Cutting MBT Adoption Time with Domain Specific Modeling

Juha-Pekka Tolvanen, PhD, MetaCase
Stephan Schulz, PhD, Conformiq
Contents

• Introduction to DSM and MBT
• DSM + MBT = ?
• Case 1: Web application (IT)
• Case 2: Military radio (embedded)
• Results
• How to get started
• Summary, Q&A
Domain-Specific Modeling (DSM)

• Models expressed with domain concepts
  – No need to learn new languages

• Domain-Specific Modeling allows using:
  – existing terminology,
  – with known semantics, and
  – familiar notation

• DSM is applied in particular for automating repetitive development efforts*, but less in testing

* See references on EADS, NSN, Nokia, Panasonic, Polar Elektro, USAF
Example: Industrial Process Plant
Domain terminology and concepts

• Detailed information specifying functional & physical characteristics of a component of a system, plant or facility (e.g. pump)
Design with domain-concepts

Domain terminology: valves
Example Specification

Closed loop, Heat transfer, Liquid circulating (CHL)

May include:
- System Requirements Tree
- System Requirements
- Component Requirements
- Interface Requirements
How to test a cooling system?

• Temperature
  – Produce too much heat?

• Pressure
  – Incorrect input/output pressure?

• Flow rates
  – Conflicting flow rates in the configuration?

• Control logic

• Instrument configuration
Example: Cooling in process plant*

Specifying properties of components

Generic

Specific
Example: Cooling in process plant*

Both structure and behavior

- Same objects: different views used to formalize different aspects of the system
- Languages integrated: can share objects used in different diagram types

Behavioral constraint:
If valve is closed then pump should be closed.
Else if value is open then pump can be open.
Domain-Specific means:

- Use of concepts from the problem domain
  - Already familiar => no need to learn new
  - Have known semantics
- Having a special focus
  - Use concepts that are relevant for the task: testing, verification, validation
- Use concrete syntax that enables communication and collaboration
  - Not a cryptic programming/scripting language
  - Apply style close to the domain’s natural representation
Steps for Defining Domain-Specific Modeling Languages and Generators

1. Specify language concepts & their properties
2. Define rules for the concepts
3. Create a notation
4. Define generators

Concepts → Rules → Symbols → Generators
About Model-Based Testing (MBT)

- Umbrella term for using models in a testing context
- One approach is to use MBT for automating test design
  - Here model reflects operation of the system to be tested
  - MBT complements test execution
  - Recognized by worldwide industrial standards (ETSI)
Evolution of Software Testing

Automated Test Design (ATD) uses models of system operation as its input and is the most advanced Model Based Testing (MBT) technology.

ATD+ is ATD driven by a domain specific language.
## Test Approach Comparison Heat Map

<table>
<thead>
<tr>
<th>Test Approach</th>
<th>Test Coverage</th>
<th>Early Problem Discovery</th>
<th>Functional Complexity</th>
<th>Test Artifact Reuse</th>
<th>Required Skill Set</th>
<th>Test Process Optimization</th>
<th>Productivity Gain Initial</th>
<th>Productivity Gain Iteration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manual Test</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Test Scripts</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>7</td>
<td>4</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>Test Modeling</td>
<td>7</td>
<td>5</td>
<td>5</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>Automated Test Design</td>
<td>10</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>DSL Driven ATD</td>
<td>10</td>
<td>8</td>
<td>8</td>
<td>9</td>
<td>4</td>
<td>8</td>
<td>8</td>
<td>9</td>
</tr>
</tbody>
</table>
ATD+: DSL driven MBT

• Draws from all benefits of conventional ATD
  – Automated test design and traceability
  – Integration into test automation ecosystem
  – 5x improvements in productivity

• Enables testers to model system operation
  – No longer programming skills required
  – Less training and faster ramp up

• Allows other stakeholders to review models
  – “Shift (really) left” ... engage your customer!

~5x (DSL) combined with ~5x (ATD) = ???
Automated Test Design Workflow

Model System Operation

Direct & Review Test Design

Generate Test Scripts & Documentation

Domain Specific Modeling Tool

Model Based Test Design Tool

Test Execution Tool(s)
Why are DSLs so Important in Testing?

rectangle(3,1, grey)
rectangle(5,2)
circle (2), circle(2)
circle(1), circle(1)

Testing is about achieving a common understanding
Case 1: Conformiq Creator

- A DSL developed for
  - Modeling system operation for **system & system integration & end-to-end testing**
  - First focus on **Enterprise IT applications**, frontends, backends, systems, etc.
  - Target testers and SMEs
- Encodes best practice
  - Provides set of pre-defined modeling building blocks
Modeling before Creator

MainScreen:

Main page display. Model calculates the list of items that will be displayed on the screen.

