Quality Assurance for
TTCN-3 Test Specifications

Helmut Neukirchen

Software Engineering for Distributed Systems Group,
Institute for Computer Science,
University of Göttingen, Germany

Outline

1. Introduction
2. TTCN-3 Metrics and Code Smells
3. TTCN-3 Refactoring
4. TReX Tool
5. Summary / Outlook

1. Introduction: TTCN-3

- Testing and Test Control Notation version 3
  - Language for specifying and implementing distributed tests.
  - Standardised by European Telecommunications Standards Institute (ETSI).
- High abstraction level increases productivity and reusability:
  - Abstract from low-level communication and data encoding/decoding.
  - Test independent from lower layers of System under Test (SUT).
- Well defined interface to adaptation layer.
  - Implements communication mechanisms and encoding/decoding.
  - Allows to execute abstract TTCN-3 test specifications.

TTCN-3: Past and Present

- Origin in telecommunication domain:
  - Standardisation bodies publish (e.g. for ISDN, GSM, UMTS):
    - Specification of a communication protocol.
    - Test suite to test conformance of an implementation to its specification.
  - Industry:
    - Implements specified protocols in their equipment,
    - Implements test adaptation layer and executes standardised test suites against their implementation.
- Today:
  - TTCN-3 not only used in telecommunication standardisation domain, but also in domains like Internet, Service-Oriented Architectures, Automotive, ...

TTCN-3 Example

- Look and feel of common programming language:

```
module exampleModule { 
  type record IpAddressType ( character ipAddress ) { 
    template ipMainType localhostTemplate = 
      ipAddress := "127.0.0.1" 
  }

  testcase exampleTestCase[ runs on ExampleComponent ] { 
    all
    | portB receive [localhostTemplate] 
      | set verdict(pass); 
    | portA receive [IpAddressType(1)] 
      | set verdict(fail); 
    } 
} 
```

Motivation

- Huge TTCN-3 test suites (>40000 LOC),
e.g. ETSI test suites for:
  - Session Initiation Protocol (SIP),
  - Internet Protocol Version 6 (IPv6).
- Suffer from quality problems like any larger software!
  - Quality assessment and improvement required!
External Quality of Test Specifications

- ISO 9126 software product quality model:
  - External quality:
    - quality of test in relation to external environment.
    - E.g. correctness of tests, coverage obtained by executing tests against SUT.
    - However, SUT often not available during test development:
      - Standardisation: Only abstract test suites are developed.
        (not executable because adaptation layer is missing).
      - Industry: SUT and tests are developed in parallel and thus SUT is not always available.

Internal Quality of Test Specifications

- Consider internal quality:
  - Quality of test specification on its own.
    - Usually determined by static analysis, e.g. compiler warnings, metrics, code anomaly analysis.
  - Allows to assure test quality during development!
  - Allows to assess properties of source code, e.g. maintainability.

Example Quality Deficiencies

- Excerpt from standardized SIP TTCN-3 test suite:
  ```java
  function ptc_CC_P_B_MCU_MCU_015(CCeq loc_CCeq_s)
  runs on SipComponent
  {
  var Request v_NONE_Request;
  initRFC(loc_CCeq_s);
  v_Default := activate(defaultCCRFC);]
  try RFCv();
  waitForTimeout(65.0+FF_11);
  notRepeatRFCV(FF_11);
  } //end ptc_CC_P_B_MCU_MCU_015
  ```
  - Variable is never used!
  - Default for alternatives activated, but never deactivated.
  - Hard coded "mapc" values.

Quality Assessment and Improvement for TTCN-3 Test Suites

- Approach:
  - Assess test suites, \[\Rightarrow\text{Metrics, Smell Detection}\]
  - Detect issues,
  - Restructure test suites, \[\Rightarrow\text{Refactoring}\]

Outline

1. Introduction
2. TTCN-3 Metrics and Code Smells
3. TTCN-3 Refactoring
4. TRex Tool
5. Summary / Outlook

2. TTCN-3 Metrics (Example)

- Developed using the Goal, Question, Metrics approach.
  - Goal: Improve readability of TTCN-3 source code.
    - Question: "Are any definitions unused or used only once?"
      - Count number of references to definitions.
    \[\Rightarrow\text{Metric: Number of References to Definitions}\]
Bad Smells in TTCN-3 Test Suites

- Metrics sometimes not powerful enough, e.g.:
  - Goal: Improve changeability of TTCN-3 source code.
  - Question: “Do local changes require further non-local changes?”
  - Find duplicated code.

⇒ Pattern-based approach required: code smells.
- Patterns of inappropriate usage of TTCN-3.
- By definition not a smell:
  - Syntax errors,
  - Violation of static semantics,
  - Defects in test case logic.

TTCN-3 Code Smells

- Collected TTCN-3 code smells in a structured catalogue.

