Potential Errors and Test Assessment in Software Product Line Engineering

A Mutation System for Variable Systems

Hartmut Lackner
hartmut.lackner@informatik.hu-berlin.de

Martin Schmidt
schmidma@informatik.hu-berlin.de

Graduate School METRIK
Humboldt-Universität zu Berlin

April 18th 2015

Software Product Line

Key Drivers

- Efficient engineering by planned reuse of software
- Satisfy customer demand for individualized products

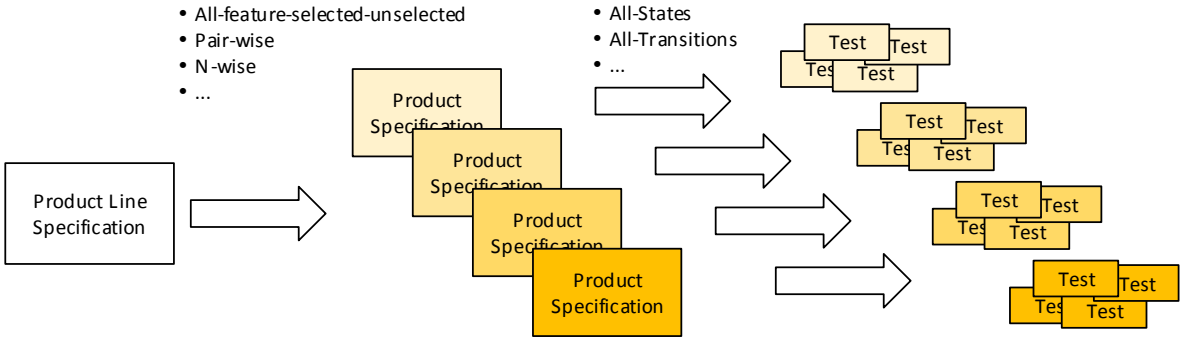
Definition: Software Product Line (SPL)

A Software Product Line 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.

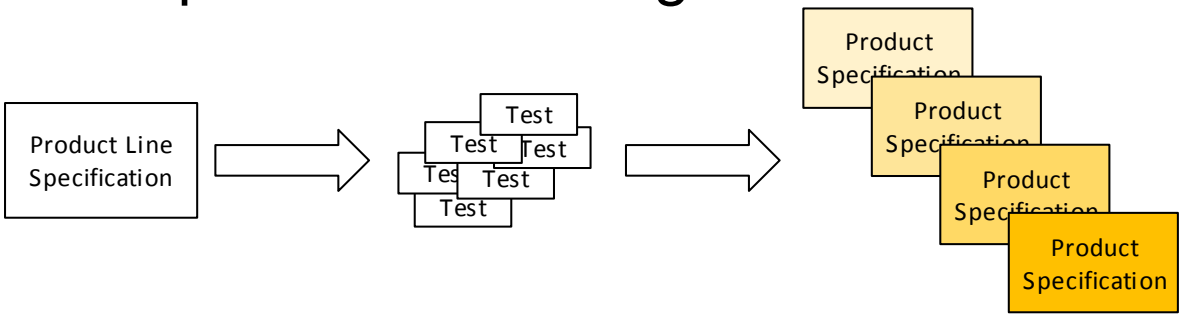
[Carnegie Mellon Software Engineering Institute]

Assessment for Product Line Tests

- Product-Centered
  1. Select products for testing from the feature model
  2. Design test cases from product models



- Product Line-Centered
  1. Design test cases from product line model
  2. Select product for testing from test cases