// MainPage(); displaySend(); var a = basket.keys(); var total_qty = 0;
while (a.hasMoreElements()) {
  var sku = a.nextElement();
  BasketItem i;
  i.sku = sku;
  i.qty = basket.get(sku);
  total_qty += i.qty;
  displaySend();
}

narrative user is told the basket contains " + total_qty + " items";

Wait:

System waits for user’s action

user:Start / narrative "user login";

[qty < 1 || qty > 10] / Error("invalid quantity");
requirement "if Checkout cannot be selected if basket is empty";
narrative "the given quantity is invalid and an error message is displayed";

user:AddItem / sku = mis.sku; qty = mis.qty;
requirement "user tries to add item with SKU " + sku + ", and quantity of " + qty + " to the basket";

user:Checkout / narrative "user attempts checkout";

[sku = "CQ_0001" &&
 sku = "CQ_0002" &&
 sku = "CQ_0003"] / Error("Invalid SKU");
requirement "if Invalid SKUs are rejected";
narrative "error message is displayed because the SKU is invalid";

[else]

[sku] / displaySend();
requirement "(User can do checkout)"

narrative "Checkout is successful" and " + "the user is displayed a statement";

InitialPage:

[narrative "user login"]

The Actual Application to Tested
Creator Concepts

• Activity Diagrams
  – Flows specify specific aspects of system operation to be tested
  – Domain specific actions and data objects from keyword repository concretize activities and decisions

• Interface Diagrams
  – Specify external interfaces available for testing based on pre-defined interface objects
  – Are the source for generated actions and data objects
About Interface Diagrams
About Activity Diagrams

Fulfill a dual purpose:

• Specifies “what” is to be tested, i.e., relevant system operation, in terms of flows
  – Using standard concepts of initial, final, activity, decision, event, merge nodes and control flows

• Specifies “how” to test based on action keywords and data objects generated from interface diagrams
  – Actions from action keyword repository refine activity descriptions
  – Data objects refine (graphical) conditions
Activity Diagram Example

- **Set URL**
- **Form variable**
- **data object**
- **Store form data produced by click action in variable**
- **Compare all form data against multiple values**
- **Conditional (☐) action**
- **Requirement action**
- **Display screen verification action**
- **Refer to subdiagram**
- **Click button action with blocking pre-condition (●)**
# Generic vs Domain Specific

<table>
<thead>
<tr>
<th>Generic Concept</th>
<th>Domain Specific Concept</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class</td>
<td>Message, Screen, Button</td>
</tr>
<tr>
<td>integer, boolean, String</td>
<td>Number, Checkbox, Dropdown Box</td>
</tr>
<tr>
<td>Receive on a port</td>
<td>Click a button, fill a form, Receive a message</td>
</tr>
<tr>
<td>Send from a port</td>
<td>Display a screen, Send a message</td>
</tr>
<tr>
<td>Compare each field of a variable to basic value</td>
<td>Compare entire message or form variable against value</td>
</tr>
</tbody>
</table>

Note: Domain = Application Domain and Testing Domain!
Idea: Simplify, Reduce & Reuse

- Symbols have look & feel closer to application domain
- Abstraction and layering of model information
- Object driven specification enables reuse
- Changes to interfaces are updated in activity diagrams
- Less modeling errors by using “specification by selection”
Modeling for Testing

- Work with complete data object values
- Enable use wildcards
- Visual indication of pre-conditions
What do Generated Tests look like?

<table>
<thead>
<tr>
<th>Step</th>
<th>Action(s)</th>
<th>Verification Point(s)</th>
<th>Verdict</th>
<th>Observations</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Configure SUT where baseUrl is &quot;esd.conformiq.com&quot;</td>
<td>Application displays a Login Screen</td>
<td>Open</td>
<td>Fill in</td>
</tr>
<tr>
<td></td>
<td>Fill out the UserInfo Form in the Login Screen where userName is &quot;Kimmo&quot;, passwd is &quot;123&quot;</td>
<td>Application displays a ESD Main Screen where Shopping cart is empty where in Item Form sku is &quot;&quot;, qty is 0, sku Text Box widget is enabled, qty Text Box widget is enabled, Add Item Button widget is enabled</td>
<td>Open</td>
<td>Fill in</td>
</tr>
<tr>
<td>3</td>
<td>Select New choice in File menu in the Login Screen</td>
<td>No errors can be observed at the SUT</td>
<td>Open</td>
<td>Fill in</td>
</tr>
</tbody>
</table>

Test case 2: Add item with sku CQ_0002 and qty 5 to shopping cart

Test case 3: Add item with sku CQ_0003 and qty 5 to shopping cart

Test case 4: Add item with sku abc and qty -1 to shopping cart

Test case 5: (1) CB: Checkout

... or VB or Java or Perl or Pyton or TTCN-3 or etc
1st Industrial Feedback on Creator

