Technische Universität Braunschweig

ISF

Model-based Testing of Software Product Lines – Part II

Prof. Dr.-Ing. Ina Schaefer – 9 June 2015

Challenges of Testing variant-rich Software Systems

Observations:

- Complex systems with many interacting functions and features
- Many system variants and versions
- Large rate of changes, in particular in agile development processes

Consequences:

- Increasing testing effort
- Combinatorial explosion during integration and system testing
- Complete re-test in case of changes mostly infeasible

Roadmap

- Describing and Managing Variant-rich Systems
- Testing Strategies for Software Product Lines
  - Sample-based Testing of SPLs
  - Regression-based Testing of SPLs
  - Family-based Testing of SPLs

Describing and Managing Variant-rich Systems

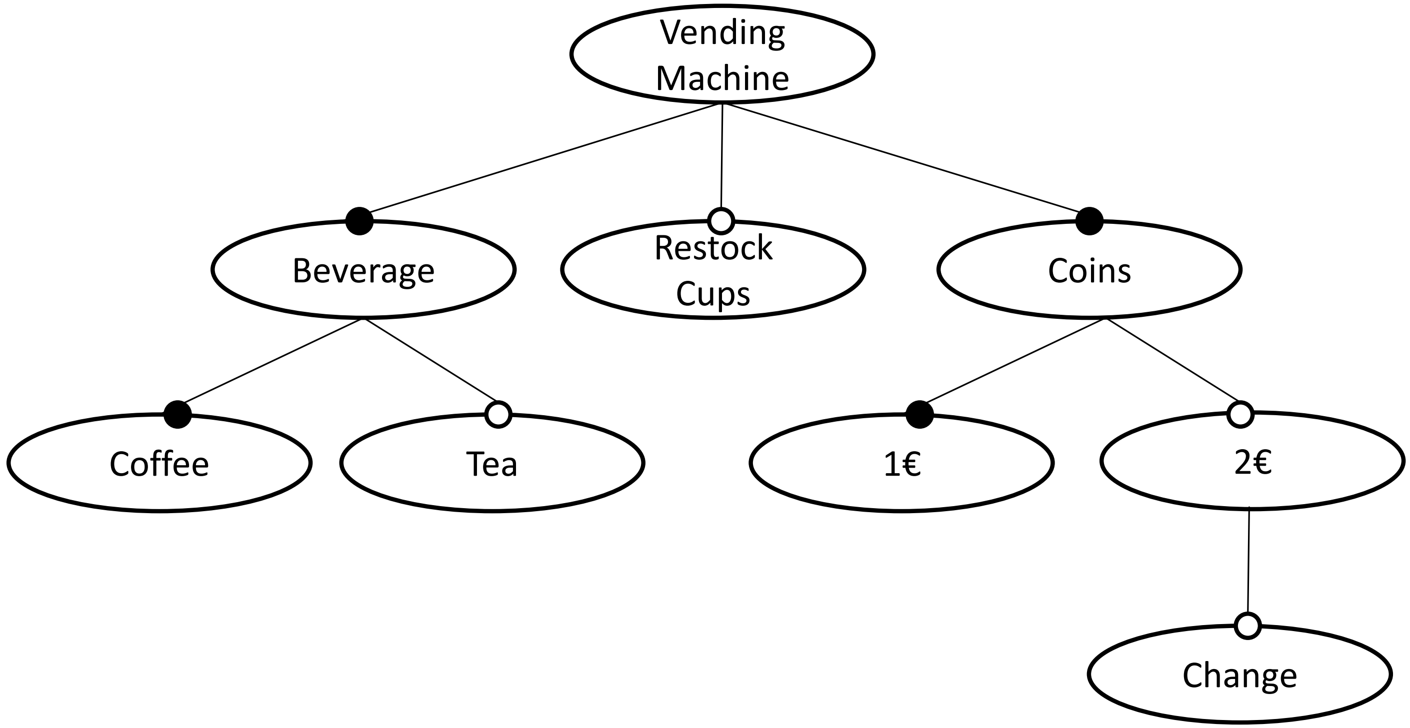
Describing and Managing variant-rich Systems

- Variant-rich systems can be described as **Software Product Lines.**
- **SPLs** are systems, which have commonalities and variabilities between each other.
- A SPL consists of several **features** which are either **mandatory** or **optional.**
- There can be further constraints between features
  - Feature A excludes feature B
  - Feature A requires feature B
  - Feature A OR feature B has to be selected
  - …

- How to describe and manage these features and there connections?

Feature Models

- Kang et al. [Kang90] introduced **Feature-Models** as possibility to represent SPLs
- FMs are **tree**-structures, which represent features and their dependencies



Feature Interactions

- A **feature** is a customer-visible product characteristic.
- Each feature in isolation satisfies its specification.
- If features are combined, the single specifications are violated. There are unwanted side effects.

→ **Feature Interaction**!

Example: Combine Fire and Water Alarms

If there is fire, start sprinkling system.

If there is water, cut the main water line.

Reasons for Feature Interactions

Intended Feature Interactions:

- Communication via shared variables: one feature writes, another feature reads values.

Unintended Feature Interactions:

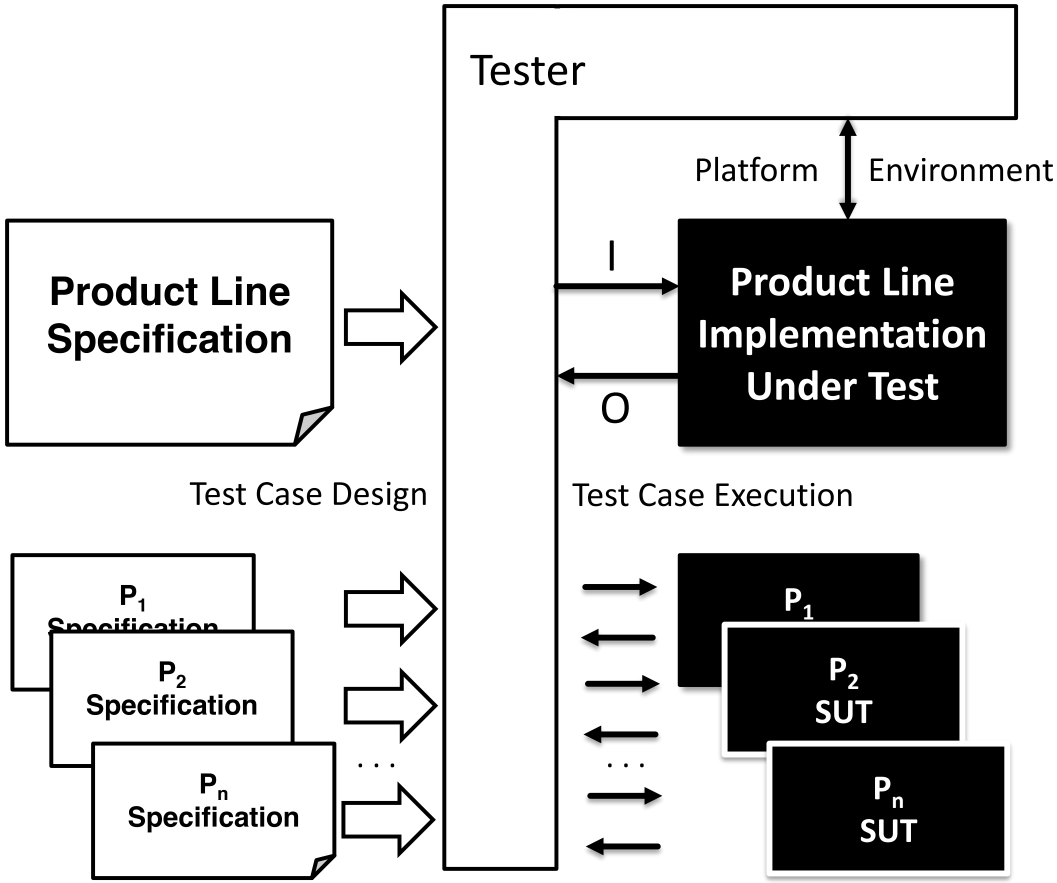
- Non-synchronized write access to shared resources, such as actuators, memory, shared variables, status flags

