Re-Usability in Testing

Helmut Neukirchen
Software Engineering
for Distributed Systems Group
Prof. Jens Grabowski
University of Göttingen
Germany
Outline

- Motivation
- Re-Use in Software Development
  - Support for Re-Use by Language Concepts and Techniques
  - Re-Usable Elements
- Re-Use in Test Development
  - Support for Re-Use by Language Concepts and Techniques
  - Re-Usable Elements
- Conclusions
Motivation

- **Re-use** of prefabricated elements is proven for decades in *classical engineering*.

- Re-use of software elements is a mature approach for *developing software*.

- Tests are just a special type of software:
  \[ \Rightarrow \text{Apply re-use techniques also for test development!} \]
Success Stories

- **NEC:**
  - 6.7 times higher productivity,
  - 2.8 times better quality through 17% re-use.

- **DEC:**
  - 25% increase in productivity through 50%–80% re-use.

- **HP:**
  - 24%–76% defect reduction,
  - 40%–50% increase in productivity,
  - 43% reduction in time to market with up to 70% re-use.

- **AT&T:**
  - 50% decrease in time-to-market for 40–90% re-use.

Source: [Leach97, Lim98, Poulin97, Putnam03]
Benefits of Re-Use

- Faster development of software from re-usable elements,
- Less costs for development & maintenance,
- Higher quality when re-using well tested elements,
- Preservation of knowledge of experienced developers.
No Silver-Bullet:
Risks of Re-Use

- Development of re-usable software difficult.
- Understanding re-usable software difficult.
- Not-invented-here syndrome.
- Suitable software development processes.
Requirements on Re-Usable Software

- Organisation,
- Documentation,
- Reliability/Trust,
- Stable interfaces,
- Self-containedness / Independence from other software elements,
- Customisability.
### Overview

<table>
<thead>
<tr>
<th>Concepts:</th>
<th>Software Development:</th>
<th>Test Development:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modules</td>
<td>?</td>
<td></td>
</tr>
<tr>
<td>Object-Orientation</td>
<td>?</td>
<td></td>
</tr>
<tr>
<td>Aspect-Orientation</td>
<td>?</td>
<td></td>
</tr>
<tr>
<td>Refactoring</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Applications</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Components</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Libraries</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Frameworks</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Patterns</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Language concepts supporting re-use**

**Programming techniques supporting re-use**

**Re-useable elements**
Modular Languages

- Reminder: requirements on re-usable software:
  - “Organisation”,
  - “Stable interfaces”,
  - “Self-containedness/Independence from other software elements”.

- Addressed by the notion of **module**:
  - Structuring,
  - Information hiding.

- Allows re-use, but lacks requirement “Customisability”!
Object-Oriented Languages

- Object-oriented concepts
  - Class, Abstract Class, Interface,
  - Inheritance, Polymorphism & Dynamic binding,
  - Generic types,
  - Meta programming/Reflection

allow to develop software which is closed, but still open for changes/extensions.

- But: Does not support separation of crosscutting concerns which restricts full re-use.
Crosscutting Concerns

- Operations often contain several **crosscutting concerns** at the same time:
  - Logging, monitoring, performance optimisation, error checking, error handling.
  - Scattered to several operations.
  - “Pollute” core concern of these operations.

- Logging in `org.apache.tomcat`:
  - Red shows lines of code that are related to logging.
Languages for Aspect-Oriented Programming (AOP)

- Separation of crosscutting concerns can be achieved using the aspect-oriented paradigm [Kiczales97].

- Aspect-oriented languages allow to separate scattered concerns as advice and to weave them back again on well-defined join points.
  - Aspect := advice + selected join point.

- Aspects support better re-usability and customisation.
# Overview

<table>
<thead>
<tr>
<th>Concepts:</th>
<th>Software Development:</th>
<th>Test Development:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modules</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Object-Onteration</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Aspect-Onteration</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Refactoring</td>
<td>?</td>
<td></td>
</tr>
<tr>
<td>Applications</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Components</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Libraries</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Frameworks</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Patterns</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Refactoring

- Systematically improving the internal structure of source code, without altering its external behaviour [Fowler99]. (Unit tests are used as safety net to get confidence that behaviour was not altered.)
  - Goal: Make existing software re-usable, instead of re-writing it from scratch.

- Example: Refactoring **Extract Method** (for Java code):
  ```java
  void printOwing() {
    printBanner();
    //print details
    System.out.println("name: " + theName);
    System.out.println("amount:" + theAmount);
  }

  void printDetails() {
    System.out.println("name: " + theName);
    System.out.println("amount:" + theAmount);
  }
  ```

**Extract Method:**
Extract code & put it into a separate method.
# Overview

<table>
<thead>
<tr>
<th>Concepts</th>
<th>Software Development</th>
<th>Test Development</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modules</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Object-Orientatio</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Aspect-Orientatio</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Refactoring</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Applications</td>
<td>?</td>
<td></td>
</tr>
<tr>
<td>Components</td>
<td>?</td>
<td></td>
</tr>
<tr>
<td>Libraries</td>
<td>?</td>
<td></td>
</tr>
<tr>
<td>Frameworks</td>
<td>?</td>
<td></td>
</tr>
<tr>
<td>Patterns</td>
<td>?</td>
<td></td>
</tr>
</tbody>
</table>
Applications

- Re-use of complete applications:
  - Inclusion of existing application into larger application.
    - Example: Embedding Microsoft Excel spreadsheet into Microsoft Word document.
  - Software product lines: Vary prefabricated application around a set of commonalities.
    - Example: SAP R/3
Components

- Re-use of "software building blocks":
  - Examples: SUN’s JavaBeans, OMG’s CORBA Component Model.