Model-Based Product Line Engineering

Mutation Analysis

FOUNDATION

Phases of Product Line Engineering

Domain Analysis

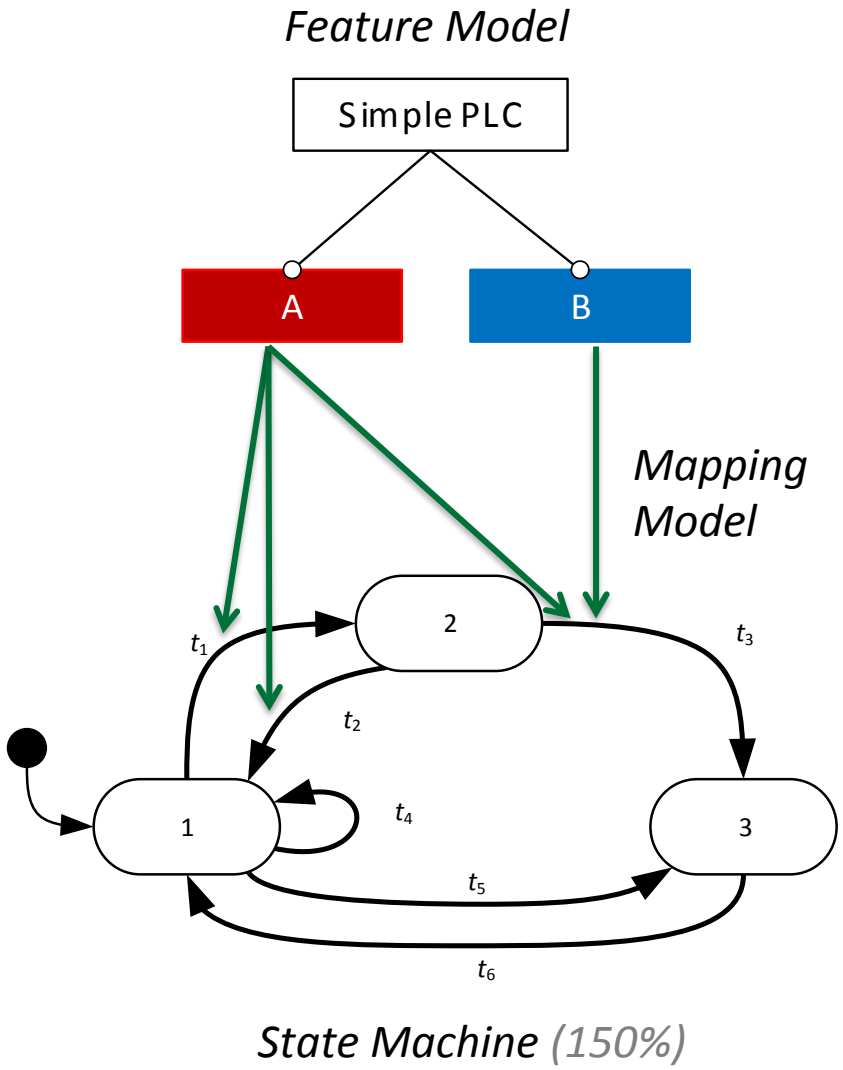
- User-Visible properties
- Represents all variants

Domain Design

- UML state charts

Domain Implementation / Feature Mapping

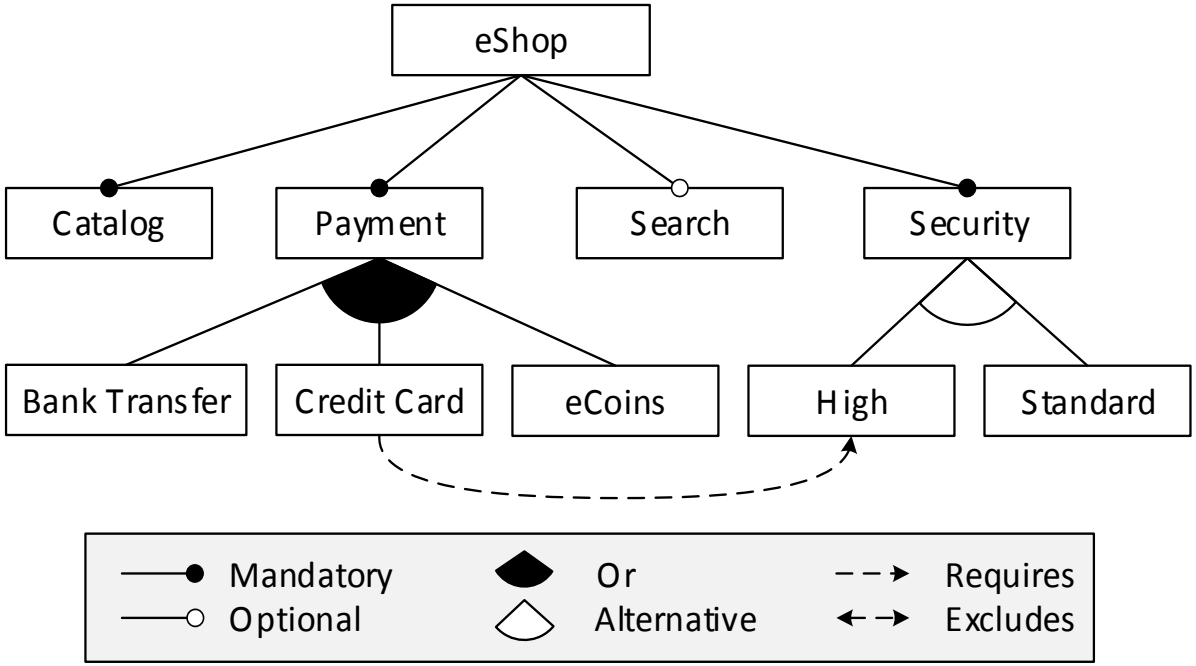
- Maps features to state machine elements
- Considers feature‘s status