⇒ So far identified 38 TTCN-3 code smells with respect to:
  - Duplicated Code, References, Parameters, Complexity,
  - Default Anomalies, Test Behaviour, Test Configuration,
  - Coding Standards, Data Flow Anomalies, Miscellaneous.

Naückirchen, Zeis, Grabowski:
An Approach to Quality Engineering of TTCN-3 Test Specifications.

⇒ Smells only give hints:
- What is considered as smell, may vary from project to project.

TTCN-3 Code Smell (Example): Activation Asymmetry

- Description:
  - Default activation and deactivation are not inside the same statement block.

- Motivation:
  - Improve analysability with respect to active defaults.
  - Enable static analysis of matching default activation and deactivation.

- Options:
  - A missing deactivate may not be considered as code smell inside TTCN-3 test case constructs, since defaults are implicitly deactivated at the end of a test case.

- Related Action(s):
  - Add default deactivation (or activation) if missing.
  - Move matching default activation and deactivation into same statement block.

TTCN-3 Code Smell (Example): Activation Asymmetry

- TTCN-3 Example:

```tcc
module ExampleModule {
  function exampleFunction() return default {
    return activate(exampleAttest());
  }

testcase exampleTestcase() runs on ExampleComponent {
  var default myDefaultVar := null;
  myDefaultVar := exampleFunction();
  alt {
    [] portA.receive(messageOne) { portB.send(messageTwo); }
    deactivate(myDefaultVar);
  }
}
```

Outline

1. Introduction
2. TTCN-3 Metrics and Code Smells
3. TTCN-3 Refactoring
4. TReX Tool
5. Summary / Outlook

3. Refactoring

“A change made to the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behavior.”

Fowler: Refactoring – Improving the Design of Existing Code. Addison-Wesley, 1999

⇒ A refactoring can be used to remove a code smell.

TTCN-3 refactoring catalogue:

1. More than 50 refactorings for improving TTCN-3 test suites:
   - Test behaviour, test data, overall test suite structure.

Naückirchen, Zeis, Grabowski, Evans, Baker:
Example: Inline Template Parameter

- Summary:
  - Inline a template parameter which is always given the same actual value.

- Motivation:
  - Unneeded parameters create code clutter,
  - more coupling than needed.

- Mechanics:
  - Copy template.
  - Remove parameter from formal parameter list,
  - Replace each reference to formal parameter inside the template
  - with the common actual parameter value.
  - Remove actual parameter from each template reference.
  - Remove original template.

Example: Inline Template Parameter

- TTCN-3 example (unrefactored):

```java
module ExampleModule {
    template ExampleType exampleTemplate
        ipAddress = [192.0.0.1];
    }

testcase exampleTestCase() runs on ExampleComponent {
    portA.send(exampleTemplate);
    portB.receive(exampleTemplate);
}
```

Example: Inline Template Parameter

- TTCN-3 example (refactored):

```java
module ExampleModule {
    template ExampleType exampleTemplate
        ipAddress = [192.0.0.1];
    }

testcase exampleTestCase() runs on ExampleComponent {
    portA.send(exampleTemplate);
    portB.receive(exampleTemplate);
}
```

Rule-Based Quality Assessment & Improvement

- Metrics-based:
  - Number of references to a template = 0
    ⇒ Remove Template.
  - Number of references to a template = 1
    ⇒ Inline Template.

- Code Smell-based:
  - Identical actual parameter value
    ⇒ Inline Template Parameter.
  - Duplicate branches in all statements
    ⇒ Extract Alstmt.

Outline

1. Introduction
2. TTCN-3 Metrics and Code Smells
3. TTCN-3 Refactoring
4. TRex Tool
5. Summary / Outlook

4. TRex

- TTCN-3 Refactoring and Metrics tool:
  - Open source plug-ins for Eclipse platform,
  - Integrated TTCN-3 development environment (advanced editing),
  - External compiler integration,
  - Automated calculation of TTCN-3 metrics,
  - Automated detection of TTCN-3 code smells using static analysis,
  - Tool supported TTCN-3 refactoring,
  - Rule-based quality assessment & improvement.
  - Refactorings associated to code smells as a "Quick Fix".
Outline

1. Introduction
2. TTCN-3 Metrics and Code Smells
3. TTCN-3 Refactoring
4. TReX Tool
5. Summary / Outlook

5. Summary and Outlook

- Summary:
  - Quality assessment for TTCN-3 test suites using metrics and smells.
  - Quality improvement of TTCN-3 test suites using refactoring.
  - TReX tool for automated quality assurance of TTCN-3 test suites.
  - Results from standardised test suites.

- Outlook:
  - Continue development of open-source TReX tool.
  - Simulation of TTCN-3 test suites to assess dynamic properties.

- Thank you for your attention!
- Any Questions?
- TReX and further publications available from http://www.trex.informatik.uni-goettingen.de