In general, **uncritical:**

- Shared read access to resources, e.g., sensors

SPL Testing Strategies

Software Product Line Testing



SPL Testing Strategies

Sample-based SPL testing

- Selection of representative subsets from a large set of possible variants

Regression-based SPL Testing:

- Reuse test cases and test results in order to efficiently test the selected variants

Family-based SPL Testing:

- Derive test suite from a 150%-SPL test model

Sample-based SPL Testing

Process of Sample-based SPL Testing

- **Problem:** Number of test cases growths exponentially
- **Solution:** Combinatorial Interaction Testing (CIT)

  1. Create Feature Model
  2. Generate a subset of variants based on the FM, covering relevant combinations of features
  3. Apply single system testing to the selected variants

- Efficiency of t-wise Covering Arrays (CA)
  - ➢ 1-wise CA: 50% of all errors
  - ➢ 2-wise CA: 75% of all errors → Trade-Off
  - ➢ 3-wise CA: 95% of all errors

Set Covering Problem and CAs

- S = {a,b,c,d,e} — SPL features
- M = {{a,b,c}, {b,d}, {c,d}, {d,e}} — valid product configurations
- What is the optimal Covering Array?
- **Solution:** $L = M_1 + M_4$ — minimal CA

- **Precondition:** All valid product configurations already known
  - SAT-problem, which is NP-complete
  - Fortunately, we deal with realistic FMs
- Foundation of pairwise testing…

First Solution by Chvátal (1979)

- **Idea of the algorithm:**

  1. $Set\ L = \emptyset$
  2. $If\ M_i = \emptyset, \forall\ i, i \in \{1, 2, \ldots, n\}\ END.$
     $ELSE\ find\ M', where\ \#\ of\ uncovered\ elements\ is\ max$
  3. $Add\ M' to\ L\ and\ replace\ elements\ in\ M_i\ by\ M_i - M'$
  4. $Goto\ Step\ 2$

- **Worst Case:** M contains only subsets with different elements

- Best solution not guaranteed

- Adaptation for pairwise CA generation is easy!

Adaptation to FMs and Improvements by the ICPL

```
input : arbitrary FM
output: t-wise covering array
S ← all t-tuples
while S ≠ ∅ do
    k ← new and empty configuration
    counter ← 0
    foreach tuple p in S do
        if FM is satisfiable with k ∪ p then
            k ← k ∪ p
            S ← S \ {p}
            counter ← counter + 1
        end
    end
    if counter > 0 then
        L ← L ∪ (FM satisfy with {k})
    end
    if counter < # of features in FM then
        foreach tuple p in S do
            if FM not satisfiable with p then
                S ← S \ {p}
            end
        end
    end
end
```

- **Adaptation is still slow in computation!**
- **(Selected) Improvements**
  - ➢ Finding core and dead features quickly
  - ➢ Early identification of invalid t-sets
  - ➢ Parallelization
  - ➢ and several more…

Vending Machine and ICPL runtimes

- VM has 12 valid variants
- t = 2, ICPL calculates CA of size 6
- 50% testing time saved

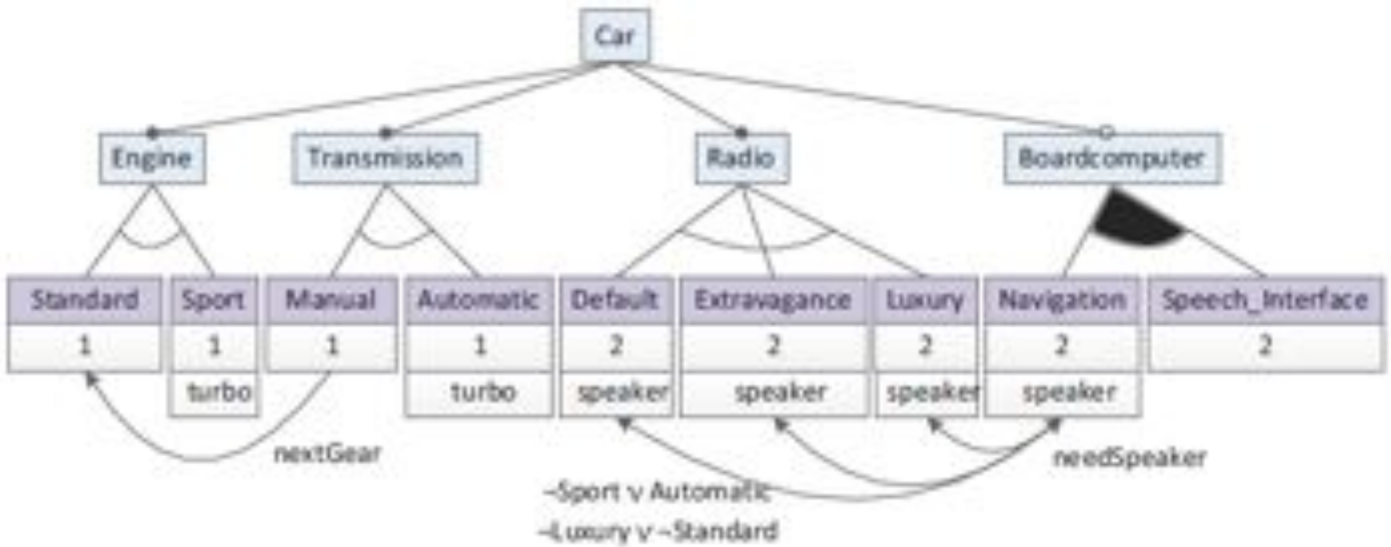
Feature\Product	0	1	2	3	4	5
Coffee	X	X	X	X	X	X
Beverage	X	X	X	X	X	X
2€		X	X	X	X	
Change		X			X	
Tea		X	X			X
Restock Cups		X		X		X
1€	X	X	X	X	X	X
Coins	X	X	X	X	X	X
Vending Machine	X	X	X	X	X	X

- ICPL can handle large-scale SPLs
- 2-wise with „normal" hardware possible
- Easily over 90% variant reduction
- Even with ICPL: Calculation time can be several hours

Feature Model	Features	Constraints	2-wise size	2-wise time (s)
2.6.28.6-icse11.dimacs	6,888	187,193	480	33,702
freebsd-icse11.dimacs	1,396	17,352	77	240
ecos-icse11.dimacs	1,244	2,768	63	185
Eshop-fm.xml	287	22	21	5

Feature Annotations for More Efficient Combinatorics

- Annotate features with shared resources, communication links, testing priorities
- Use additional information for combinatorial testing
- **Consequence**: Even lesser variants to test and shorter computation time

