Model-Based Testing of Software Product Lines

Top-Down and Bottom-Up Approach
Contents

• **Software Product Lines**
  – Product Line Design
  – Variability in Behavior Models
  – Feature Mapping
• **Model-Based Testing**
  – Models for Testing
  – Automation of Test Design
• **Model-Based Testing for Product Lines**
  – Top-Down vs Bottom-Up
  – Comparison
• **Summary**
What is a Product Line?
Software Product Lines

• Analogue to product lines from end-user products:
  – similar use,
  – similar features („look-and-feel“),
  – different pricing
• Most important for business domains where there is a necessity to offer similar but different products
• Explicit modeling of commonalities and differences within the products of a line

"A software product line (SPL) is a set of software-intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way."

CMU SEI
Building Blocks, Platform, Product Family

- «Product Blocks»

- **Product Platform:**
  Commonalities of a product line

- **Product Family:**
  Products that are based on the same platform.

Quelle: Blackenfelt 2001
Product Line Model vs. Variant Model

Product Line Model
(Domain Engineering)

Variant Model
(Application Engineering)

configure

overlap

extract

P1
P2
P3
Example of a »simple« Product Line Model

Variants of a Vehicle

- Car
  - Chassis
    - Off-Road
    - Sport
  - Transmission
    - Automatic
    - Manual
  - Engine
    - Electric
    - Gasoline
    - Diesel
    - Standard
    - ACC
      - V1
      - V2
  - Cruise Control

Mandatory: Chassis, Transmission, Engine

Optional: Cruise Control

Alternative: Off-Road, Sport

Requires: Engine

Excludes: Gasoline, Diesel

(m..n)
Case Study Example
A Toy Example
An even simpler Example: Online Shop Family

- Online Shop
  - Catalog
  - Payment
    - Bank Account
    - ECoins
    - Credit Card
  - Security
    - Low
    - High

{1..3}

Credit Card implies High
A valid Variant

Diagram:
- Online Shop
  - Catalog
  - Payment
  - Security
  - Search
    - Bank Account
    - ECoins
    - Credit Card
      - Low
      - High

- Credit Card implies High

Fraunhofer FOKUS
Behavior Model of the Product Family

150% Model
Feature Mapping

Maps features to model elements
If a feature is selected for a variant the corresponding elements in the state machine are present
A Product’s Behavior

100% Model

State machine: OnlineShop with Secure Credit Card Payment only

- Start
- OpenProductCatalog
- /listProducts
- Catalog
- Product Selection
- AddProductToCart
- ReturnToCatalog/listProducts
- ProductDetails
- ProductDetailsFor/showProductDetails
- ToCart/showCartContent
- CartContent
- RemoveProduct/showCartContent
- ProceedToCheckout
- ProceedToCheckout
- Checkout
- /showSummary
- Order
- SelectPaymentMethod
- Payment
- ChooseMethod
- SelectCreditCard
- CreditCard
- ProceedPayment/validate
- Validation
- Valid

Fraunhofer FOKUS
Model-Based Testing
Model-Based Testing

- Models can be used in several ways
  - Models for **Test Specification**, i.e., Formalization of Requirements
  - Models as the source of **Test Generation**
  - Models as **Test Objects** in model-based development
  - Models as **Test Oracles**, i.e. for evaluation of tests
Tools for Model-Based Testing

- **Test specification**
  Model creation (Editor and Syntax checker)
  Linking to Requirements-Management-Tools

- **Test generation**
  Automatic derivation of tests from models
  Selection of test data

- **Test execution**
  Connection to the target
  Test monitoring and management

- **Test evaluation**
  coverage checker, report generators,
  verification, model checker
Model-Based Testing: Automation of Test Design

Model → Tests → Test execution
Model-Based Testing: Automation of Test Design

- Feature models, domain models
- Error models, coverage criteria

Test models

Quality criteria

Test generation

Test suite

Embedded, GUI-Tests

Model Checking, Constraint Solving, Evolutionary Algorithms
Automatic Test Generation with Test Goals

Test generation

Criterion „Cover all states“

Test goal „Cover A“
Test goal „Cover B“
Test goal „Cover C“

Test generation
Model-Based Testing of Product Lines
General Approaches to MBT for SPL

When applying MBT to SPLs there is one fundamental choice:

When to bind variability?

1. Before test case design
2. After test case design
Binding Variability BEFORE Test Case Design
Product-Centric

Domain Engineering
- Requirements (incl. Variability)
  - Creation/Completion
  - Binding of Variability

Application Engineering
- Test Model (incl. Variability)
  - Test Model
  - Derivation
  - Test Cases
Top-Down
Product-Centric

Variants (100% Models) and Products are generated according to „Feature Coverage Criteria“:
• All-Features-Selected/Unselected
• Pair-/N-wise
• ...

Test cases are generated for every variant/product according to test goals:
• Transition Coverage
• MCDC
• ...
Binding Variability AFTER Test Case Generation

Domain-Centric

**Domain Engineering**

- Requirements (incl. Variability)
- Test Model (incl. Variability)
- Test Cases (incl. Variability)

**Application Engineering**

- Test Cases

Creation/Completion → Derivation → Binding of Variability
The 150% model is merged with the feature model.

Test cases are generated from the merged model. A single test case may be applicable to more than one product.

Products for testing have to be selected from the test suite according to a criterion:

- Minimal number of variants
- All variants
- ...
Comparison

Selected Coverage Criteria

• Top-Down
  • Feature Model: All-Features-Selected and –Unselected;
  • State Machine: Transition Coverage
• Bottom-Up
  • State Machine: Transition Coverage
  • Variant Coverage: minimum number of Variants

<table>
<thead>
<tr>
<th></th>
<th>Top-Down</th>
<th></th>
<th>Bottom-Up</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Manual</td>
<td>Automatic</td>
<td>Manual</td>
<td>Automatic</td>
</tr>
<tr>
<td>Variants</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Test Cases</td>
<td>2</td>
<td>18</td>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>Test Steps</td>
<td>43</td>
<td>70</td>
<td>27</td>
<td>59</td>
</tr>
</tbody>
</table>

Automatic test design by Conformiq Designer
Evaluation

• Redundancy-wise, the bottom-up approach seems to be more efficient

→ Variant selection and test generation heavily depend on the applied coverage criteria

• Weaker coverage criteria for variant selection can lead to more efficient results for Top-Down

→ Importance of a single variant for the behavior is not easy to determine.
Summary
Summary

• Theoretical considerations for efficiently testing software product lines
  – Top-Down approach (Product-Centric)
  – Bottom-Up approach (Domain-Centric)

Future Work

• Larger Examples
• Complete the tool chain
  • Retrieve a minimal number of variants from the 150% model based test cases
  • ...
• More experiments on the pros and cons of each approach
  • Is it advisable to apply strong coverage criteria on feature models or on 100% models?
  • ...