Model-based Testing using Visual Contracts

Baris Güldali
07.09.2010, s-lab Research Days, Willebadessen
Software development

[Diagram: V model by Baresi, Pezze]

Requirements → System specification → Subsystem design → Component design → Components → Subsystems → Integrated System → Deliverable

Acceptance testing

System testing

Integration testing

Unit testing

[Notes: MBT using VC - B. Güldali - s-lab Research Days 2010]
Software development

Requirements

Acceptance testing

Deliverable

Testing

Design

Implementation

[V model by Baresi, Pezze]
Model-based software development

Requirements

Design

Tests model

Test case

System model

PIM

PSM

Testing

System model

Test model

Code

Impl.

Code

Deliverable

[V model by Baresi, Pezze]
Model-based software development

- **Requirements**
- **Design**
  - PIM: System model
  - PSM: System model
- **Testing**
  - Test model
  - Test case
- **Impl.**
  - Code
  - Monitoring

Partial completion

class OnlineShop {
  @
  @ requires ( @ precondition @ );
  @ ensures ( @ postcondition @ );
  @
  public String cartAdd(...) {
      ...
  }
}

[V model by Baresi, Pezze]
[VC, MDM by Lohmann’06]
Research questions

- MBT using VC - B. Güldali - s-lab Research Days 2010
Outline

• Model-based development ✔
• Visual Contracts & Related Work
• Testing using Visual Contracts
  – Unit Testing
  – Integration Testing
  – System Testing
• Evaluation & Conclusion
Visual Contracts

- Inspired by Design-by-Contract [Meyer]
- Behavioral specification using pre- and post-conditions
- \( \rightarrow \) Composite structure diagrams typed over a class diagram
- Partial specification
- Based on the theory of graph transformations

[Loehmann'06]
Visual Contracts (example)

- **Control**:
  - OnlineShop
  - `cartCreate() : Cart`
  - `cartAdd(cartId : String, productNo : String, quantity : Integer) : String`

- **Entity**:
  - Cart
    - `cartId : String`
    - `+cartItem`
  - CartItem
    - `cartItemld : String`
    - `+cartItem`
  - Product
    - `productNo : String`
    - `title : String`

- **Typed over**:
  - `vc cartAdd(cid, prNo, num):cartitemid`

- **Intuitive semantics**

- **Further language constructs**

---

**MBT using VC** - B. Güldali - s-lab Research Days 2010
## Contract-based Testing

Systematic comparison criteria based on Vegas, Basili and Dias-Neto

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>People</td>
<td>Low level</td>
<td>Low level</td>
<td>n.A.</td>
<td>Low level</td>
</tr>
<tr>
<td>Process</td>
<td>UT</td>
<td>UT, ST</td>
<td>UT, IT, ST</td>
<td>UT</td>
</tr>
<tr>
<td>Quality</td>
<td>WB</td>
<td>WB</td>
<td>BB</td>
<td>BB</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MBT-specific characteristics</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Artifact</td>
<td>Eiffel</td>
<td>JML</td>
<td>UML</td>
<td>OCL</td>
</tr>
<tr>
<td>Redundancy</td>
<td>Shared</td>
<td>Shared</td>
<td>Separate</td>
<td>Separate</td>
</tr>
<tr>
<td>Automation</td>
<td>Yes</td>
<td>Yes</td>
<td>n.A.</td>
<td>Yes</td>
</tr>
<tr>
<td>Tool</td>
<td>Yes</td>
<td>Yes</td>
<td>n.A.</td>
<td>Yes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CBT-specific characteristics</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Contract</td>
<td>Declarative Artific. states</td>
<td>Declarative Artific. states</td>
<td>Declarative Artific. states</td>
<td>Declarative Artific. states</td>
</tr>
<tr>
<td>Test case</td>
<td>Artific. states</td>
<td>Artific. states</td>
<td>Artific. states</td>
<td>Artific. states</td>
</tr>
</tbody>
</table>
Requirements on a novel approach

- Light-weight easy-to-learn notation
- Throughout support for all test levels
- Integration into UML-based development process
- (Re)use of systems models
- Setting natural states by longer test sequences

**Thesis**: A new model-based testing approach which uses visual contracts as modeling notation can fulfill these requirements.
Outline

• Model-based development ✅
• Visual Contracts & Related Work ✅
• Testing using Visual Contracts
  – Unit Testing
  – Integration Testing
  – System Testing
• Evaluation & Conclusion
Unit testing

1. set components into a controlled pre-state
2. invoke operation under test with test inputs
3. check post-state for correctness

Requirements

System specification

Subsystem design

JUnit

Test case

Integrated System

Components

Subsystem

Deliverable

Component design

Java

VC
1. Generation of input parameters

\[ P = \{ \text{cid="abc", prNo="def", num=1} \} \]

Random and boundary values

2. Generation of pre-state

\[ S_{\text{input}} = \]

3. Setting system state \( s \supseteq S_{\text{input}} \)

- artificially

[Modevva06, Motes08]
Unit testing: Test execution

Test driver

Component

invoke operation

set pre-state

Pass

No Pass

alt

return

exception

execute original operation with call parameters

check pre-condition

check post-condition

return normally

throw pre-condition exception

[pre-condition holds]

[post-condition holds]

[else]

[else]

created by

implemented by

Designer

Programmer

Model-driven Monitoring (MDM)

system state $s_k$

system state $s_{k+1}$

[Modevva06, Motes08]
public void testCartAdd_0()
{
    Product product = new Product();
    product.setTitle("abc");
    product.setASIN("def");
    int quantity = 1;
    String cid = "xyz";

    OnlineShop self = new OnlineShop();
    Cart c = new Cart();

    c.setCartId("xyz");
    self.addCart(c);
    self.addProduct(product);

    //@ requires ( precondition );
    //@ ensures ( postcondition ); @/
    self.cartAdd(product, quantity, cid);
}
Integration testing

1. compute a preamble which sets a subsystem into a controlled pre-state
2. Invoke operation under test with test inputs
3. check post-state for correctness
Integration testing

vc cartCreate():c

vc cartAdd(cid, prNo, num):cartitemid

vc cancelOrder():c

vc processOrder():c

vc Subsystem

Subsystem

cartCreate | cartAdd | cancelOrder | processOrder | ...