Feature-Oriented Domain Analysis

Feature Model

- Features are user-visible properties
- Compact representation of all products

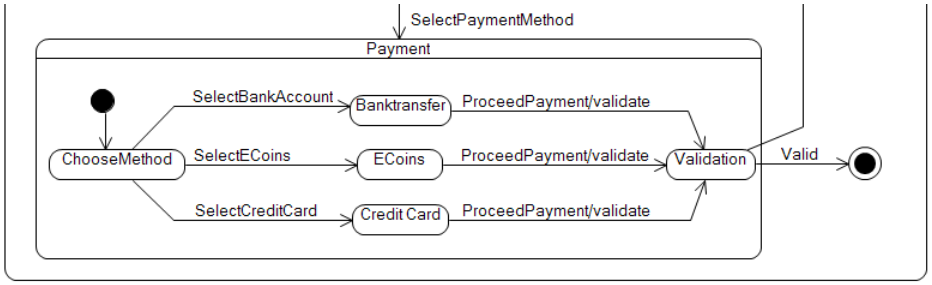


Case Study Example: e-Commerce Shop

Domain Analysis and Design

Domain Design

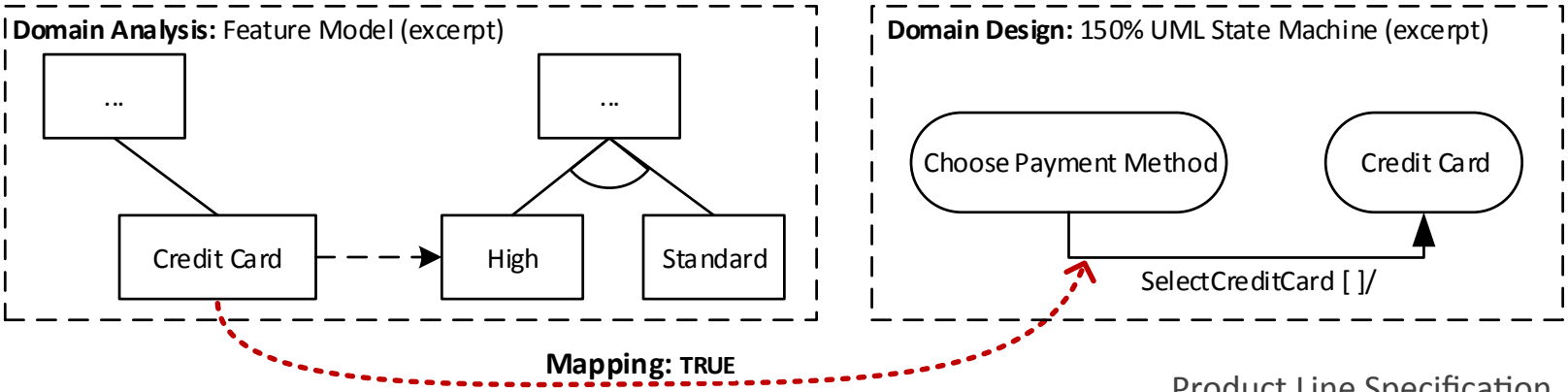
- 150% model
- UML state chart



150% model