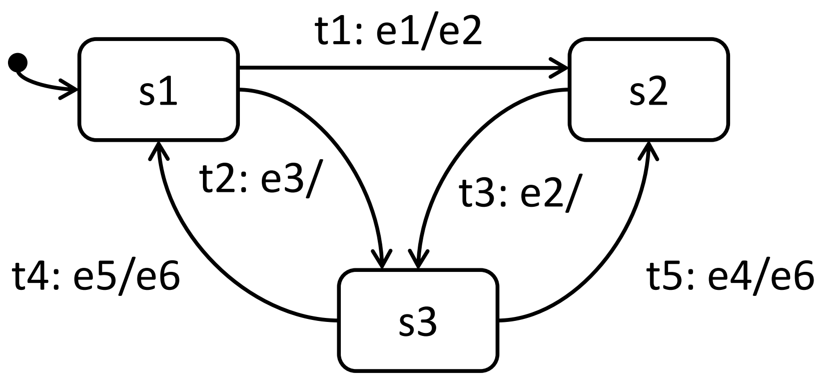


Kowal, M., Schulze, S., Schaefer, I.: Towards Efficient SPL Testing by Variant Reduction. In: VariComp. pp. 1–6. ACM (2013)

Regression-based SPL Testing

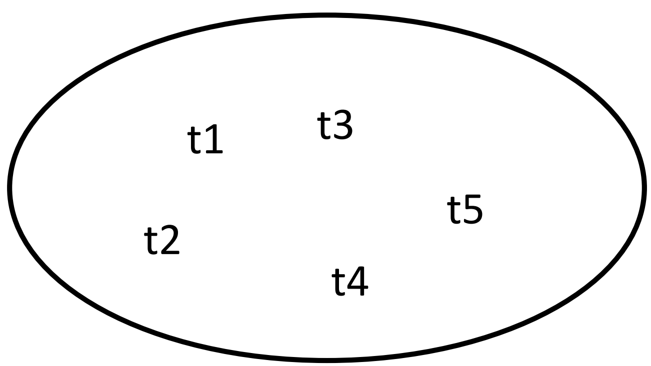
Model-based Testing - Procedure

Test Model **TM**



Coverage Criterion:
e.g., all transitions **C1**

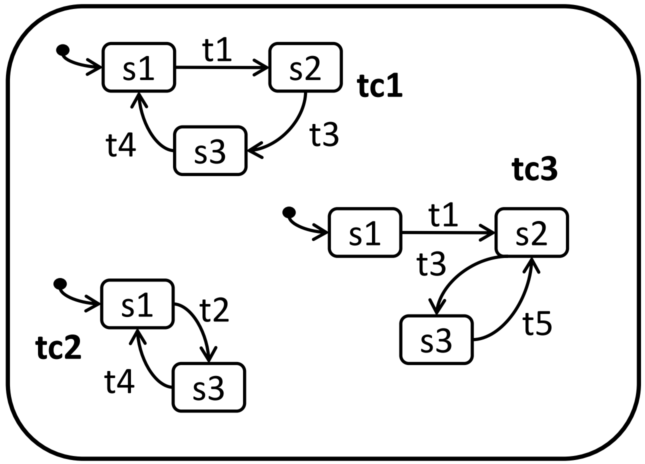
Test Goals **TG**



Model-based Testing – Procedure (2)

