PHD RESEARCH PROPOSAL: AUTOMATING SCENARIO BASED TESTING WITH UML AND AOP

João Pascoal Faria (jpf@fe.up.pt), FEUP/INESC Porto
PRODEI, 24/2/2012
Motivation – Software Testing

- **Software is important**
  - Increasing % of system functionality realized by software in all domains (e.g., 80% functions in military aircrafts)
  - Increasing criticality of functions realized by software

- **Developing software is challenging**
  - Increasing complexity (~$10^7$ LOCs, #platforms & technol.)
  - Human, knowledge intensive, error-prone (1 error/10 LOC)

- **Bugs in delivered products are common and costly**
  - 1-7 defects/KLOC in delivered products
  - NIST, USA: direct costs of software errors represent 0.6% GNP ($10^{11}$)

- **Testing is important but challenging and costly**
  - Microsoft: 1 tester per developer, still ~1 defect/KLOC
  - Need of more effective and efficient testing methods and tools
Motivation – Model Driven Eng.

- Models are the instrument engineers use to handle complexity

- But the development of computer-based UML design models for documentation only
  - is time consuming
  - the result is often wrong
  - the result soon becomes outdated & is not maintained

- This is a concern both for
  - Educators/students: effective teaching/learning OOD
  - Professionals: cost-effective and agile development of high-quality software
MDE: Possible solutions

- Not using UML or similar notations

- Paper/hand drawings
  - Fast, but difficult to verify and maintain
  - Good for initial thinking

- Reverse engineering (from code to models)
  - Fast, ensures consistency, difficult to abstract details away
  - May be good for documenting but not doing the design

- Automatic code/test generation from models (MDD/MBT)
  - Time invested can be recovered
  - The quality of the models can be checked and improved
  - There is a good chance that they are kept up-to-date
  - More effective & efficient testing
What behavioral UML diagrams?

- **Sequence diagrams / Scenario based testing**
  - Capture essential inter-object/component/system behavior: message exchange
  - Good for iterative use case/scenario driven development
  - Good for specifying test scenarios (unit, interaction, system):
    instead of full heavy-weight formal behavior spec,
    partial light-weight (exec.) behavior spec as test specs
  - Need tools to generate test code from seq. diagrams
    - Check that interactions among objects/components/systems occur as specified, etc.
Goals

- Develop effective and efficient approaches and tools for the automatic generation & execution of
  - Unit
  - Integration
  - and system tests
- from behavioral UML models, particularly interaction diagrams
- Building upon previous work
Example 1

User interaction

API interaction

Internal Interactions

Parametrization and exemplification

User

start()

display("t? ")

enter(x)

display("degrees of freedom? ")

enter(d)

TStudentDistribution(d)

dist

TStudentDistribution

ProbabilityDensityFunction()

calcTotalProbability(x)

calcIntegralWithMaxError(pdf, 0.0, x, 1E-7)

*calcIntegralWithNumSegments(pdf, 0.0, x, -)

*function(-)

calcProbabilityDensity()

p()

display(p)

display("Terminate (y/n)? ")

enter("y")

numerics::SimpsonNumericalIntegration

TestParameters

<table>
<thead>
<tr>
<th>x: double</th>
<th>d: int</th>
<th>p: double</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>9</td>
<td>0.35006</td>
</tr>
<tr>
<td>1.1812</td>
<td>10</td>
<td>0.36757</td>
</tr>
<tr>
<td>2.750</td>
<td>30</td>
<td>0.49500</td>
</tr>
</tbody>
</table>

statistics::TStudentDistributionCLI
Example 2

Stub in the middle
("not yet implemented")
[around advice, call by reflection]

Combined fragments
Example approach (unit testing level)

1. design
2. Test generator (Add-in for EA)
3. Test results
4. Complete method bodies (code)
5. Traces execution of methods & constructors
6. refactor
7. iterate

Enterprise Architect (EA) code generator

UML Class Diagrams

UML Interaction Diagrams

JUnit tests

Java Classes

Java Libraries

Tracing Utilities (AspectJ)

New

developer
Challenges

- Intercept run-time behavior and check conformance with the UML specification
  - With AOP
- User interaction testing
  - Simulate user actions (AOP, around advices, etc.)
  - Map abstract to concrete actions
- Complex features in UML interaction diagrams
- Support the generation of executable tests for different platforms using MDA concepts
- **Testing distributed and concurrent systems**
- **Test data generation**
  - With constraint satisfaction tools
- **Test path generation**
  - With symbolic execution techniques
Previous work and experience

- Automated model-based user interface testing
  - AMBER iTest project, FCT, 2008-11, with Critical Software
  - State-based approach; Black-box user interaction testing

- Automatic test generation from algebraic specs of generic types
  - QUEST project, FCT, 2010-12, with FC/UL
  - Constraint satisfaction approach; Unit testing

- Enterprise architect plug-in for JUnit test generation
  - Initial prototype
  - Zhuanli Yang, IAESTE Internship, 2011
Motivation

The development of computer-based UML design models for documentation only is time consuming and the result is often wrong and soon becomes outdated. This is a concern both for educators and practitioners. But if the UML models are used also as a basis for automatic code generation and/or test generation, then the time invested can be recovered, the quality of the models can be checked and improved, and there is a good chance that they are kept up-to-date. Producing a full behavioral specification in UML is not cost-effective in the general case. Instead, we claim that producing partial behavioral specifications in UML that can act simultaneously as formal test specifications that can be translated automatically to executable tests gives the best balance between quality and productivity.

Goals

The main goals of this research work are to develop approaches and tools for the automatic generation of executable tests from UML behavioral models (particularly interaction diagrams), taking advantage of existing unit testing frameworks and aspect oriented programming techniques for test execution. These approaches and tools should enable a new generation of “model-based test-driven development (TDD)”, that is, a TDD approach in which test are specified in UML.

The work should build upon previous work in user interaction test automation (FCT AMBER iTesT project, 2008-2011) and test automation with AOP.

Some of the challenges are:
- Intercept run-time behavior and check conformance with the UML specification, using AOP techniques;
- Interaction testing in distributed and concurrent systems;
- User interaction testing;
- Support for complex features in UML interaction diagrams;
- Support the generation of executable tests in different platforms using MDA concepts.