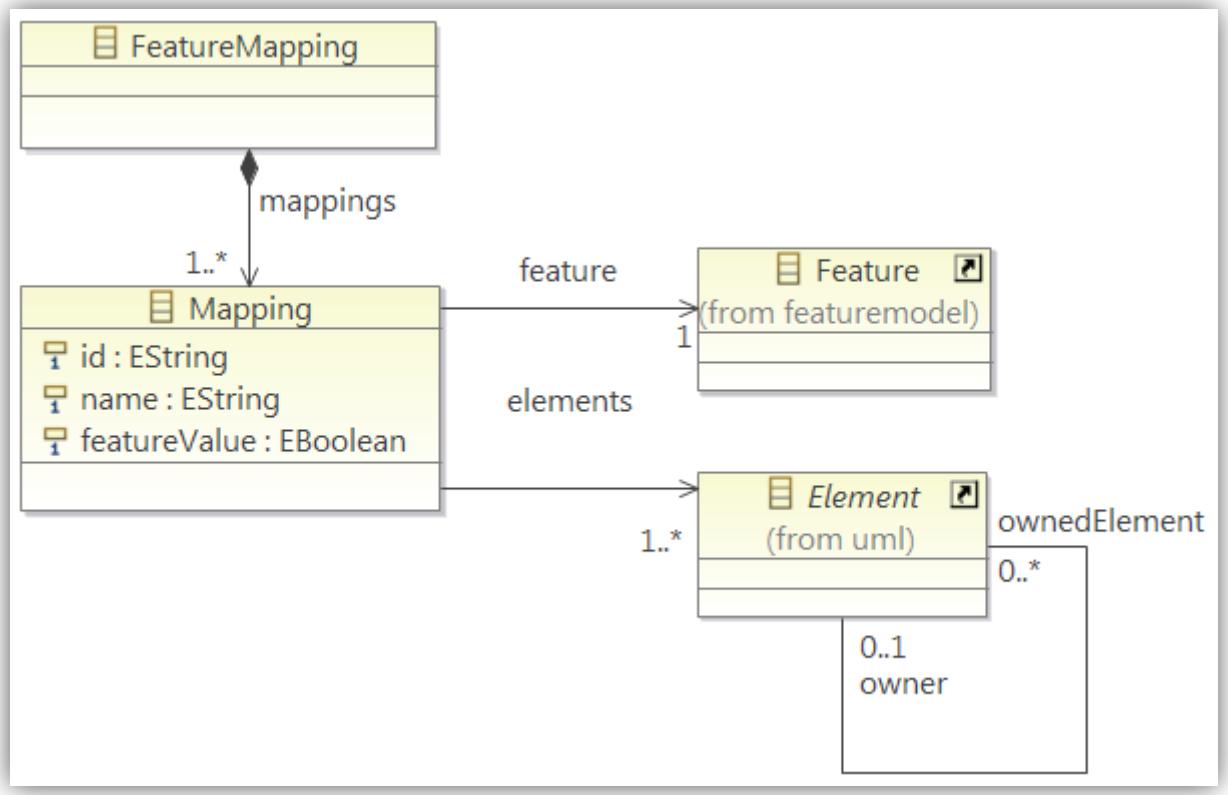
Feature Mapping

- Maps features to transitions
- Considers feature‘s status



Product Line Specification

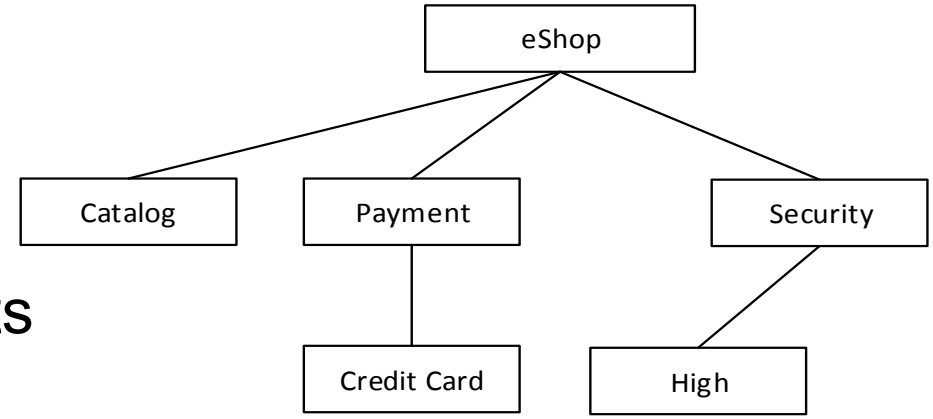
UML Class Diagram for Feature Mapping



Description of SPL Members

Configuration

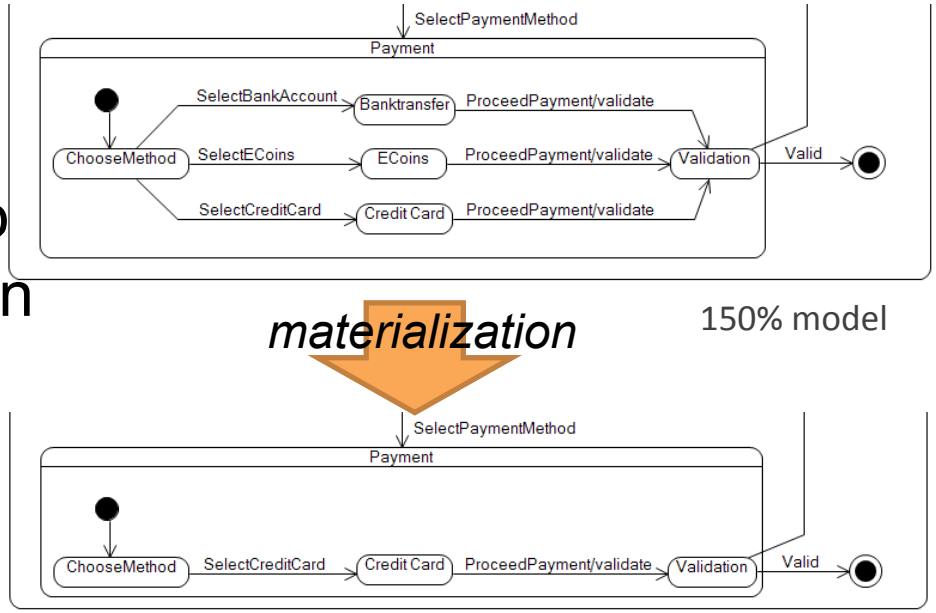
- Set of features