- Services are similar to components:
  - In addition usually distributed.
  - Example: Web Services.
Libraries

- „Classical” procedural libraries:
  - Re-use of functions.
  - Example: C standard lib.

- Class libraries:
  - Re-use of a set of classes.
  - Example: C++ Standard Template Library
Object-Oriented Frameworks

- Frameworks [Lewis95] are like libraries, but the other way around:

- Re-use based on object-oriented concepts.
- Example:
  - Java AWT/Swing: framework for graphical user interfaces.
Patterns

- Idea of patterns [Alexander79]:
  - Proven solutions for problems arising again and again in a certain context.
  - Patterns provide abstract solutions: need to be instantiated.

- Example: Design Patterns [Gamma95]:
  - Re-use proven micro architectures.
## Overview

<table>
<thead>
<tr>
<th>Concepts:</th>
<th>Software Development:</th>
<th>Test Development:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modules</td>
<td>✓</td>
<td>?</td>
</tr>
<tr>
<td>Object-Orientation</td>
<td>✓</td>
<td>?</td>
</tr>
<tr>
<td>Aspect-Orientation</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Refactoring</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Applications</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Components</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Libraries</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Frameworks</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Patterns</td>
<td>✓</td>
<td></td>
</tr>
</tbody>
</table>
Test Development

- Languages and technologies used for test development:
  - Testing and Test Control Notation version 3 (TTCN-3) [ETSI05],
  - UML Testing Profile [OMG04],
  - JUnit [Gamma02].
TTCN-3

- Language concepts supporting re-use:
  - Modules with parameters,
  - Import of
    - TTCN-3 modules (data & behaviour),
    - ASN.1, IDL, XML data descriptions.
  - Refinable test data templates,
  - Type parameterisation,
  - Compatibility rules for re-using component types.

- Intensively used in existing test suites.
  - These concepts are a subset of OO concepts.
  - Further research concerning impact of missing OO concepts on re-use!
UML Testing Profile

- All OO concepts for re-use available.

- No experience which OO concepts for re-use are actually applied in test suites specified using UML Testing Profile.
JUnit

- All OO concepts for re-use available.

- Re-use concepts applied in JUnit test cases:
  - Abstract test case classes containing tests for abstract Java classes under test.
  - Re-use of pre-/postambles (fixtures).
  - Re-usable test case classes (mock objects).
## Overview

<table>
<thead>
<tr>
<th>Concepts:</th>
<th>Software Development:</th>
<th>Test Development:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modules</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Object-Oriented</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Aspect-Oriented</td>
<td>✓</td>
<td>?</td>
</tr>
<tr>
<td>Refactoring</td>
<td>✓</td>
<td>?</td>
</tr>
<tr>
<td>Applications</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Components</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Libraries</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Frameworks</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Patterns</td>
<td>✓</td>
<td></td>
</tr>
</tbody>
</table>
Aspect-Oriented Testing?

- Separate different concerns contained in test cases into aspects.
- Re-use test cases which contain core concerns.
- Weave aspects as required into test cases.
Aspect-Oriented Testing?

- First thought:
  - Test cases contain by definition only a single concern: the test purpose.

- Further thinking:
  - Even test cases are polluted by logging statements, handling of supervision timers, etc.
  - Non-functional test cases are usually functional test cases extended by aspects required for real-time, load, and performance testing.

⇒ Aspect-oriented testing reasonable!
Aspect-Oriented Test Languages?

- AO for **JUnit**:
  - Java AOP extensions can be applied to JUnit test cases as well.

- AO for **UML Testing Profile**:
  - AO extensions/profiles for UML not existing, yet.

- AO for **TTCN-3**:
  - No AO extension for TTCN-3, but:
    - Existing concept of `defaults` for alternatives can be considered as AO.
    - Existing non-functional tests would benefit from AO.
Test Refactoring?

- Systematically improving the internal structure of a test case/test suite, without altering the behaviour of the test.
  - Industrial test suites are large and suffer from “aging” just like ordinary software.
  - Maintenance of standardised test suites requires huge efforts.
Test Refactoring?

- When refactoring implementations, tests are used as safety net to have confidence that behaviour has not changed.
  - As many paths covered as are executed by tests.
- What are safety nets for test cases?
  - Running refactored test against tested implementation is not sufficient, since only one path of the test case is executed!

