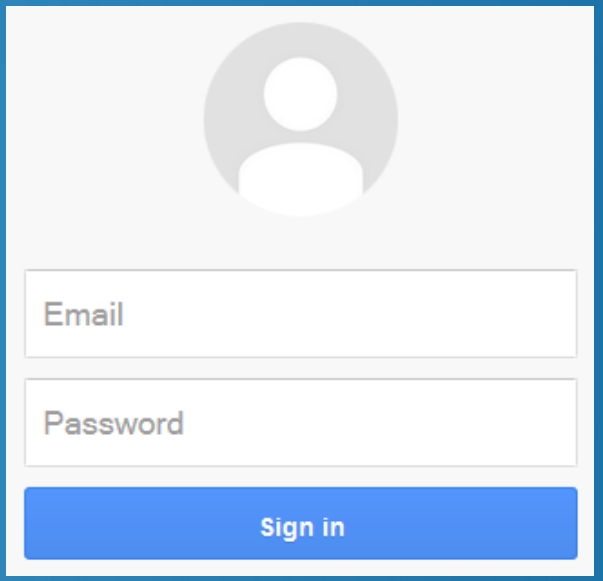
The road to the first test case

- Create a new Java project in Eclipse,
- Add Test Generation Library and new source folder src-gen to its build path,
- Create test specification file (*.testspec),
- Declare properties responsible for test steps,
- Implement and run strategy for one test case,
- Implement test driver and add import statements to the test specification fixing compile errors in the generated tests.

Define properties responsible for test steps

```
package tutorial.login
import org.dpolivaev.testgeneration.engine.strategies.StrategyHelper

strategy TestStepKeywords
let script.precondition.alias be "beforeAll"
let script.postprocessing.alias be "afterAll"
let testcase.precondition.alias be "arrange"
let testcase.focus.alias be "act"
let testcase.verification.alias be "assert"
let testcase.postprocessing.alias be "after"

let lazy testcase.name be StrategyHelper.testcaseName
let lazy testcase.description be StrategyHelper.testcaseDescription
```

Log-in page example



Implement and run strategy for one test case

```
package tutorial.login
import static tutorial.login.TestSteps.*

strategy LoginTests
apply ScriptConfiguration
apply TestDefinition

strategy ScriptConfiguration
apply TestStepKeywords
let script.name be "login/LoginSubmit"
let script.driver be "login/LoginTestDriver"

strategy TestDefinition
let arrange#1 be "go to page(page:login page)"
let arrange#2 be "enter email(email:valid mail)"
let arrange#3 be "enter password(password:valid password)"
let act be "submit form data using https"
let assert#1 be "check page(expectedPage:success page)"

run strategy LoginTests
apply "/java.xslt" output "generated-tests/java"
```

Start test generation

LoginTestSuite.testspec

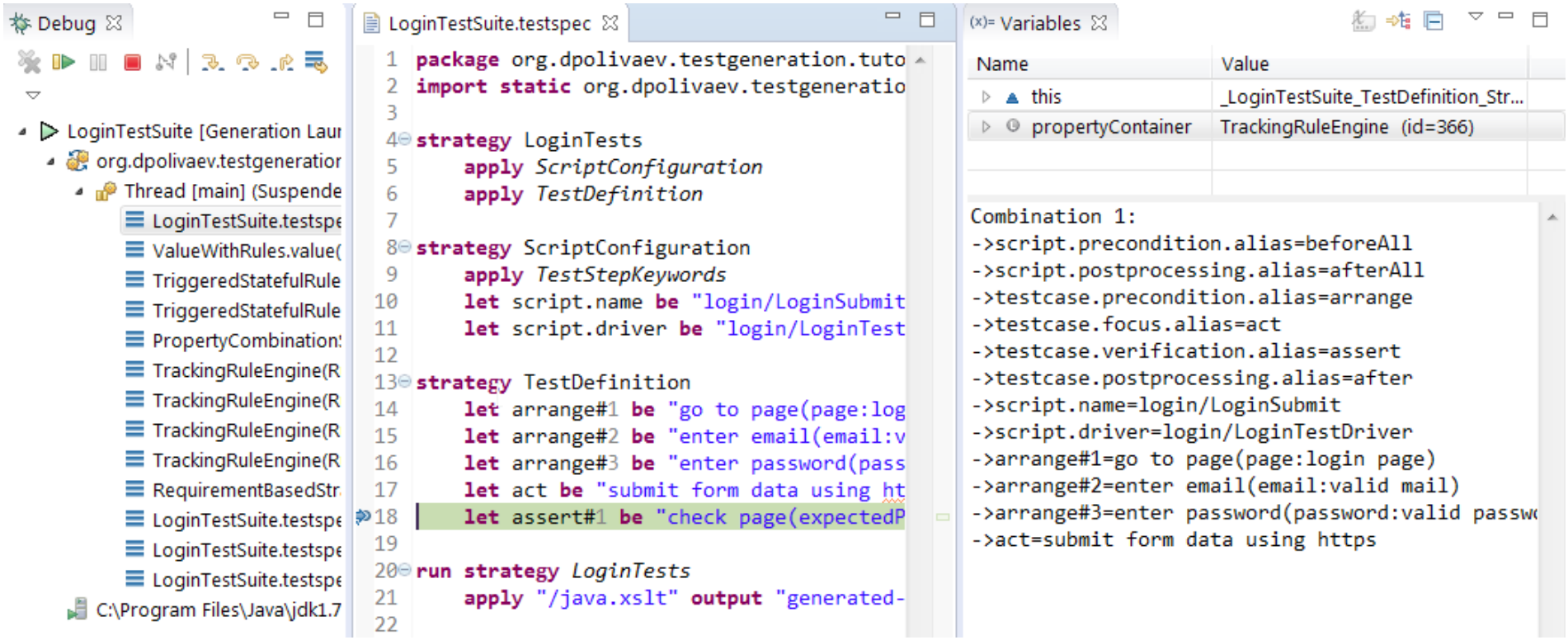
```
1  package org.dpolivaev.testgeneration.tutorial.login
2  import static org.dpolivaev.testgeneration.tutorial.login.TestSteps.*
3
4  strategy LoginTests
5      apply ScriptConfiguration
6      apply TestDefinition
7
8  strategy ScriptConfiguration
9      apply TestStepKeywords
10     let script.name be "login/LoginSubmit"
11     let script.driver be "login/LoginTestDriver"
12     let script.imports be "import static login.LoginTestDriver.Page.*;
13                            import static login.LoginTestDriver.EMail.*;
14                            import static login.LoginTestDriver.Password.*;"
15
16 strategy TestDefinition
17     let arrange#1 be "go to page(page:login page)"
18     let arrange#2 be "enter email(email:valid mail)"
19     let arrange#3 be "enter password(password:valid password)"
20     let act be "submit form data using https"
21     let assert#1 be "check page(expectedPage:success page)"
22
23 run strategy LoginTests
24     apply "/java.xslt" output "ge
25
```