- Must not violate the model's constraints



Example Configuration for our e-Commerce Webshop

Materialization

- Derivation of a product
- Configuration applied to product line specification



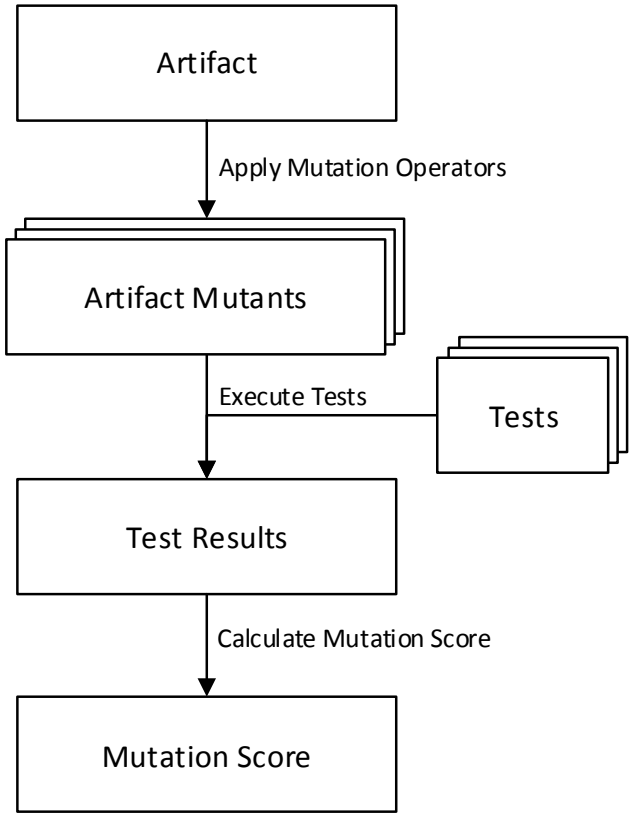
Mutation Analysis

Uses

- Assess test quality by means of fault detection capability
- Generate test data