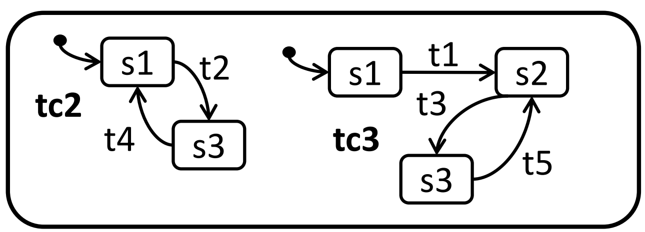
Test Case Generation

Test Suite **TS**

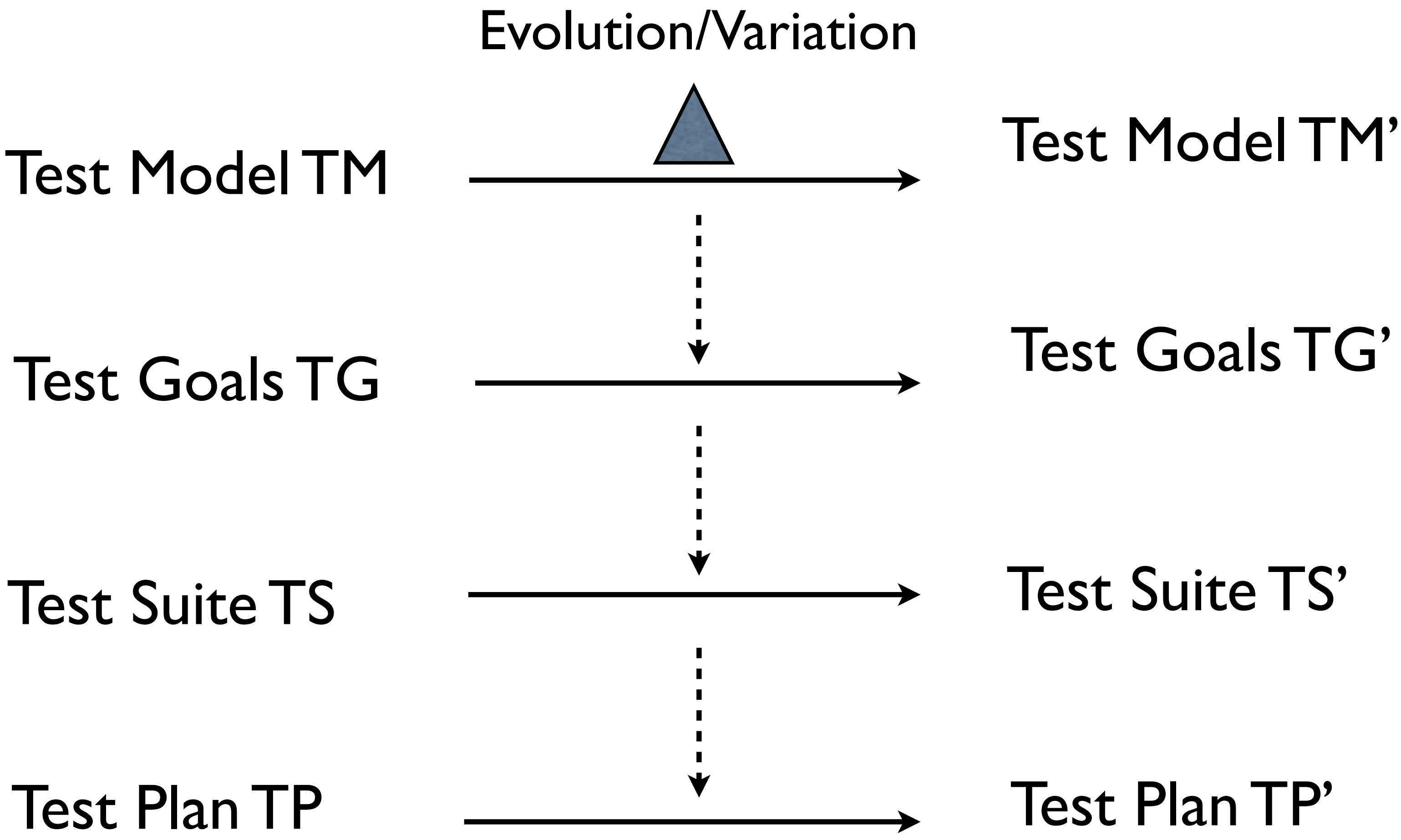


Test Selection

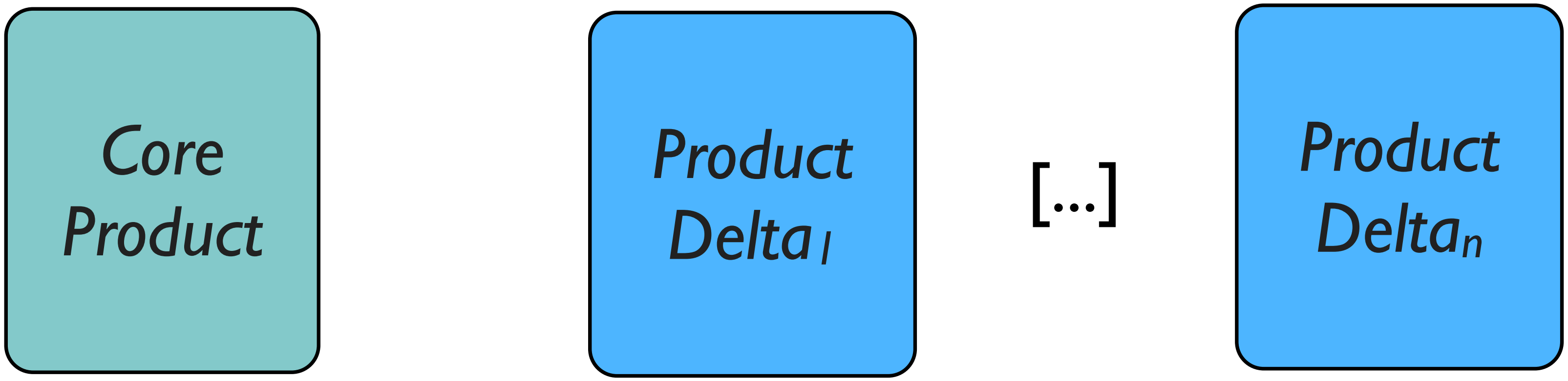
Test Plan **TP**



Incremental Model-based Testing



Delta-Modeling of Variant-Rich Systems



- Product for valid feature configuration.
- Developed with Standard Techniques

- Modifications of Core Product.
- Application conditions over product features.
- Partial ordering for conflict resolution.

Delta-Modeling - Background

Instances of Delta-Languages:

- Software architectures (Delta-MontiArc)
- Programming languages (Delta-Java)
- Modeling languages (Delta-Simulink, Delta-State Machines, Deltarx)

Advantages of Delta-Modeling:

- Modular and flexible description of change
- Intuitively understandable and well-structured
- Traceability of changes and extensions
- Support for proactive, reactive and extractive SPLE

Delta-oriented Testing approaches

- Based on delta languages and modeling techniques, different testing approaches can be defined [Lity13]

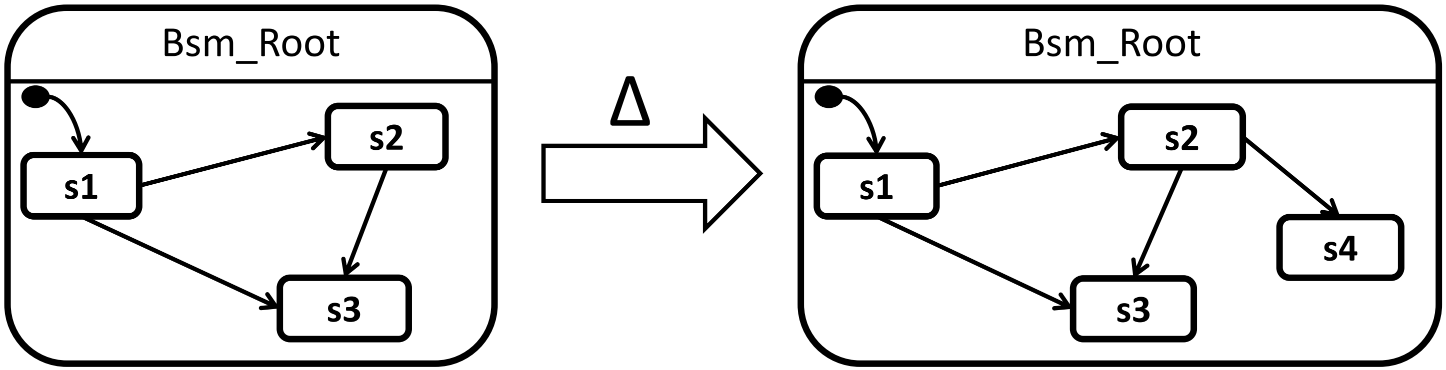
- **Goal**: Reduce regression testing effort by only testing differences between products and not every product as a whole

- **Deltas on variable test-models:**
  - Statemachines
  - Architectures
  - Activity Diagrams

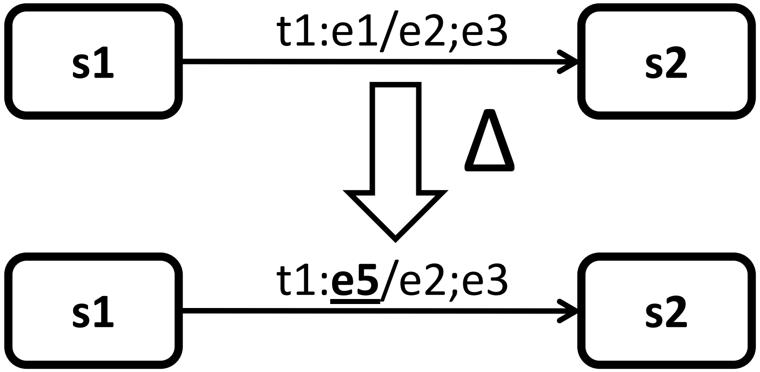
- **Deltas on requirements** in natural language

Delta-oriented Test Models (Examples)

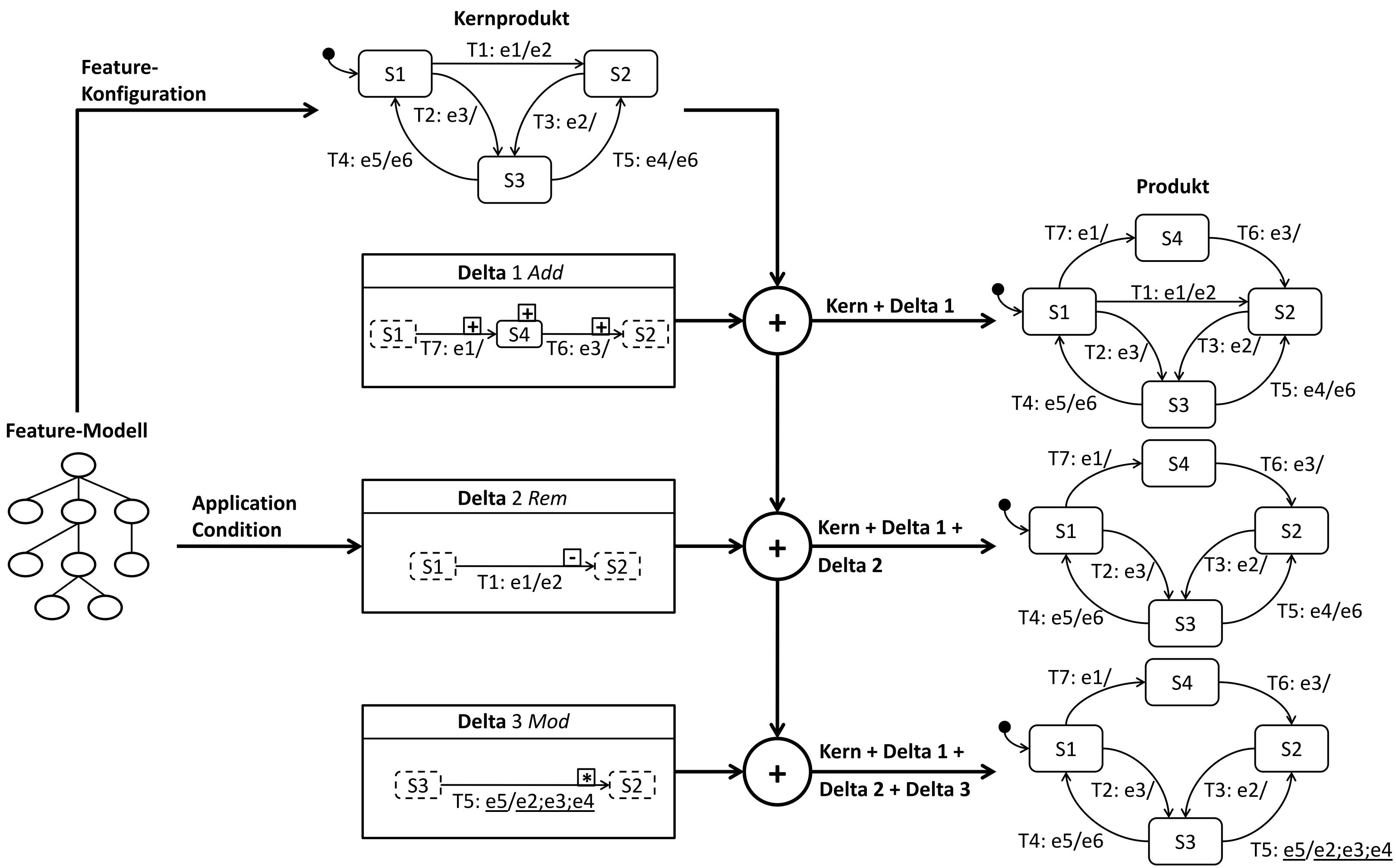
Adding a state to a State Machine:



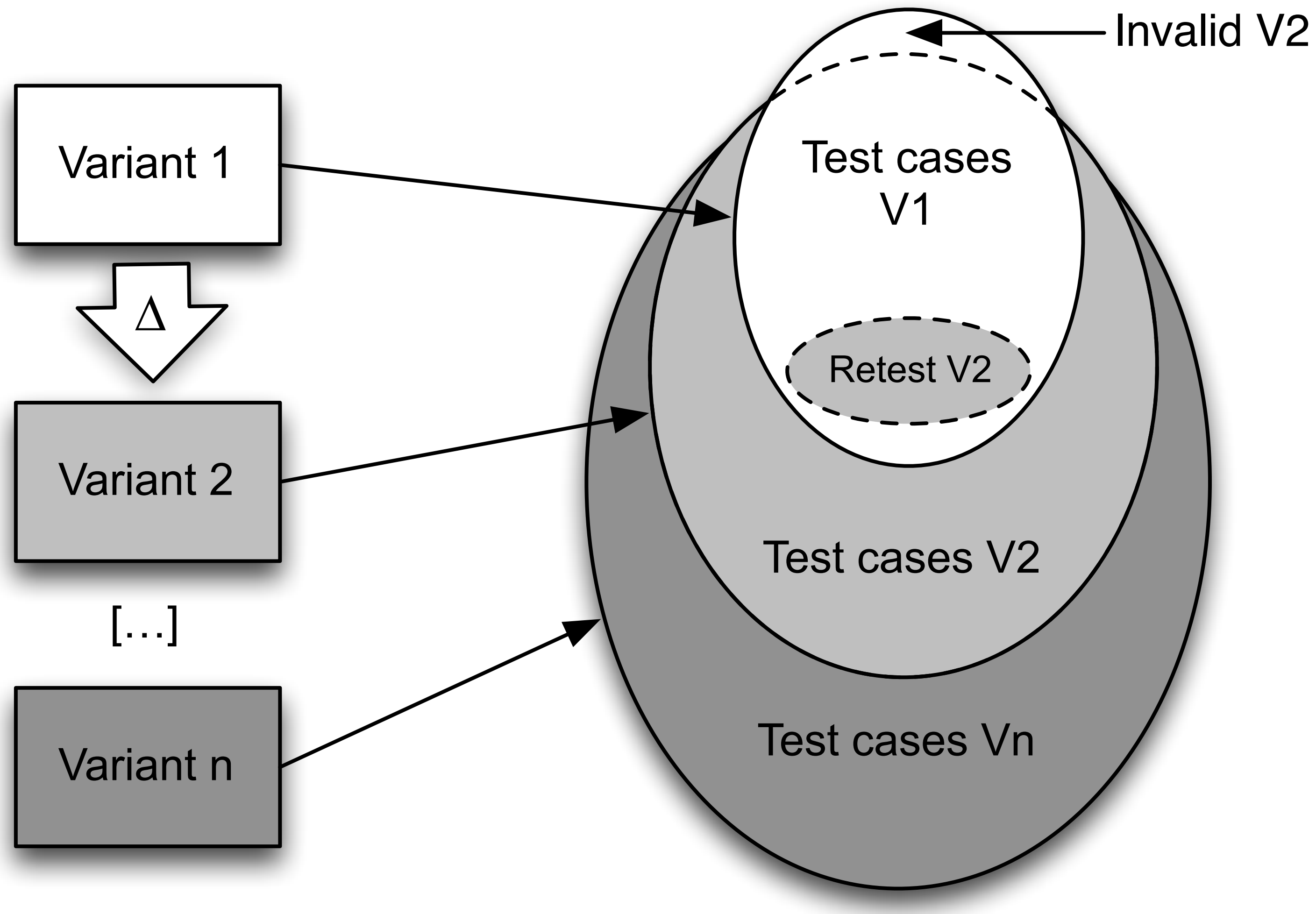
Changing the transition labels:



Delta-oriented Test Modeling



Classification of Test Cases by Delta-Analysis



Delta Testing - Procedure

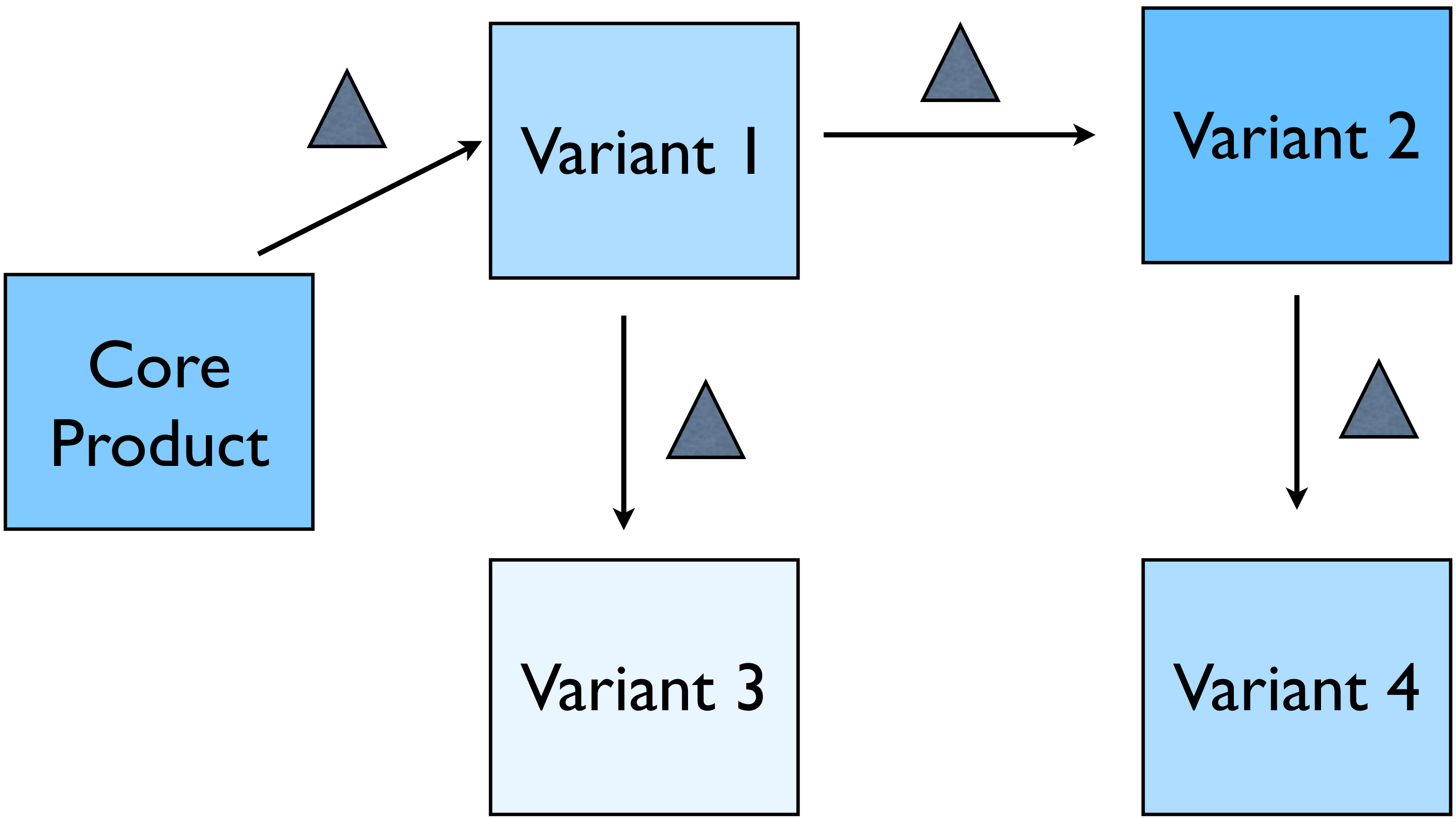
0. Fully test first product variant

1. Generate test cases for subsequent variants
   - Still valid and reuseable test cases?
   - Invalid test cases?
   - New test cases?

2. Selection of test cases by delta analysis:
   - Always test new test cases
   - Select subset of reuseable test cases for re-test