- **Doubled** productivity over conventional UML/Java based automated test design solution
- Training need reduced from 4 weeks to 4 days
- Subject Matter Experts (SMEs) and manual testers are able to model for testing
- Ecosystem from conventional automated test design approach could be reused
Case 2: Elektrobit Military radio
EB Tough VoIP Features

- Tough VoIP is a wired phone that is using UDP/IP network for connection
- Manufacturer: Elektrobit
- Main features:
  - Easy configuration
  - Point-to-Point call
  - All call
  - War-proof device
  - As simple as possible
Testing problem
Two language solution

Modeling one test case

Generating one test case

Executing the test case

Modeling a test logic

Model-Based Testing generates multiple test cases

Executing test cases

Model

MBT

TTCN-3

EB Test Tool Platform + OpenTTCN tester

TTCN-3

Generating one test case

Executing the test case
Language development

EB’s test expert, coder

Modeling, Trying, Coding

Specs + code sample

Language, example models

Change request

Language

N times CR + update...

Testing

Language developing

Language developer

Model development

Model development
Model example 1: Modeling test cases
Model example 2: Modeling for test generation
How to get started on a DSL design

• Define
  – Concepts
  – Rules
  – Symbols
  – Generators

• Focus on how you think about a problem not how you (re)solve or describe it today
  – DSLs are not effective as graphical general purpose programming languages
Experiences

• About 10 times faster with modeling
• Set-up time estimation:
  – 2 weeks for the first version
  – 1 more week for making it better

• Other benefits:
  – Visualization makes it easy to understand
  – Easy test configuration
  – Test coverage dramatically increase with MBT
  – Mass testing with MBT models
  – No special skills needed for creating test cases
Results of combining DSLs + MBT

The case studies show:

• Easier adoption
  – Better acceptance, short ramp up
• Significantly faster model development
  – Higher abstraction leads to improved productivity
  – Automation of model creation
  – Immediate feedback & guidance during model creation
• Wider model accessibility
  – Visualization makes it easier to understand
  – Domain experts can participate
  – Customers can review models!
Summary

• Classic DSLs benefits found to be applicable in testing
  – Driven by fully automatic model transformations
  – Prevent illegal model construction & enforce methodology

• Challenge: Keep DSL lean and expressive
  – Leanness yields simplicity but too lean may lead to rejection!
  – Important to use tools that enable flexibility by allowing language evolution

• We believe DSL driven MBT will establish itself as the next step in evolution of software testing
How to get started: Concepts

• What are the different object types?
  – Example: Screen, forms, widgets, messages

• What are their properties? What kind of values can they take? What is really relevant for testing?
  – Example: Dependencies between form fields? Yes
  – Example: Screen where button is located? Yes
  – Example: Pixel location of a button? No
  – Example: Underlying data base table structure? No

• What is the mapping domain concepts to concepts in the general purpose language?
  – Example: Button click maps to receiving a class
How to get started: Rules

• How many objects can exist?
  – Example: Only one starting point
• How can objects be connected?
  – Example: Only input actions can produce data
• Which property values have to be unique?
  – Example: Screen and form names
• What are valid property values?
  – Example: Only optional fields can be omitted
• When is a diagram ready for test generation?
  – Example: At least one input and verification action
How to get started: Symbols

• What type of diagrams are needed?
• Which objects are important to visualize in which diagram or at all?
  – Example: Author of a diagram
• What is the absolutely essential information important to get first understanding?
  – Example: Action has a pre-condition
• How should the information be represented?
  – Example: Symbol color, shape versus text
How to get started: Generators

• What type of information is needed to be generated?
  – Example: Code for test generation
  – Example: Model documentation
  – Example: “Live” model analysis

• In which order should objects be traversed to produce the generated code?

• How should property values be processed and converted to produce best target code?

• How to structure and modularize generator code to maximize reuse?
Thank you!

• Questions, comments, counter arguments, own experiences...

• Contact
  – Juha-Pekka Tolvanen [jpt@metacase.com]
  – www.metacase.com
  – Stephan Schulz [stephan.schulz@conformiq.com]
  – www.conformiq.com
References [1/2]

• U. Oligschläger, “Modell-gestütztes Framework für das Testen von Mess- und Automatisierungssoftware für Prüfstände der Automobilindustrie“ [in German], GI TAV#34 Report, February 2013
• Industrial presentations and tutorials at ETSI User Conferences on MBT
  – http://www.model-based-testing.de/mbtuc11/program.html
  – http://ucaat.etsi.org/2013/program.html
• MBT community http://model-based-testing.info/
• ETSI MBT Standardization
  – Summary available via www.conformiq.com
References [2/2]

- USAF, ICSE, http://dl.acm.org/citation.cfm?id=227842