Test Assessment

- Small *changes* to a program
- Mutated version is a *mutant*
- Failed test *kills* a mutant
- *Mutation score* = percentage of killed mutants



Traditional Mutation Process for Test Assessment

Possible errors in Model-Based Product Line Engineering

Mutation analysis

CONTRIBUTION

General Guidelines for Mutation Operators

1. Mutation categories should model *potential faults*
2. Only simple, *first-order mutants* should be generated. Only one syntactic change is applied to the original artifact.
3. Only *syntactically and semantically* legal mutants should be generated (div. by 0)
4. Do not produce *to many/equivalent* mutants.

Mutation Operators for Models

Four basic operations

1. **Insert**
   - Adds superfluous elements to the model
2. **Omit**
   - Removes a necessary element from the model
3. **Change**
   - Changes a property of an element in the model
     (e.g. changes a transition‘s target state to another state)
4. **Mix**
   - Extends, restricts, or both the behavior of affected products.

Errors in Feature Mapping (1)

Omitted mapping: a necessary mapping is left out by its entirety. Mapped elements will be part of every product unless they are restricted by other features.

Superfluous mapping: a superfluous mapping is added, such that a previously unmapped feature is now mapped to some domain model elements. This may also include adding a mapping for an already mapped feature, but with inverted feature value.

Omitting a mapped element: a mapped model element is missing from the set of mapped element in a mapping. Subsequently, a previously mapped element will not only be available in products which the said feature is part of, but also in products unrelated to this feature.

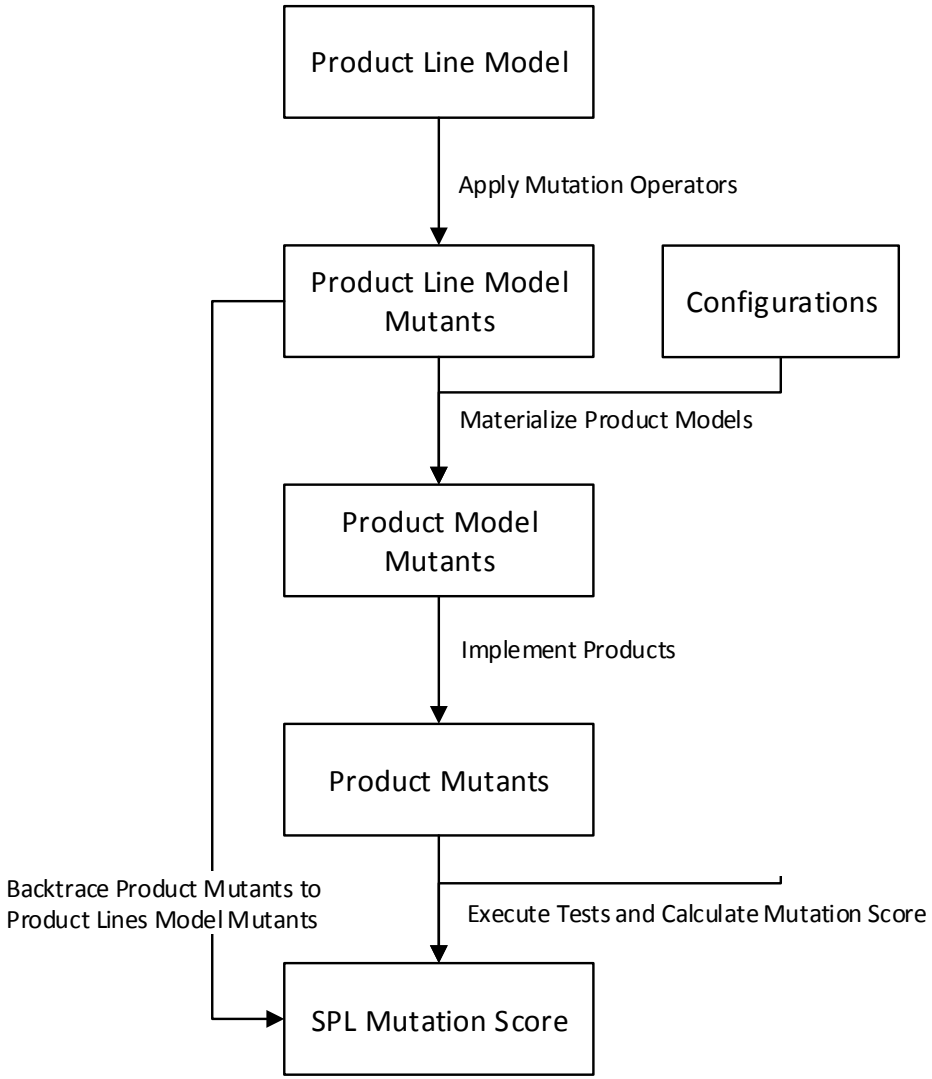
Errors in Feature Mapping (2)

