How Industrial Testing Can Benefit From Formal Methods

Teade Punter
Eindhoven University
How Industrial Testing Can Benefit From Formal Methods

Teade Punter & Petra Heck

1st of December 2005
Who are we?

- LaQuSo – Laboratory for Quality Software
  - Activity of the Faculty of Computer Science and Mathematics, Eindhoven University of Technology
- Verification and Validation of Software
- Bridge gap between research and industry needs
- Themes:
  - Case Studies (Certificate)
  - Model Structure
  - Software Structure
  - Model Behavior
  - Software Behavior
  - Tools
Testing – the process of questioning a product in order to evaluate it

- Find bugs, not prove the absence
- Related to V&V:
  - Verification = "Is the product being built right?"
  - Validation = "Is the right product being built?"

Formal methods (FM) – set of principles and procedures for specification, design and verification

- Based on mathematics, in particular logic
- Examples: model checking, theorem proving
The Need for Formal Methods (FM)

- Critical software applications
  - Example, Ariane 5 rocket explosion, June 1996
- Intensive techniques necessary to test critical applications
  - E.g., ESPRIT SCOPE proposal (Rae et al, 1994)

### System Criticality

<table>
<thead>
<tr>
<th></th>
<th>Level D</th>
<th>Level C</th>
<th>Level B</th>
<th>Level A</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Functionality</strong></td>
<td>functional testing</td>
<td>review (checklists)</td>
<td>component testing</td>
<td>formal proof</td>
</tr>
<tr>
<td><strong>Reliability</strong></td>
<td>programming language</td>
<td>fault tolerance analysis</td>
<td>reliability growth model</td>
<td>formal proof</td>
</tr>
<tr>
<td><strong>Usability</strong></td>
<td>user interface inspection</td>
<td>conformity to interface</td>
<td>laboratory testing</td>
<td>user mental model</td>
</tr>
<tr>
<td></td>
<td></td>
<td>standards</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Efficiency</strong></td>
<td>execution time measurement</td>
<td>benchmark testing</td>
<td>algorithmic complexity</td>
<td>performance profiling analysis</td>
</tr>
<tr>
<td><strong>Maintainability</strong></td>
<td>inspection of documents</td>
<td>static analysis</td>
<td>analysis of development process</td>
<td>traceability evaluation</td>
</tr>
<tr>
<td></td>
<td>(checklists)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Portability</strong></td>
<td>analysis of installation</td>
<td>conformity to programming rules</td>
<td>environment constraints evaluation</td>
<td>program design evaluation</td>
</tr>
</tbody>
</table>
Why formal methods?

- Formal methods help in detecting:
  - Inconsistencies
  - Ambiguities
  - Incompleteness

- But how can formal methods benefit your testing?
Objective of This Presentation

- Provide examples where it is beneficial to apply Formal Methods (FM) for testing
Testing phases

- According to TMap (Pol et al, 2002)
Problems with Testing

- Preparation / Specification -
  Which test cases to conduct? (decision problem)

- Execution -
  How to be sure that all bugs are out? (uncertainty problem)

- Strategy -
  How to find bugs as early as possible? (optimization problem)

- Challenge: can formal methods help to tackle these problems?
Areas where FM can benefit Testing

- Which test cases to conduct?
  - Model-based testing
  - Assertion checking

- How to be sure that all bugs are out?
  - Model checking
  - Statistical testing

- How to find bugs as early as possible?
  - Modeling in early development stages
Model-Based Testing

- Testing with respect to a (formal) model
  - E.g., UML, Promela

- Derive test cases out of the model
  - Choose possible input for the test
  - Calculate expected output from specification
Example: Train Ticket System

- Model
  - ?destination code
  - ?one way
  - ?return

- Test case
  - !destination code
  - !one way
  - !return
  - Other values
    - pass
    - fail
    - fail

Introduction
- Formal Methods & Testing
- Model-Based Testing
- Assertion Checking
- Model Checking
- Statistical Testing
- Conclusions
Experiences with Model-Based Testing

- Different kind, medium scaled systems
  - Smart cards
  - ATM
  - Fail safe system (storm barrier)
- Model is a precise, complete, consistent and unambiguous basis for design
- Enabling test generation
  - Deriving test case automatically and propose test oracles, e.g., Torx
Assertion Checking

- Add statements (assertions) to the code
  - Preconditions
  - Post-conditions
  - Invariants

- Try to prove that assertions cannot be violated

- Types of assertion checking
  - Compile time
  - Run time
Example: Java and JML

```java
package org.jmlspecs.samples.jmlrefman;

public abstract class IntHeap {
    //@ public model non_null int [] elements;

    // lazy instantiation
    public abstract /*@ pure @*/ int largest();

    //@ ensures \result == elements.length;
    public abstract /*@ pure @*/ int size();
}
```
Experiences with Compile time Assertion Checking