3. Optionally minimize resulting test suite by redundancy elimination

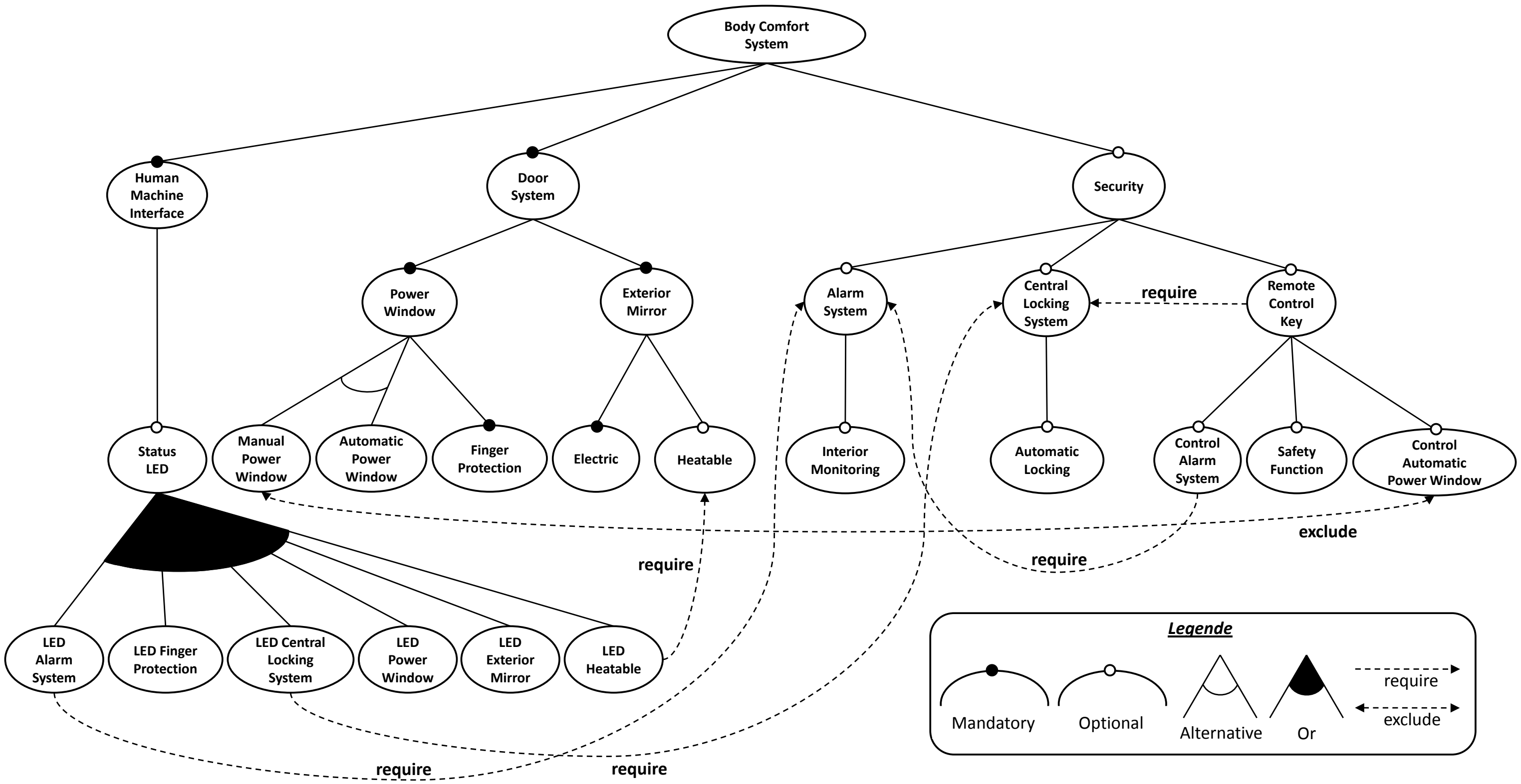
Delta-Testing Strategy



Case Study – Body Comfort System 2

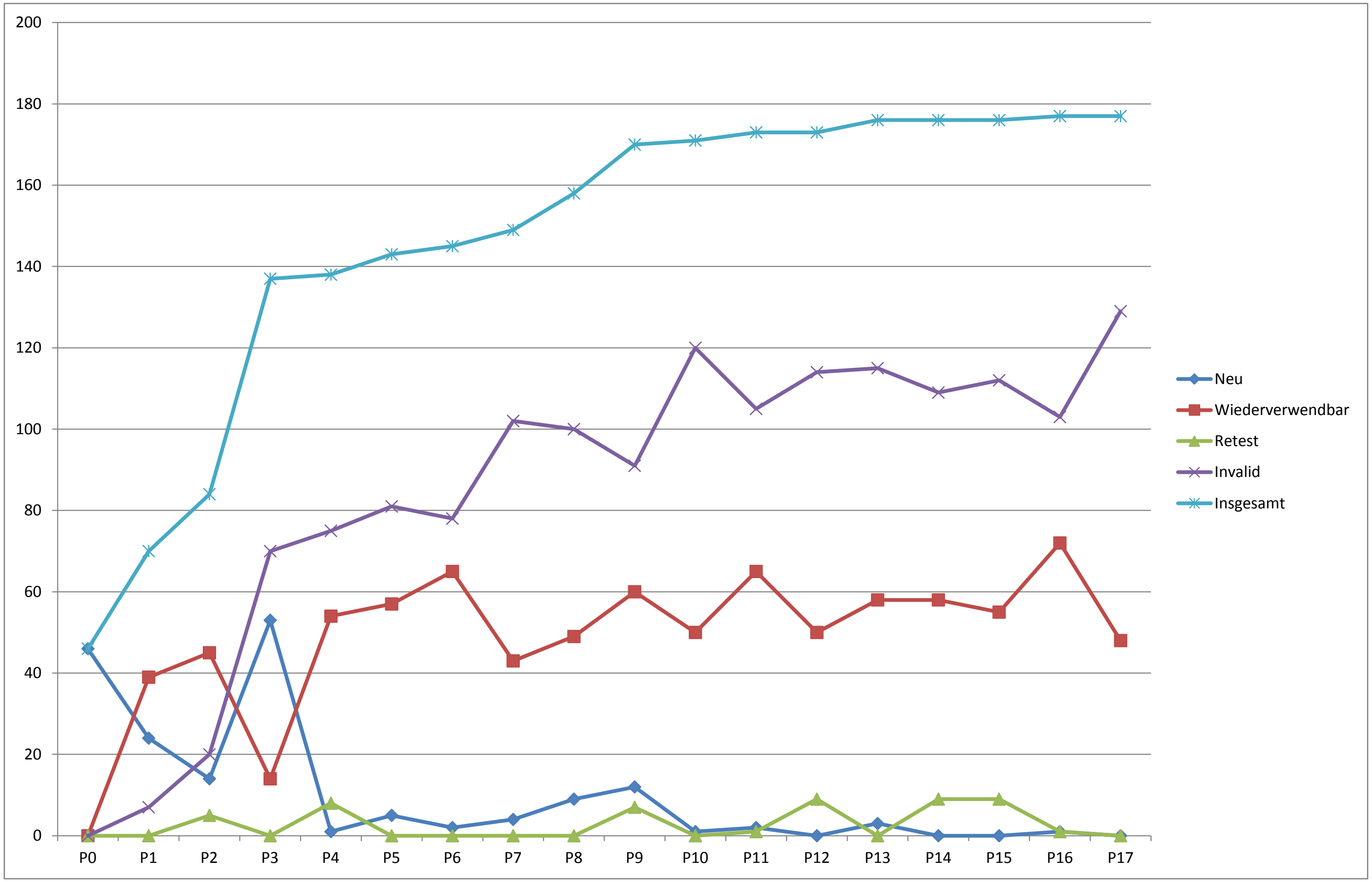
28 Features, 11616 Product Variants, 1 Core Product, 40 Deltas

16 Products for Pair-Wise Feature Coverage

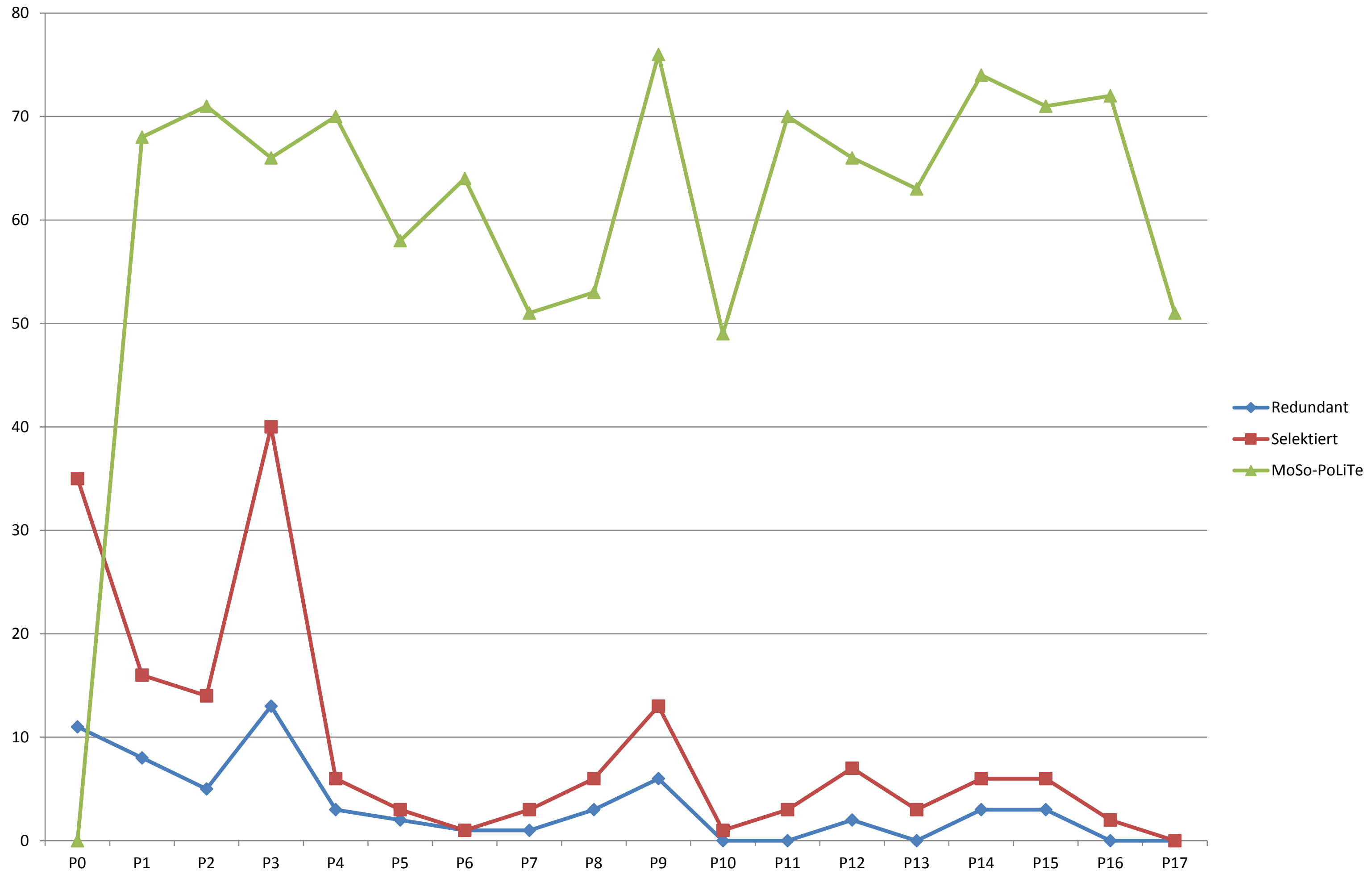


For more information see [BCS12]

Case Study BCM 2 – Delta-Testing Results



Case Study BCM 2 – Delta-Testing Results (2)



Requirements-Based Delta-oriented Testing

Requirements

BCS_R1
If an object is detected in the window (window pressure P > threshold), activate the finger protection to prevent the power window from moving any further.

BCS_R2
If the central locking system is activated and the power window is not in the top position, move the power window up, until it reaches the top position.

BCS_R3
If the move down button for the power window is pressed and there is no Central locking system, move the power Window down. Otherwise, only move down if the central locking system is deactivated

BCS_R3V1
without CLS

BCS_R3V2
with CLS

BCS_R4
After the move up button has been tapped shortly (< 1 sec), the power window moves automatically up until it reaches the top position and then the movement stops.

...

BCS_Rn

Test cases

BCS_TC1
Precondition: Window is open and an object is within the window
Action: Press move up button
Expected Result: Window moves up, until it reaches the objects and stops

BCS_TC2
Precondition: CLS is activated & power window is not in top position
Action: Press move up button
Expected Result: Power window moves to the top position and stops

BCS_TC3
Precondition: No CLS installed
Action: Press move down button
Expected Result: Power window moves to the bottom position and stops

BCS_TC4
Precondition: CLS installed and deactivated
Action: Press move down button
Expected Result: Power window moves to the bottom position and stops

BCS_TC5
Precondition: Power window is at bottom position
Action: Press move up button for less then 1 second
Expected Result: Power window moves to the top position and stops

...

