Model-based Testing
Principles

Henry Muccini
Computer Science Department
University of L’Aquila - Italy
muccini@di.univaq.it - www.HenryMuccini.com

Copyright Notice

» The material in these slides may be freely reproduced and distributed, partially or totally, as far as an explicit reference or acknowledge to the material author is preserved.

Henry Muccini
Agenda

» Modeling

» MBT:
  - Why?
  - Models for MBT
  - UML-based Testing
    > Testing activities

Modeling: there is not just programming

» What is a model:
  - A model is a simplification of the reality
  - The reality is abstracted, since it is too complex

» Why modeling
  - Models are built for a better understanding of the system under development

» Everything is a model ?!?
Hardware Modeling

Modeling a bridge
Why modeling

- **To document** design decisions
  - They help specifying how the system we want shall be
  - They allow the specification of the structure and behavior of the system
  - They provide guidelines on how to build the system

- **To analyze** system characteristics, even before the system is built

Why modeling? For Documenting and Analysis
Model-based Testing

Why MBT?

» if source code is not available (not yet developed or not publicly released) test cases can be produced out of a model

» It allows the anticipation of the testing process in the software life-cycle
  - Test cases are still run on the implementation.

» It augments and complements implementation-based testing techniques
**Modeling for testing:**

- How and what to model for testing purposes

**How UML-based testing works**

**Testing stages in MBT:**

- Test Selection
- Test Execution

Essentially, we need (at least) a **behavioral model** of the software system:

- Labelled Transition System
- StateCharts
- UML State of Sequence Diagrams
- IOLTS
- Finite State Automata
- ...
Modeling for testing

» Such a model can be produced:
  - Through a formal specification language
    > Typically referred as formal testing
  - Through a diagrammatic (visual) notation
    > Typically referred as UML-based Testing

Formal Specification-based Testing

» Since the 80’s, many specification-based testing approaches have been proposed, based on formal languages such as
  - Z, VDM, CSP, CCS, LOTOS, SDL, and Petri Nets.
» More recently:
  - Based on dynamic behavior:
    > FSM-based (Bochmann & Petrenko, Lee & Yannakis, …)
    > LTS-based (Brinksma & Tretmans, Jard & Jeron, …)
  - Focussing on static aspects,
    > ADT theory (Bernot & Gaudel, …)
    > Z-based (Hierons, …)
Formal Specification-based Testing

- TorX (Côte de Resyste)
  - on-the-fly test generation and execution
  - random
  - LOTOS and Promela

- TGV (IRISA - Rennes)
  - derives tests in TTCN from LOTOS or SDL
  - uses test purposes

- TVEDA (CNET - France Telecom)
  - derives TTCN tests from SDL specification

UML-based Testing
What is MBT?

- It consists in:
  - Extracting Test Cases from the Model
  - Run the test case over the system implementation

- Purpose:
  - To validate the implementation conformance to the model
  > The model itself is the oracle, i.e., it represents the expected behavior

Model-based Testing

- A model based testing approach

  - accepts two main inputs
    - a model of the software under test,
    - a set of test generation directives which guide the test cases selection

  - and outputs a test specification
    - which includes a set of stimuli the tester should introduce in the system together with expected responses
Model-based Testing

- **UML State-machine based testing:**
  - which might require the translation of the UML diagrams into an intermediate formal description

Model-based Testing > cont.

- **Scenario-based testing:**
  - based on the use of opportune annotated UML Sequence Diagram
  - the approaches provide tools for test cases derivation
    - E.g., Graubmann and Rudolph, Harel and Marelly, TOTEM, SCENTOR, SeDiTeC, etc.