Superfluously mapped element: an element is mapped although it should not be related to the feature it is currently mapped to.

Swapped feature: the associated features of two mappings are mutually exchanged. Subsequently, behavior is exchanged among the two features and thus, affected products offer different behavior than expected.

Inverted feature status: the bit-value of the feature value attribute is flipped. The mapped elements of the affected mapping become available to products where they should not be available. At the same time, the elements become unavailable in products where they should be.

Mutation System for SPLs



Operators for Mapping Models

Feature Mapping Operators

- **Delete Mapping (DMP):** Permanently enables mapped elements.
- **Delete Mapped Element (DME):** Permanently enables mapped UML element
- **Swap Feature (SWP):** Exchange a mapping‘s feature by the following feature in a given list of features
- **Insert Mapped Element (IME):** Removes UML element from all unrelated products
- **Change Feature Value (CFV):** Removes them from all related products

Invalid Mutants

- Chance of non-determinism: concurrently enabled transitions
- Equivalence: DMP, DME products including the associated feature

Operators for UML Models

UML State Chart Mutation

- **Delete Transition (DTR):** Deletes a transition from a region in an UML state machine.
- **Change Transition Target (CTT):** Changes the target of a transition to another state of the target state's region.
- **Delete Effect (DEF):** Deletes the entire effect from a transition.
- **Delete Trigger (DTI):** Deletes a transition's trigger. Only a single trigger is deleted at a time, but every trigger is deleted once.
- **Insert Trigger (ITG):** Copies an additional trigger to a transition. The trigger is copied from a another transition within the same region.
- **Delete Guard (DGD):** Deletes the entire guard of a transition
- **Change Guard (CGD):** Changes a guard’s term by exchanging operators

Invalid Mutants

- Chance of non-determinism: concurrently enabled transitions
- Deletion of initial transition
- Equivalence: deleted element was associated to a feature that is not in contained in the product