⇒ Two possible solutions:
  - Bi-simulation of manually refactored and original test case.
  - Tool supported application of formally proven transformation steps.
Refactoring Support for Test Languages?

- **Refactoring for JUnit:**
  - Refactoring of JUnit tests well established and supported by tools.

- **Refactoring for UML Testing Profile:**
  - Refactoring of UML models is known [Sunyé01] and supported by tools.

- **Refactoring for TTCN-3:**
  - Not studied.
### Overview

<table>
<thead>
<tr>
<th>Concepts:</th>
<th>Software Development:</th>
<th>Test Development:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modules</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Object-Orientation</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Aspect-Orientation</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Refactoring</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Applications</td>
<td>✓</td>
<td>?</td>
</tr>
<tr>
<td>Components</td>
<td>✓</td>
<td>?</td>
</tr>
<tr>
<td>Libraries</td>
<td>✓</td>
<td>?</td>
</tr>
<tr>
<td>Frameworks</td>
<td>✓</td>
<td>?</td>
</tr>
<tr>
<td>Patterns</td>
<td>✓</td>
<td>?</td>
</tr>
</tbody>
</table>
Test Product Lines?

- Test product lines might be build around a common (TTCN-3) compiler and run-time platform.

- As part of a test development process, commonalities of test suites may be re-used:
  - Conformance tests may be re-used for regression testing and non-functional testing.
Test Components?

- No component-based test development like in development with software components available.

- TTCN-3 module control part allows to compose test campaigns from pre-defined test cases.

- In distributed testing, “test components” execute the distributed behaviour of a test case. Such “test components” may be re-used:
  - Test components which dynamically create a complex performance test architecture.
  - Test components which are a hub for synchronisation of other test components.
Test Libraries?

- Standardised abstract test suites can be regarded as libraries.
  - Still need to be adapted to concrete implementation under test.

- TTCN-3:
  - Test libraries for certain domains/protocols.

- UML Testing Profile:
  - Data pools and packages of test data and behaviour.

- JUnit:
  - Libraries of mock objects.
Test Frameworks?

- **JUnit** itself is a re-usable OO framework for unit testing.
- **TTCN-3** can be regarded as framework for black box testing.
- **TTCN-2** [ISO97] is a framework and part of a methodology for OSI conformance testing.

- TTCN-3 skeletons are used to enforce a common test case structure [STF176].
  - May be regarded as framework, but missing parts are not plugged in using OO-like concepts – instead, they are copied & modified.
- JUnit and UML Testing Profile test case frameworks are not known.
Test Patterns?

- Technology independent test patterns:
  - Test Design Patterns for testing OO systems [Binder99],
  - Code Review Patterns [Cunningham03].

- Unit test patterns (**JUnit** & friends):
  - Mock Objects [Mackinnon01],
  - C# Unit Test Patterns [Clifton2003],
  - Patterns of Unit Test Automation [Meszaros04].

- **MSC** & **TimedTTCN-3** based real-time test patterns:
  - Real-Time Communication Patterns [Neukirchen04].
Example: Latency Real-Time Communication-Pattern

Context: Two PCOs involved

Context: First event: stimulus

Solution: Time constraint between first and last event

Context: SUT

Abstraction: Decomposed instance

Abstraction: Reference

Context: Last event: response
Relating Timed TTCN-3 to Real-Time Communication-Pattern

```plaintext
import from EvaluationFunctionModule all;

var float timeA, timeB;
...

timeA := self.now;
PCO1.send(m1);
furtherEvents();
PCO2.receive(m1);
timeB := self.now;
setverdict(evalLatencyOnline (timeA, timeB, t1, t2));
```

Library

```plaintext
module EvaluationFunctionModule {
  function evalLatencyOnline
...
}
...
```
## Overview

<table>
<thead>
<tr>
<th>Concepts:</th>
<th>Software Development:</th>
<th>Test Development:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modules</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Object-Oriented</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Aspect-Oriented</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Refactoring</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Applications</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Components</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Libraries</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Frameworks</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Patterns</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
Conclusions

- **Transfer of re-use concepts from software development** may leverage re-use in test development.

- **Support for re-use must be part of development process:**
  - Testing of re-usable software,
  - Re-use of tests:
    - Extreme Programming [Beck2004] where unit tests are refactored together with implementation code.
    - Conformance tests re-used for regression tests, load tests, performance tests.
  - Re-usable test architectures:
    - Generic Web Services test architecture [Dssouli05]
ETSI Work Item “Patterns in Test Development” (PTD)

- Produce guidelines to support test developers in
  - applying test patterns,
  - identifying test patterns (“pattern mining”).

- Work on
  - classification of test patterns,
  - test specific pattern template,
  - methodological aspects,
  - example test patterns.

- Active contributors:
  - University of Göttingen (Software Engineering for Distributed Systems Group), Fraunhofer FOKUS, Nokia Research Center.

- Supporters:
  - Ericsson, mmO2.

Web page: [PTD05]
End of Presentation

- Thank you for your attention!
- Any questions?
References (1/3)


References (2/3)

References (3/3)