Calculated property values

```
package tutorial.login
import static tutorial.login.TestSteps.*

strategy LoginTests
apply ScriptConfiguration
apply TestDefinition

strategy ScriptConfiguration
apply TestStepKeywords
let script.name be "login/LoginSu
let script.driver be "login/Login

strategy TestDefinition
let arrange#1 be "go to page(page
let arrange#2 be "enter email(ema
let arrange#3 be "enter password(
let act be "submit form data usin
let assert#1 be "check page(expec

run strategy LoginTests
apply "/java.xslt" output "generated-tests/java"
```

```
->script.precondition.alias=beforeAll
->script.postprocessing.alias=afterAll
->testcase.precondition.alias=arrange
->testcase.focus.alias=act
->testcase.verification.alias=assert
->testcase.postprocessing.alias=after
->script.name=login/LoginSubmit
->script.driver=login/LoginTestDriver
->arrange#1=go to page(page:login page)
->arrange#2=enter email(email:valid mail)
->arrange#3=enter password(password:valid password)
->act=submit form data using https
->assert#1=check page(expectedPage:success page)
<-testcase.name=submit form data using https
```

Debugger


Debug
LoginTestSuite [Generation Laur
org.dpolivaev.testgeneratior
Thread [main] (Suspende
LoginTestSuite.testspe
ValueWithRules.value(
TriggeredStatefulRule
TriggeredStatefulRule
PropertyCombination
TrackingRuleEngine(R
TrackingRuleEngine(R
TrackingRuleEngine(R
TrackingRuleEngine(R
RequirementBasedStr
LoginTestSuite.testspe
LoginTestSuite.testspe
LoginTestSuite.testspe
C:\Program Files\Java\jdk1.7
LoginTestSuite.testspec
1 package org.dpolivaev.testgeneration.tuto
2 import static org.dpolivaev.testgeneratio
3
4 strategy LoginTests
5 apply ScriptConfiguration
6 apply TestDefinition
7
8 strategy ScriptConfiguration
9 apply TestStepKeywords
10 let script.name be "login/LoginSubmit
11 let script.driver be "login/LoginTest
12
13 strategy TestDefinition
14 let arrange#1 be "go to page(page:log
15 let arrange#2 be "enter email(email:v
16 let arrange#3 be "enter password(pass
17 let act be "submit form data using ht
18 let assert#1 be "check page(expectedP
19
20 run strategy LoginTests
21 apply "/java.xslt" output "generated-
22
Variables
Name
Value
this
_LoginTestSuite_TestDefinition_Str...
propertyContainer
TrackingRuleEngine (id=366)
Combination 1:
->script.precondition.alias=beforeAll
->script.postprocessing.alias=afterAll
->testcase.precondition.alias=arrange
->testcase.focus.alias=act
->testcase.verification.alias=assert
->testcase.postprocessing.alias=after
->script.name=login/LoginSubmit
->script.driver=login/LoginTestDriver
->arrange#1=go to page(page:login page)
->arrange#2=enter email(email:valid mail)
->arrange#3=enter password(password:valid passwo
->act=submit form data using https

Count test step numbers automatically

```
class StepCounters
val arrangeSteps = stepCounter("arrange")
val assertSteps = stepCounter("assert")

strategy TestDefinition
val stepCounters = new StepCounters
let (stepCounters.arrangeSteps.next) be "go to page(page:login page)"
let (stepCounters.arrangeSteps.next) be "enter email(email:valid mail)"
let (stepCounters.arrangeSteps.next) be "enter password(password:valid password)"
let act be "submit form data using https"
let (stepCounters.assertSteps.next) be "check page(expectedPage:success page)"
```

The first generated test script

```
public class LoginSubmit {
  LoginTestDriver driver = new LoginTestDriver();

  @Description("")
  @Test
  public void test001_submitFormDataUsingHttps() throws Exception {
    // Precondition 1
    driver.goToPage(/* page */ LOGIN_PAGE);
    // Precondition 2
    driver.enterEmail(/* email */ VALID_MAIL);
    // Precondition 3
    driver.enterPassword(/* password */ VALID_PASSWORD);
    // Focus 1
    driver.submitFormDataUsingHttps();
    // Verification 1
    driver.checkPage(/* expectedPage */ SUCCESS_PAGE);
  }
}
```