[Modevva06, Agtive07]
Integration testing: GT rules

vc cartAdd(p1, 7, 4):ci

NAC
/pr : Product
/c : Cart
cartId = 4
/cinac : CartItem
/cartAdd(p1, 7, 4):ci

OnlineShop
/cartId = 4
/Cart
/c : Cart
cartId = 4
/Product
/quantity = 7
/CartItem
/c : Cart
cartId = 4
/ci : CartItem
/c : Product
/pr : Product
-design

MBT using VC - B. Güldali - s-lab Research Days 2010
Integration testing: GT system

Model checking techniques for computation of the preamble: Reachability analysis for $s_v$

$S_v \supseteq S_{input}$

Modevva06, Agtive07, Motes10
System testing

1. compute an abstract pre-state
2. transform it to a concrete pre-state
3. set pre-state
4. invoke operation(s) under test with test inputs
5. check post-state for correctness
System testing: Deriving test models

Online Shop

- Login
- Add product into cart
- Logout
- Checkout cart

Customer

Trigger: "Order" button is clicked

Precondition: Cart contains cart items. Customer owns a payment method.

Steps:
1. Check the cart items
2. Give delivery address and receipt address
3. Give bank account data
4. Start checkout

Postcondition: An order is generated. A receipt is sent to the customer.

...
System testing: setting pre-state

OMG-QVT

[TAV08, SEAA09]
System testing: Higher order transformation

Abstract test case

System spec.

Test data transformation specification

Refinement specification

Component design.

Concrete test case

UML

Legend:
conforms to
uses

derive

[SEAA09]
Outline

• Model-based development ✔
• Visual Contracts & Related Work ✔
• Testing using Visual Contracts ✔
  – Unit Testing
  – Integration Testing
  – System Testing
• Evaluation & Conclusion
Evaluation: Tool support

- Modeling: Visual Contract Workbench (VCW) + JML
- Unit testing: VCTW + JUnit
- Integration testing: VCTW + Groove or LAMA + JUnit
- System testing: VCTW + QVT + Hibernate

[LOHMANN‘06]
Evaluation: Case studies

- Modeling: sd&m [SE06]
- Testing: student work [Ellerweg06, Beulen08, Klaholt08, Hannwacker08]

- Capable to detect typical errors, e.g.
  - initialization failures
  - missing links, missing objects
  - false variables
## Conclusion: comparison

<table>
<thead>
<tr>
<th></th>
<th>AutoTest</th>
<th>Korat</th>
<th>Built-In Test</th>
<th>WeSUF</th>
<th>VCBT</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>General characteristics</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>People</td>
<td>Low level UT</td>
<td>Low level UT, ST</td>
<td>n.A.</td>
<td>Low level UT</td>
<td>High level UT</td>
</tr>
<tr>
<td>Process</td>
<td>UT</td>
<td>ST</td>
<td>UT, IT, ST</td>
<td>UT</td>
<td></td>
</tr>
<tr>
<td>Quality</td>
<td>Industrial c.s.</td>
<td>Industrial c.s.</td>
<td>Small c.s.</td>
<td>Small c.s.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>MBT-specific characteristics</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Artifact</td>
<td>Eiffel</td>
<td>JML</td>
<td>UML</td>
<td>OC</td>
<td>UML</td>
</tr>
<tr>
<td>Redundancy</td>
<td>Separate</td>
<td>Separate</td>
<td>Separate</td>
<td>Separate</td>
<td>Sep./shared</td>
</tr>
<tr>
<td>Automation</td>
<td>Yes</td>
<td>Yes</td>
<td>n.A.</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Tool</td>
<td>Yes</td>
<td>Yes</td>
<td>n.A.</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>CBT-specific characteristics</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test case</td>
<td>Artificial</td>
<td>Artificial</td>
<td>Artificial</td>
<td>Artificial</td>
<td></td>
</tr>
</tbody>
</table>

**Note:** The comparison highlights the differences and similarities between different testing methods, focusing on aspects such as people involved, process, quality, inventory, and tools used.
Conclusion: fulfillment of requirements

• **Pros**
  + Light-weight easy-to-learn notation
  + Throughout support for all test levels
  + Integration into UML-based development process
  + (Re)use of systems models
  + Setting natural states by longer test sequences

• **Cons**
  – Case study
  – Error detecting capability
  – Integrated tool support
  – Efficiency
    • creation of VCs
    • test case generation
## Own publications on Visual Contracts

<table>
<thead>
<tr>
<th>Year</th>
<th>Event</th>
<th>Authors</th>
<th>Title</th>
<th>Conference/Journal</th>
<th>Volume/Pages</th>
</tr>
</thead>
</table>
Thank you for your attention!

Baris Güldali  
s-lab – Software Quality Lab  
Universität Paderborn  
Warburger Str. 100  
33098 Paderborn  
Tel.: (05251) 60 5392

http://s-lab.upb.de  
bguldali@s-lab.upb.de  
Twitter: bguldali