- **State-based and Scenario-based:**
  - combine the use of interaction diagrams and state diagram, possibly annotated with further (formal) information (e.g. OCL)
    - E.g., UMLAUT tool, AGEDIS, SCENT, etc.
Model-based Testing: the overall idea

From Leila Naslavsky Advancement report

Model-based Testing Approaches

<table>
<thead>
<tr>
<th>Method</th>
<th>UCSC-System</th>
<th>ScenToC</th>
<th>SCENTER</th>
<th>COWS-SUITE</th>
<th>SOOTF</th>
<th>TOTEM</th>
<th>AGEDIS</th>
<th>UMLTest</th>
<th>UMLAUT</th>
<th>TESTER</th>
<th>AML</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scenario-based</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>State Machine</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Structural spec.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Other specifications</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Code</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Main Testing Activities

- **Test Selection/Generation:**
  - it consists in selecting a suitable and finite set of test cases from the possibly infinite set;

- **Test Execution and Evaluation:**
  - it consists in executing the code accordingly to the selected test cases and comparing real and expected results;

Test Selection

- **A test case, in MBT, is typically a possible “scenario”**
  - In scenario-based testing:
    - A scenario itself is an abstract test case
  - In state-based testing
    - Execution flows are extraced from the state machine and represent a test case
    - MB coverage criteria are applied on this model
Test Case Selection

» **Abstract Test case** = path over the LTS/FSM, corresponding to sequences of events

> They are not “concrete” test cases, but abstract (depending on the specification)

- **Concrete Test cases** = mapping abstract test cases into execution statements to be launched
Test Case Selection

Test Execution

- It consists in running the test cases on the system implementation
  - So to compare the real execution with the expected behavior
MBT practical concerns

» Traceability, i.e. relating the abstract values of the specification to the concrete values of the implementation.

» Execution, i.e. forcing the Implementation Under Test (IUT) to execute the specific sequence of events that has been selected.

Testing and Industry

» In order to be suitable for industrial needs a testing approach has to emphasize the following qualities:

  - **Usability**
    > Not ad-hoc models
  - **Timeliness**
    > Soon and even incomplete
  - **Tool support**

» **Goal of a suitable testing approach should be “to improve the test results accuracy without sensibly raising the testing effort”**
References


» Further Specialized Readings:
  - L. Mariani and M. Pezz’ e [26] (Book Chapter, 2008): both specification-based and structural-based approaches for intra-class and inter-class testing of OO software are discussed.

MBT State-machine based [references are those in our CBSE05]

» Many model-based testing approaches are state-machine based.
  - Offutt and Abdurazik [21], translate these diagrams into formal SRC specifications
  - Liuying and Zhichang [18], use a formal semantic of state machines to derive the test cases
  - Kim et al. [17], who focus on class testing.

» More recently,
  - Hartmann et al. [15] extend to a component-based paradigm the approach of [21],
  - Antoniol et al. [5], consider the derivation of test sequences from UML statecharts by covering selected paths in a FSM.
Some relevant proposals for scenario-based testing include:

- Graubmann and Rudolph [12], in which MSC inline expressions and hMSC are included into Sequence Diagrams for the specification of test cases;

- the methodology of Harel and Marely [13], which is specifically designed for scenario-based specification of reactive systems;

- TOTEM (Testing Object-orientTEd systems with the unified Modelling language) [8], which uses sequence or collaboration diagrams associated to each use case for deriving test cases, test oracles and test drivers

- SCENTOR [25], which uses JUnit as a basis for test case derivation.

- SeDiTeC [11] automatically generates test stubs for the classes and methods whose behaviour is specified in the sequence diagrams.

- The Cow Suite tools and methodology recently discussed in [6] provides an integrated and practical approach for generating and planning UML-based test suites for industrial contexts.

There are very few approaches, were both state machines and scenarios are required for test case generation.

- UMLAUT (Unified Modelling Language All pUrposes Transformer)[3] is an approach tool supported which translates UML diagrams into an intermediate formal description understandable by the Test Generation and Verification (TGV) tool.

- AGEDIS (Automated Generation and Execution of Test Suites for Distributed Component-based Software)[1], generates and executes test cases for application modelled according to the AML (AGEDIS Modelling Language), which is a specialised UML profile.

- SCENT (SCENario-based validation and Test of software) [24], creates scenarios in a structured way, formalizing them into statecharts.