Implement test driver

```
package login;

public class LoginTestDriver {
  public enum Page {LOGIN_PAGE, SUCCESS_PAGE}
  public enum EMail {VALID_MAIL}
  public enum Password {VALID_PASSWORD}

  public void goToPage(Page page) {}
  public void enterEmail(EMail email) {}
  public void enterPassword(Password password) {}
  public void submitFormDataUsingHttps() {}
  public void checkPage(Page page) {}
}
```

Add import statements to the test specification

```
strategy ScriptConfiguration
apply TestStepKeywords
let script.name be "login/LoginSubmit"
let script.driver be "login/LoginTestDriver"
let script.imports be "import static login.LoginTestDriver.Page.*;
   import static login.LoginTestDriver.EMail.*;
   import static login.LoginTestDriver.Password.*;"
```

Motivation

Terminology

Test generation by example

Agile development

Requirement coverage

Property combinations by example

Advanced topics

Conclusions

Agile development

Extend the test suite step by step

- Add test for invalid mail,
- Add test for invalid password using lazy rules,
- Add test for invalid protocol varying test steps,
- Respond to a case of specification change: new CAPTCHA field added to the form,
- Add test for invalid CAPTCHA.

After each step neither the test specification nor the test driver should contain duplicated code.

Add test for invalid mail

```
strategy TestDefinition
val stepCounters = new StepCounters
let (stepCounters.arrangeSteps.next) be "go to page(page:login page)"
let (stepCounters.arrangeSteps.next) be "enter email(:email)"
let (stepCounters.arrangeSteps.next) be "enter password(password:valid password)"
let act be "submit form data using https"
let (stepCounters.assertSteps.next) be "check page(:expectedPage)"

let email be "valid mail", "invalid mail"
let expectedPage be if (:email == "valid mail") "success page" else "login page"
```

Calculated property values for 2 test cases

```
strategy TestDefinition
val stepCounters = new StepCounters
let (stepCounters.arrangeSteps.next) be "go to page(page:login page)"
let (stepCounters.arrangeSteps.next) be "enter email(:email)"
let (stepCounters.arrangeSteps.next) be "enter password(password:valid password)"
let act be "submit form data using https"
let (stepCounters.assertSteps.next) be "check page(:expectedPage)"

let email be "valid mail", "invalid mail"
let expectedPage be if (:email == "valid mail") "success page" else "login page"
```

```
->arrange#1=go to page(page:login page)
->arrange#2=enter email(:email)
->arrange#3=enter password(password:valid password)
->act=submit form data using https
->assert#1=check page(:expectedPage)
->email=valid mail
->expectedPage=success page
```

```
->arrange#1=go to page(page:login page)
->arrange#2=enter email(:email)
->arrange#3=enter password(password:valid password)
->act=submit form data using https
->assert#1=check page(:expectedPage)
->email=invalid mail
->expectedPage=login page
```

Generated test case

```
@Description("")
@Test
public void test002_invalidMailSubmitFormDataUsingHttps()
  throws Exception {

// Precondition 1
driver.goToPage(/* page */ LOGIN_PAGE);
// Precondition 2
driver.enterEmail(/* email */ INVALID_MAIL);
// Precondition 3
driver.enterPassword(/* password */ VALID_PASSWORD);
// Focus 1
driver.submitFormDataUsingHttps();
// Verification 1
driver.checkPage(/* expectedPage */ LOGIN_PAGE);
}
```

Add test for invalid password

```
strategy TestDefinition
val stepCounters = new StepCounters
let lazy testcase.name be StrategyHelper.testcaseName.include("testPurpose")
let (stepCounters.arrangeSteps.next) be "go to page(page:login page)"
let (stepCounters.arrangeSteps.next) be "enter email(:email)"
let (stepCounters.arrangeSteps.next) be "enter password(:password)"
let act be "submit form data using https"
let (stepCounters.assertSteps.next) be "check page(:expectedPage)"

let lazy email be "valid mail"
let lazy password be "valid password"

let testPurpose be "successful login"{},
"invalid email"{ let email be "invalid mail" },
"invalid password"{ let password be "invalid password" }

let expectedPage be
   if (:testPurpose == "successful login") "success page" else "login page"
```

Calculated property values for invalid password

```
strategy TestDefinition
val stepCounters = new StepCounters
let lazy testcase.name be StrategyHelper.testcaseName.include("testPurpose")
let (stepCounters.arrangeSteps.next) be "go to page(page:l
let (stepCounters.arrangeSteps.next) be "enter email(:emai
let (stepCounters.arrangeSteps.next) be "enter pass
let act be "submit form data using https"
let (stepCounters.assertSteps.next) be "check page(:expect

let lazy email be "valid mail"
let lazy password be "valid password"

let testPurpose be "successful login"{},
"invalid email"{ let email be "invalid mail" },
"invalid password"{ let password be "invalid passwo

let expectedPage be
   if (:testPurpose == "successful login") "success
```

```
->arrange#1=go to page(page:login
page)
->arrange#2=enter email(:email)
->arrange#3=enter
password(:password)
->act=submit form data using
https
->assert#1=check
page(:expectedPage)
->testPurpose=invalid password
testPurpose->password=invalid
password
->expectedPage=login page
<-email=valid mail
<-testcase.name=submit form
data using https login page
invalid password
```

Generated test case

```
@Description("testPurpose: invalid password")
@Test
public void test003_submitFormDataUsingHttpsLoginPageInvalidPassword()
  throws Exception {

// Precondition 1
driver.goToPage(/* page */ LOGIN_PAGE);
// Precondition 2
driver.enterEmail(/* email */ VALID_MAIL);
// Precondition 3
driver.enterPassword(/* password */ INVALID_PASSWORD);
// Focus 1
driver.submitFormDataUsingHttps();
// Verification 1
driver.checkPage(/* expectedPage */ LOGIN_PAGE);
}
```