- Different kind of systems
  - Java Card electronic purse (Cataño & Huisman, 2002)
  - Internet voting system (Kiniry & Cok, 2004)
  - Toy train security system (Huizing et al., 2005)

- The code is the (formal) model

- Rigorous approach compared to inspections and reviews

- Iterative way of working
  - Tool & manual verification

- Alternative for test case specification

- Tool support available:
  - ESC/Java
  - Krakatoa
Model Checking

- Model – representation of software
  - Exhibits same behavior as software itself
  - Model states and events
- Check if the properties (of interest) hold for the set of states
- Example properties:
  - Deadlock
  - Response time
  - Reachability of a given state
Model Checking

Introduction

Formal Methods & Testing

Model-Based Testing

Assertion Checking

Model Checking

Statistical Testing

Conclusions

The system, Executable program

Model (of the Program)

Verification by Model checker

State space explosion

System satisfied

Violation of properties + counter examples

system requirements

(formal) specification
Example: Process in \(\mu\)CRL

\[
\text{proc } R(b: \text{Bit, queue: List}) = \\
\text{sum}(d: D, \\
\text{read}(b,d) \cdot \text{ack}(b) \cdot R(\text{invert}(b), \\
\text{add}(d,\text{queue}))) \\
\langle\neg \text{is-full}(\text{queue}) \rangle
\]

\textit{delta}

Description:
- process reads a frame (alternating bit and a datum)
- acknowledges the receipt
- adds datum to a bounded queue
Experiences with Model Checking

- Model small, finite systems or system parts
  - State space explosion
- Focus on behavior, not code, look at mathematical properties
  - “completeness”, “soundness”, “deadlock”
- Tools, e.g.,:
  - SPIN – Promela based
  - UPPAAL – timing
  - μCRL – ACP + data
  - CPN – Petri nets
Introduction

Formal Methods & Testing

Model-Based Testing

Assertion Checking

Model Checking

Statistical Testing

Conclusions

**Statistical Testing**

- Test cases are executed and results are logged with time indications
- **Fit to reliability model**
  - Statistical model to estimate e.g., number of defects left
- **Estimate reliability of the software**
- Define your failure distribution
  - Choose appropriate reliability model
    - E.g., Musa Basic Execution Time Model (Musa)

\[ \lambda = \beta (v_0 - \mu) \]

\[ F(T) \]

\[ \text{Expected number of failures} \]

\[ \text{Failure rate} \]

\[ \text{Time} \]

\[ \text{Failure} \]
- Chosen model describes failure behavior
- Determine time and choice in system test, given model parameters, $\beta$ and $\nu_0$
- Stopping criterion is defined by confidence bounds
- Additional:
  - Define failure probability for system components
  - Update probability with new evidence
  - Define appropriate tests
Experiences with Statistical Testing

- Method for release testing (Kruidhof, 2005)
- Choice of model influences outcome
- Coverage analysis
- Tools, e.g.,:
  - Casre
  - Smerfs
Possible Areas Where FM Can Benefit Testing:

- **Preparation / Specification - Which test cases to conduct?**
  - Generate case automatically from a model
  - Assertion checking is alternative for test case generation

- **Execution - How to be sure that all bugs are out?**
  - Proof properties with model checking – 100%
  - Use statistics to be confident

- **Strategy - How to find bugs as early as possible?**
  - Start with formal model, check it and base code on that
• Impact of FM approach on life-cycle phase
  Impact: where to apply

<table>
<thead>
<tr>
<th></th>
<th>Specification</th>
<th>Design</th>
<th>Code</th>
<th>Executing system</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model based testing</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Assertion checking</td>
<td></td>
<td></td>
<td>X</td>
<td>Compile time X</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Run time</td>
</tr>
<tr>
<td>Model checking</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Statistical testing</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>
Outlook and challenges

- Improve usability of Formal methods
  - Experienced people can write formal specifications
  - Formal specifications are not easy to read by business users
- Existing (for model checking and statistical testing) can be improved

Bridging this gap by developing new tools and techniques…
References

• Musa model: http://www.dacs.dtic.mil/techs/baselines/reliability.html

• Casre: http://www.openchannelfoundation.org/projects/CASRE_3.0/
• CPN: http://wiki.daimi.au.dk/cpntools/cpntools.wiki
• ESC/Java: http://www.sos.cs.ru.nl/research/escjava/
• Krakatoa: http://krakatoa.lri.fr/
• SPIN: http://spinroot.com/spin/whatispin.html
• Smerfs: http://www.slingcode.com/smerfs/
• Torx: http://fmt.cs.utwente.nl/tools/torx/introduction.html
• UPPAAL: http://www.uppaal.com/
Visit Address:
TU/e campus, Hoofdgebouw 5.91
Den Dolech 2 Eindhoven

Mail Address:
HG 5.91
Postbus 513
5600 MB Eindhoven

Telephone:
040-2472526
Fax:
040-2474252

Email:
info@laquso.com
Web Site:
www.laquso.com

LaQuSo is an activity of Technische Universiteit Eindhoven