„Case Studies“ / Toy Examples

Ticket Machine

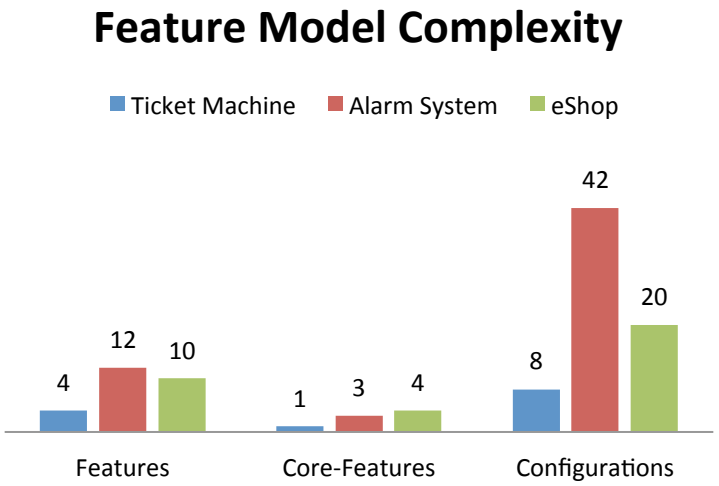
- Small model

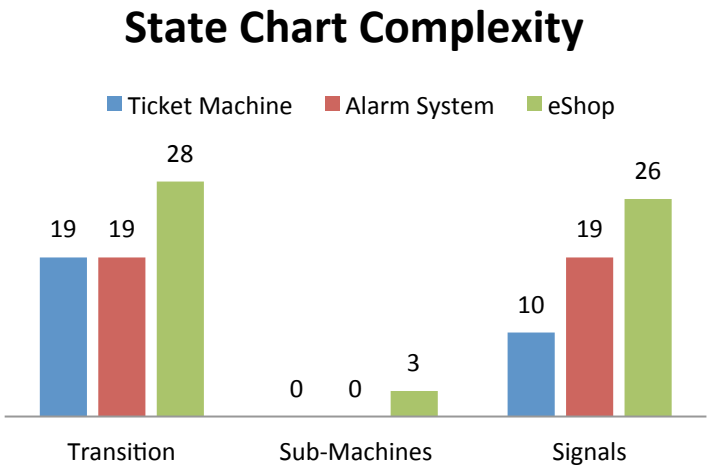
Alarm System

- More signals
- Exposes many configurations

eShop

- More complex state chart
- Still not realistic





Results: Overview

Summarized Results for Feature Mapping Operators

	eShop	TicketMach	AlarmSys
Products for testing	4	4	6
Product line mutants	30	28	53
Product mutants	96	56	278
Tests	13	9	12
Test steps	103	68	62
Tests executed	302	252	537
Failed Tests	20	30	37

Summarized Results for UML State Machine Operators

	eShop	TicketMach	AlarmSys
Products for testing	4	4	6
Product line mutants	122	148	98
Product mutants	478	296	585
Tests	13	9	12
Test steps	103	68	62
Tests executed	1553	1332	1168
Failed Tests	283	272	123

Results: Scores per Operator

Mutation Scores for Feature Mapping Operators

Op.	eShop	TicketMach	AlarmSys	Acc
DMP	0.00 (4)	0.00 (5)	0.00 (8)	0.00
DME	0.00 (14)	0.00 (8)	0.00 (21)	0.00
IME	75.00 (4)	40.00 (5)	50.00 (8)	52.94
SWP	100.00 (4)	60.00 (5)	62.5 (8)	70.59
CFV	100.00 (4)	100.00 (5)	87.50 (8)	94.12
Acc	36.67 (30)	35.71 (28)	30.19 (53)	33.33

Mutation Scores for UML State Chart Operators

Op.	eShop	TicketMach	AlarmSys	Acc
DTR	89.29 (28)	84.21 (19)	63.16 (19)	80.30
CTT	64.29 (28)	63.16 (19)	36.84 (19)	56.06
DEF	100.00 (16)	82.35 (17)	61.54 (13)	82.61
DTI	82.61 (23)	100.00 (13)	94.12 (17)	90.57
ITG	20.83 (24)	27.78 (18)	16.67 (18)	21.67
DGD	0.00 (1)	42.86 (14)	50.00 (2)	41.18
CGD	100.00 (2)	68.75 (48)	90.00 (10)	73.33
Acc	69.67 (122)	66.89 (148)	57.17 (98)	65.21

- Low scores for DMP, DME, ITG!
- Transition coverage does not detect superfluous elements
- For MC/DC and MCC holds the same
- Infectivity of sneak path analysis:
  - Product-centered: removed signals are not part of the specification anymore.
  - Product line-centered: two transition with the same trigger leave the same state.

Conclusions

- Overview of possible errors for feature models in model-based product line engineering
- We lifted mutation analysis to the product line-level
  - Mutation operators
  - Showed feasibility for three example SPLs
- Transition coverage is insufficient for SPLs
  - Accidentally enabled behavior will not detected
- Future work
  - Define model transformation for improving transition coverage
  - Assess our SPL test design methods

Potential Errors and Test Assessment in Software Product Line Engineering

A Mutation System for Variable Systems

Hartmut Lackner
hartmut.lackner@informatik.hu-berlin.de

Martin Schmidt
schmidma@informatik.hu-berlin.de

Graduate School METRIK
Humboldt-Universität zu Berlin

April 18th 2015