Add test for invalid protocol

```
strategy TestDefinition
val stepCounters = new StepCounters
let (stepCounters.arrangeSteps.next) be "go to page(page:login page)"
let (stepCounters.arrangeSteps.next) be "enter email(:email)"
let (stepCounters.arrangeSteps.next) be "enter password(:password)"
let act be "submit form data using :protocol"
let (stepCounters.assertSteps.next) be "check page(:expectedPage)"

let lazy email be "valid mail"
let lazy password be "valid password"
let lazy protocol be "https"

let testPurpose be "successful login"{},
"invalid email"{ let email be "invalid mail" },
"invalid password"{ let password be "invalid password" },
"invalid protocol"{ let protocol be "http" }

let expectedPage be
   if (:testPurpose == "successful login") "success page" else "login page"
```

Generated test case

```
@Description("testPurpose: invalid protocol")
@Test
public void test004_submitFormDataUsingHttpLoginPageInvalidProtocol()
  throws Exception {

// Precondition 1
driver.goToPage(/* page */ LOGIN_PAGE);
// Precondition 2
driver.enterEmail(/* email */ VALID_MAIL);
// Precondition 3
driver.enterPassword(/* password */ VALID_PASSWORD);
// Focus 1
driver.submitFormDataUsingHttp();
// Verification 1
driver.checkPage(/* expectedPage */ LOGIN_PAGE);
}
```

New CAPTCHA field was added to the form

LoginTestSuite.testspec: one line was added

```
let (stepCounters.arrangeSteps.next) be "go to page(page:login page)"
let (stepCounters.arrangeSteps.next) be "enter email(:email)"
let (stepCounters.arrangeSteps.next) be "enter password(:password)"
let (stepCounters.arrangeSteps.next) be "enter captcha(captcha:valid captcha)"
let act be "submit form data using :protocol"
let (stepCounters.assertSteps.next) be "check page(:expectedPage)"
```

enterCapthca call was added to all generated test cases e.g.

```
@Description("testPurpose: successful login")
@Test
public void test001_submitFormDataUsingHttpsSuccessPageSuccessfulLogin() throws Exception {
// Precondition 1
driver.goToPage(/* page */ LOGIN_PAGE);
// Precondition 2
driver.enterEmail(/* email */ VALID_MAIL);
// Precondition 3
driver.enterPassword(/* password */ VALID_PASSWORD);
// Precondition 4
driver.enterCaptcha(/* captcha */ VALID_CAPTCHA);
// Focus 1
driver.submitFormDataUsingHttps();
// Verification 1
driver.checkPage(/* expectedPage */ SUCCESS_PAGE);
}
```

Add test for invalid CAPTCHA

- `let (stepCounters.arrangeSteps.next) be "enter captcha(:captcha)"`
- `let lazy captcha be "valid captcha"`
- `let testPurpose be … "invalid captcha"{ let captcha be "invalid captcha" }`

Motivation

Terminology

Test generation by example

Agile development

Requirement coverage

Property combinations by example

Advanced topics

Conclusions

Requirement coverage

Properties for requirement tracking

- Properties starting and ending with rectangular brackets like **`[Req1]`** are interpreted as references to requirements,
- their values explain the connection between tests and requirements.
- Keywords **`goal`** and **`report`** are used for requirement statistics generation.
- Evaluation of requirement coverage helps to identify missing verification test steps, test cases or test categories.

Track requirement R1

Requirement R1: After submitting the form with valid user credentials the log-in success page should be shown

```
let testPurpose be "successful login"{
  let [R1] be "Check login success if valid user credentials are entered"
}
```

Generated test script:

```
@Description("testPurpose: successful login")
@Coverage(first = {
  @GoalCoverage(goal = "requirement coverage", item="R1",
  coverage={"Check login success if valid user credentials are entered"})
})
@Test
public void test001_submitFormDataSuccessPage() throws Exception { ...
```

Configure generator to create test coverage report

LoginTestSuite.testspec

```
run strategy goal LoginTests
  apply "/java.xslt" output "generated-tests/java"
  report  "report/testcoverage.xml"
```

testcoverage.xml

```
<?xml version="1.0" encoding="UTF-8"?><Report>
  <Goal name="requirement coverage" required="1" achieved="1" total="1">
    <Item name="R1" reached="1">Check login success if valid user credentials are entered</Item>
  </Goal>
</Report>
```

Track requirement R2

Requirement R2: The page must be treated as sensitive data (https)

```
let testPurpose be ...

"invalid protocol"{
  let protocol be "http"
  let [R2] be "Check that login does not succeed if protocol http is used"
},
```

Add verification steps for requirement R3

Requirement R3: After entering invalid email address or password or captcha an error message should appear and the password should be removed and the email stays

```
for each testPurpose if (:testPurpose != "successful login") {
  let (stepCounters.assertSteps.next) be "check error message"
  let (stepCounters.assertSteps.next) be "check empty password field"
  let (stepCounters.assertSteps.next) be "check email field contains(:email)"
  let [R3] be "Verify error message, empty password field and non emtpy email field after
failed login"
}
```

```
@Test
public void test003_submitFormDataUsingHttpsLoginPageInvalidPassword() throws Exception {
// Precondition 1
driver.goToPage(/* page */ LOGIN_PAGE);
// Precondition 2
driver.enterEmail(/* email */ VALID_MAIL);
// Precondition 3
driver.enterPassword(/* password */ INVALID_PASSWORD);
// Precondition 4
driver.enterCaptcha(/* captcha */ VALID_CAPTCHA);
// Focus 1
driver.submitFormDataUsingHttps();
// Verification 1
driver.checkPage(/* expectedPage */ LOGIN_PAGE);
// Verification 2
driver.checkErrorMessage();
// Verification 3
driver.checkEmptyPasswordField();
// Verification 4
driver.checkEmailFieldContains(/* email */ VALID_MAIL);
}
```

Add test for requirement R4

Requirement R4: The logon page must use the password field to keep the password from being viewable

```
strategy TestDefinition
let lazy testcase.name be StrategyHelper.testcaseName.include("testPurpose")
let testPurpose#1 be
"Check data processing" { apply CheckDataProcessing },
"Check formatting" { apply CheckFormatting }

strategy CheckFormatting
let testPurpose be "check that password is hidden"{
  let arrange be "go to page(page:login page)"
  let act be "enter password(password:valid password)"
  let assert be "check that password is hidden"
  let [R4] be :assert
}

strategy CheckDataProcessing
val stepCounters = new StepCounters
let (stepCounters.arrangeSteps.next) be "go to page(page:login page)"
let (stepCounters.arrangeSteps.next) be "enter email(:email)"
```

Test coverage report

```
<?xml version="1.0" encoding="UTF-8"?><Report>
  <Goal name="requirement coverage" required="4" achieved="4" total="4">
    <Item name="R1" reached="1">Check login success if valid user credentials are entered</Item>
    <Item name="R2" reached="1">Check that login does not succeed if protocol http is used</Item>
    <Item name="R3" reached="4">Verify error message, empty password field and non emtpy email
field after failed login</Item>
    <Item name="R4" reached="1">check that password is hidden</Item>
  </Goal>
</Report>
```

Discussion

- Strategy describes test suite as a whole by defining variation of properties.
- The same mechanism is used to describe variation of test steps, input values, script and test case names, test categories and requirement tracking.
- Property values and even their names can be defined as expressions depending on other properties.
- Defined once property values can be used for as many test cases as needed.
- Elimination of redundancy makes as well coverage tracking as changes in the test suite and incremental development easier.
- Test script syntax is not relevant for the strategy. The output format can be defined as xslt transformation. The scripts can be generated for any language.

Motivation

Terminology

Test generation by example

Agile development

Requirement coverage

Property combinations by example

Advanced topics

Conclusions

Property combinations by example

Property combinations

- Test strategy defines rules describing which test properties are set and how they should be combined.
- This sections demonstrates the syntax of the rules.
- The rule interpreter is the core part of the test generator.

Property combinations by example 1

```
strategy SyntaxExamples
let CASE be
'stand alone property' {
  let x be 'a', 'b', 'c'
    },
'all combinations' {
    let x be 'a', 'b', 'c' {
          let y be  'A', 'B', 'C'
        }
    },
```

```
'stand alone property'
CASE->x=a
CASE->x=b
CASE->x=c

'all combinations'
CASE->x=a x->y=A
CASE->x=a x->y=B
CASE->x=a x->y=C
CASE->x=b x->y=A
CASE->x=b x->y=B
CASE->x=b x->y=C
CASE->x=c x->y=A
CASE->x=c x->y=B
CASE->x=c x->y=C
```

Property combinations by example 2

```
strategy SyntaxExamples
let CASE be
'parallel combinations ordered' {
    let ordered x be 'a', 'b', 'c'
    let ordered y be 'A', 'B'
    let ordered z be '1', '2', '3',
                     '4', '5'
    },
'parallel combinations shuffled'{
    let shuffled x be 'a', 'b', 'c'
    let shuffled y be 'A', 'B'
    let shuffled z be '1', '2', '3',
                      '4', '5'
    },
'parallel combinations first ordered
          then shuffled by default' {
    let x be 'a', 'b', 'c'
    let y be 'A', 'B'
    let z be '1', '2', '3',
             '4', '5', '6'
    },
```

```
'parallel combinations ordered'
CASE->x=a CASE->y=A CASE->z=1
CASE->x=b CASE->y=B CASE->z=2
CASE->x=c CASE->y=A CASE->z=3
CASE->x=a CASE->y=B CASE->z=4
CASE->x=b CASE->y=A CASE->z=5

'parallel combinations shuffled'
CASE->x=b CASE->y=B CASE->z=3
CASE->x=c CASE->y=A CASE->z=5
CASE->x=a CASE->y=B CASE->z=2
CASE->x=b CASE->y=A CASE->z=1
CASE->x=a CASE->y=A CASE->z=4

'parallel combinations first ordered
then shuffled by default'
CASE->x=a CASE->y=A CASE->z=1
CASE->x=b CASE->y=B CASE->z=2
CASE->x=c CASE->y=B CASE->z=3
CASE->x=c CASE->y=A CASE->z=4
CASE->x=a CASE->y=B CASE->z=5
CASE->x=b CASE->y=A CASE->z=6
```

Property combinations by example 3

```
strategy SyntaxExamples
let CASE be
'dependent values'{
    let ordered x be 'a', 'b', 'c'
    let ordered y be 'A', 'B'
    let z be :x + " && " + :y,
             :x + " || " + :y
    },
'explicit dependencies' {
    for each x, y
      let z be (:x " && " :y),
               (:x" || ":y)
    let x be 'a', 'b', 'c'
    let y be'A', 'B'
    },
'branches'{
    let x be 'a', 'b' {
        let y be 'A', 'B'
    },
    c'{let y be 'C'}
    },
```

```
'dependent values'
CASE->x=a CASE->y=A CASE->z=a && A
CASE->x=a CASE->y=A CASE->z=a || A
CASE->x=b CASE->y=B CASE->z=b && B
CASE->x=c CASE->y=A CASE->z=c || A
'explicit dependencies'
CASE->x=a CASE->y=A y->z=a && A
CASE->x=a CASE->y=A y->z=a || A
CASE->x=b CASE->y=B y->z=b && B
CASE->x=b CASE->y=B y->z=b || B
CASE->x=c CASE->y=A y->z=c && A
CASE->x=c CASE->y=A y->z=c || A
'branches'
CASE->x=a x->y=A
CASE->x=a x->y=B
CASE->x=b x->y=A
CASE->x=b x->y=B
CASE->x=c x->y=C
```

Property combinations by example 4

```
strategy SyntaxExamples
let CASE be
'conditions' {
  let x be 'a', 'b', 'c'{
  rule y let y be 'C'
  rule y if :x == 'c' let y be 'A', 'B'
  }
},
'lazy rules' {
   let x be
    'a', 'b', 'c'{},
        'd'{
           let y be 'A', 'B'
        }
   let lazy y be 'C'
   if :x != 'b' let lazy y be 'D', 'E'
   let z be :y
},
```

```
'conditions'
CASE->x=a x->y=C
CASE->x=b x->y=C
CASE->x=c x->y=A
CASE->x=c x->y=B


'lazy rules'
CASE->x=a z<-y=E CASE->z=E
CASE->x=b z<-y=C CASE->z=C
CASE->x=c z<-y=D CASE->z=D
CASE->x=d x->y=A CASE->z=A
CASE->x=d x->y=B CASE->z=B
```

Property combinations by example 5

```
strategy SyntaxExamples
let CASE be
'branched lazy rules'{
  let x be
    'a', 'b', 'c'{},
    'd'{ let y be 'A', 'B' },
    'e' { let lazy y be 'E'}
  let lazy y be 'C'
  let lazy y be 'C'
  if :x != 'b' let lazy y be 'D', 'E'
  let z be :y
},
'chained lazy rules'{
  let x be
    'a', 'b', 'c'{},
    'd'{ let y be 'A', 'B'},
    'e' { let lazy y be 'E'}
    let lazy y be :z
    if :z != 'b' let lazy y be :z
    let lazy z be :x
    if :x != 'b' let lazy y be 'D', 'E'
    let w be :y
},
```

```
'branched lazy rules'
CASE->x=a z<-y=E CASE->z=E
CASE->x=b z<-y=C CASE->z=C
CASE->x=c z<-y=D CASE->z=D
CASE->x=d x->y=A CASE->z=A
CASE->x=d x->y=B CASE->z=B
CASE->x=e z<-y=E CASE->z=E


'chained lazy rules'
CASE->x=a w<-y=D CASE->w=D
CASE->x=b Condition<-z=b w<-y=b CASE->w=b
CASE->x=c w<-y=E CASE->w=E
CASE->x=d x->y=A CASE->w=A
CASE->x=d x->y=B CASE->w=B
CASE->x=e w<-y=E CASE->w=E
```

Property combinations by example 6

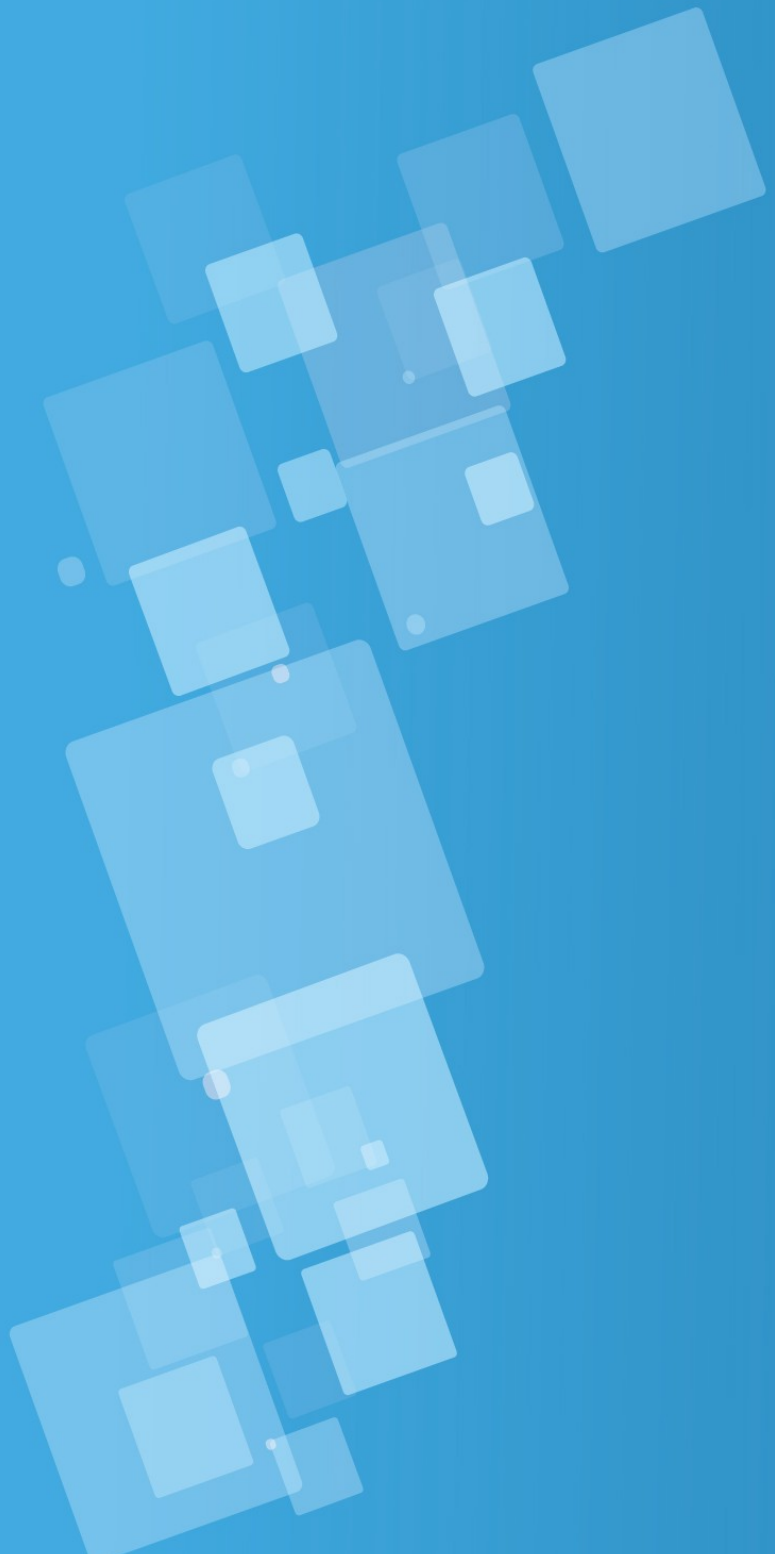
```
strategy SyntaxExamples
let CASE be
'skip'{
    let x be 'a', 'b', 'c'{
       if :x == 'b' skip
       let y be 'A', 'C'
    }},
'conditionally skip values'{
    let x be 'a', 'b', 'c'{
       let ordered y be 'A' only if :x=='a',
       'B' only if :x=='b',
       'C'
    }},
"new rules override old rule's content
 and keep rule evaluation order" {
    rule x let x be "a"
    let lazy y be "A"
    let lazy y be "B"
    let lazy y be "C"
    let z be :y
    rule x let x be "b"
    rule x let x be "c"
    },
```

```
'skip'
CASE->x=a x->y=A
CASE->x=a x->y=C
CASE->x=c x->y=A
CASE->x=c x->y=C

'conditionally skip values'
CASE->x=a x->y=A
CASE->x=a x->y=C
CASE->x=b x->y=B
CASE->x=b x->y=C
CASE->x=c x->y=C

'new rules override old...'
CASE->x=c z<-y=C CASE->z=C
```

Property combinations by example 7

```
strategy SyntaxExamples
let CASE be
'named value lists' {
  let x be namedListOfXValues with 'a','b','c'{
    let y be 'A', 'B'
  }
},
'value list references' {
  let x be from namedListOfXValues
  let shuffled lazy y be from namedListOfXValues
  let z be :y
},
'big integer' {
  let x be 0xff#BI
}
```

```
'named value lists'
CASE->x=a x->y=A
CASE->x=a x->y=B
CASE->x=b x->y=A
CASE->x=b x->y=B
CASE->x=c x->y=A
CASE->x=c x->y=B

'value list references'
CASE->x=a z<-y=c CASE->z=c
CASE->x=b z<-y=a CASE->z=a
CASE->x=c z<-y=b CASE->z=b

'big integer'
CASE->x=255
```



Motivation

Terminology

Test generation by example

Agile development

Requirement coverage

Property combinations by example

Advanced topics

Conclusions

Advanced topics

Overview

- Test step definition examples
- Test generation for cucumber scenarios
- Parametrized strategies
- Test step partitioning
- Calculation of expected results with oracle
- Requirement coverage based on oracle code coverage

Test step definition examples

```
strategy stepDefinitions
val steps = stepCounter("step")
let testcase.focus.alias be "test step"
let (steps.next) be "test step without parameters"
let (steps.next) be "commented test step without parameters ; some comment"
let (steps.next) be 'test step with fix string parameter(parameter:"string parameter")'
let (steps.next) be 'test step with fix number parameter(parameter:12345)'
let (steps.next) be 'test step with fix identifier parameter(parameter:fix parameter value)'
let (steps.next) be 'test step with property as parameter(:property)'
let (steps.next) be 'commented test step with property as parameter(:property) ; some comment'
let (steps.next) be 'commented test step with property as parameter(:property)
; some comment distributed
over multiple lines'
let (steps.next) be 'test step with multiple parameters(parameter:fix parameter value,
string:"string" :property)'
let (steps.next) be 'test step with property :property as a step name part'

let (steps.next) be '; only comment, the step does nothing'
let (steps.next) be '"// embedded code fragment
    codeFragmentTestStep();"'

let property be "property value"
```

Generated test steps for Java

```
driver.testStepWithoutParameters();
/* some comment */
driver.commentedTestStepWithoutParameters();
driver.testStepWithFixStringParameter(/* parameter */ "string parameter");
driver.testStepWithFixNumberParameter(/* parameter */ 12345);
driver.testStepWithFixIdentifierParameter(/* parameter */ FIX_PARAMETER_VALUE);
driver.testStepWithPropertyAsParameter(/* property */ PROPERTY_VALUE);
/* some comment */
driver.commentedTestStepWithPropertyAsParameter(/* property */ PROPERTY_VALUE);
/* some comment distributed
over multiple lines */
driver.commentedTestStepWithPropertyAsParameter(/* property */ PROPERTY_VALUE);
driver.testStepWithMultipleParameters(/* parameter */ FIX_PARAMETER_VALUE,
/* string */ STRING_PROPERTY);
driver.testStepWithPropertyPropertyValueAsA_stepNamePart();
/* only comment, the step does nothing */
// embedded code fragment
     codeFragmentTestStep();
```

- Editing xslt template file can change this output as needed.

Test specification in cucumber style

```
package login.testgeneration

strategy loginPageTests

let script.precondition.alias be "background"
let testcase.alias be "scenario"
let scenario.precondition.alias be "given"
let scenario.focus.alias be "when"
let scenario.verification.alias be "then"

let script.name be "login/LoginTest"
let script.feature be "Log in"
let script.description be "I want to be logged in after submitting correct credentials"
 let lazy scenario.name be (:password  " password and " :email " email entered")

 let background be "go to page <page> (page:login)"

 let given#1 be "<email> mail address is entered (:email)"
 let given#2 be "<password> password is entered (:password)"
 let when be "submit is clicked"
 let then be "page <pageAfterSubmit> is opened(:pageAfterSubmit)"

 let email be "valid"{ let password be "valid", "invalid", "empty" },
 "invalid" { let password be "valid" },
 "empty" {let password be "valid", "empty" }

 let pageAfterSubmit be if(:email == "valid" && :password == "valid") "loggedIn" else "login"


run strategy goal loginPageTests
output "output/xml" , apply "/cucumber.xslt" output "generated-tests/feature"
```

Generated cucumber scenarios

```
Feature: Log in

Background:
    Given go to page login

Scenario: valid password and valid email entered
    Given valid mail address is entered
    And valid password is entered
    When submit is clicked
    Then page loggedIn is opened

Scenario: valid password and invalid email entered
    Given invalid mail address is entered
    And valid password is entered
    When submit is clicked
    Then page login is opened

Scenario: invalid password and valid email entered
    Given valid mail address is entered
    And invalid password is entered
    When submit is clicked
    Then page login is opened

Scenario: valid password and empty email entered
    Given empty mail address is entered
    And valid password is entered
    When submit is clicked
    Then page login is opened

Scenario: empty password and valid email entered
    Given valid mail address is entered
    And empty password is entered
    When submit is clicked
    Then page login is opened

Scenario: empty password and empty email entered
    Given empty mail address is entered
    And empty password is entered
    When submit is clicked
    Then page login is opened
```

Parametrized strategies

```
strategy SyntaxExamples
let example be
    'applying external parameterized strategy'{
    let x be 1,2,3
    for each x if (:x as int) < 2 apply calledStrategy(4)
    }

strategy calledStrategy(int a)
val b = 2
let y be a + b, a - b
for each y let ("property" a b) be (:y"-"a"+" b), (:y"-"a"-" b)
for each ("property" a b) let z be ("/" "property" a b "=" :("property" a b) "/")

run strategy SyntaxExamples

run
val parameter = 1
def parameterCall() {return parameter}
strategy calledStrategy(parameterCall)

run strategy calledStrategy(parameter)

run strategy calledStrategy(parameterCall)
```

Step partitioning

```
class StepCounters
val arrangeSteps = stepCounter("arrange")
val gotoLoginPageStep = arrangeSteps.nextSubsequence(1)
val assertSteps = stepCounter("assert")

strategy TestDefinition
let lazy testcase.name be StrategyHelper.testcaseName.include("testPurpose")
val stepCounters = new StepCounters
let (stepCounters.gotoLoginPageStep.next) be "go to page(page:login page)"
let testPurpose#1 be
"Check data processing" { apply CheckDataProcessing },
"Check formatting" { apply CheckFormatting }

strategy CheckFormatting
let testPurpose be "check that password is hidden"{
  let act be "enter password(password:valid password)"
  let assert be "check that password is hidden"
  let [R4] be :assert
}

strategy CheckDataProcessing
val stepCounters = new StepCounters
let (stepCounters.arrangeSteps.next) be "enter email(:email)"
```

Calculation of expected results with oracle

```
strategy CheckDataProcessing
...
let expectedPage be
  if (LoginOracle.isLoginSuccesful(:email, :password, :captcha, :protocol))
    "success page"
  else
    "login page"

oracle LoginOracle
  def boolean isLoginSuccesful(Object email,
                               Object password, Object captcha, Object protocol) {
     return email == "valid mail" && password == "valid password"
       && captcha == "valid captcha" && protocol == "https"
}
```

Requirement coverage based on oracle code coverage

```
let expectedPage be traced
  if (LoginOracle.isLoginSuccesful(:email, :password, :captcha, :protocol))
    "success page"
  else
    "login page"

oracle LoginOracle
def boolean isLoginSuccesful(Object email, Object password, Object captcha, Object protocol){
  if (email == "valid mail" && password == "valid password"
      && captcha == "valid captcha")
    if (protocol == "https")
      ["R1" "Check login success if valid user credentials are entered"] true
    else
      ["R2" "Check that login does not succeed if protocol http is used"] false
    else
      false
}

run strategy goal LoginTests with oracle goal LoginOracle
apply "/java.xslt" output "generated-tests/java"
report  "report/testcoverage.xml"
```

Motivation

Terminology

Test generation by example

Agile development

Requirement coverage

Property combinations by example

Advanced topics

Conclusions

Conclusions

Conclusions

- Test generation framework allows to create test suites from test specifications for any test targets and any programming languages and check requirement coverage by the tests.
- The approach solves problems of test case code duplication.
- It can reduce costs of creating big test suites for complex software systems, particularly when requirements are changing over the time.
- Implementations of test specifications in a DSL and test driver middle-ware are required for generating and running the tests.
- Download the generator and examples at https://sourceforge.net/projects/testgeneration/files

Questions and answers

Thank you !