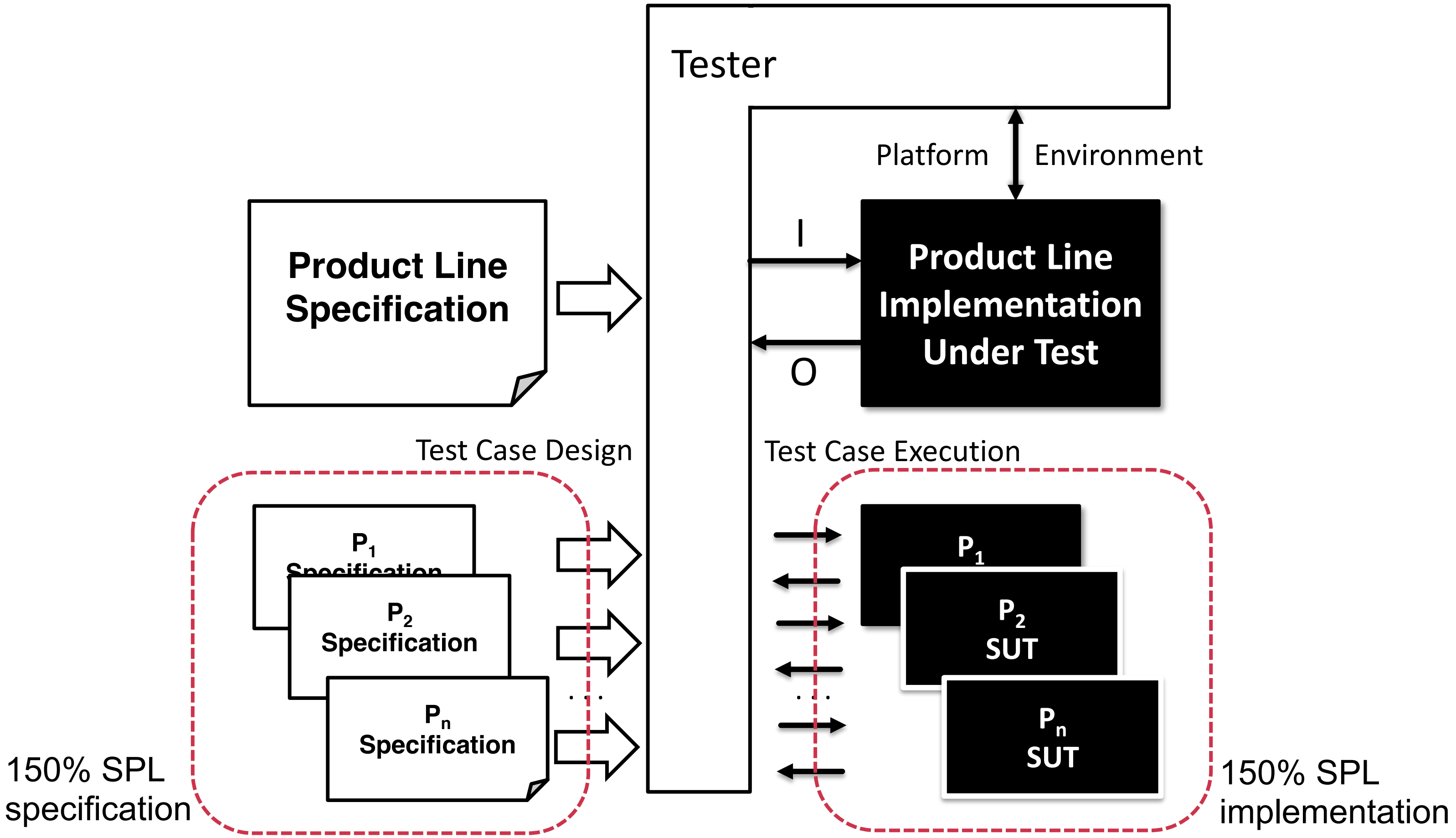
BCS_TCn

Possible Strategies for Re-Test Selection

- Manually by test engineer
- (Semi-)Automatical classification of test cases into variants
- Formulation of requirements in delta-sets with linking of test cases to requirements
- Model-based impact analysis of changes by delta analysis

Family-based SPL Testing

Software Product Line Testing

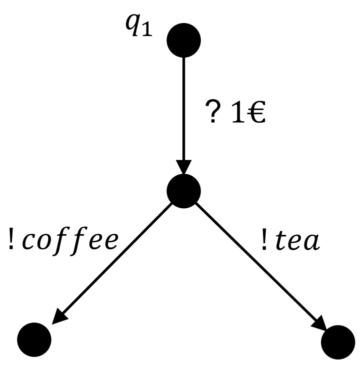


Meaning of Specifications

Implementation freedom in single system IOCO testing

- The implementation must show **at least one** specified output behavior for specified input behaviors

- The implementation may show **arbitrary output behaviors** for unspecified input behaviors

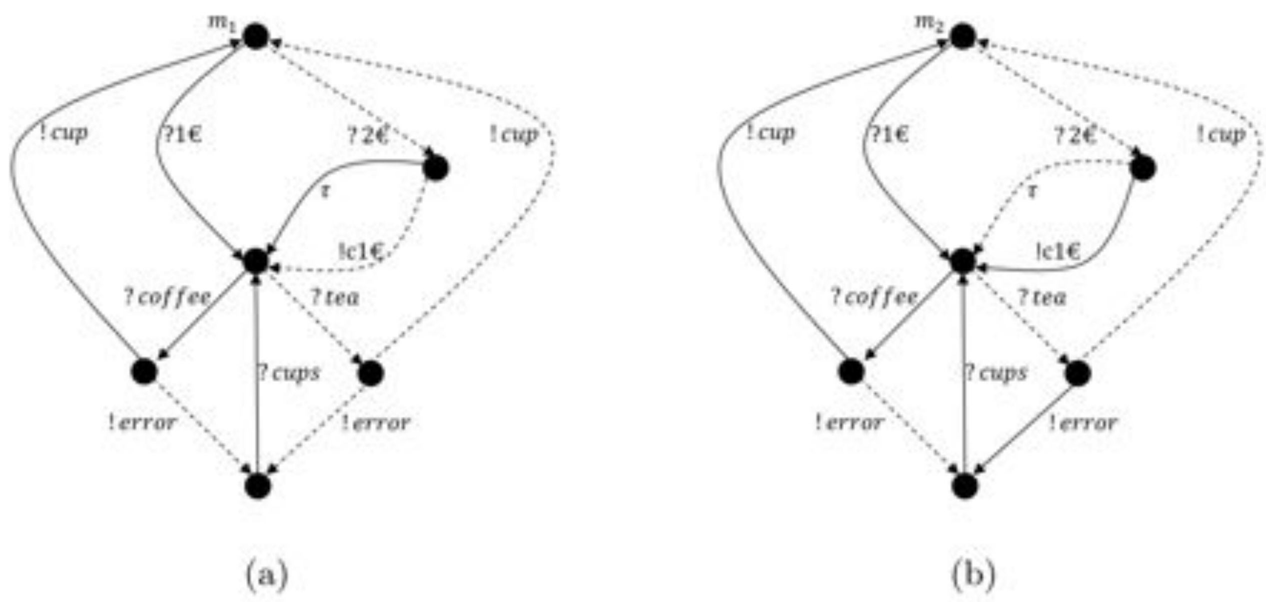


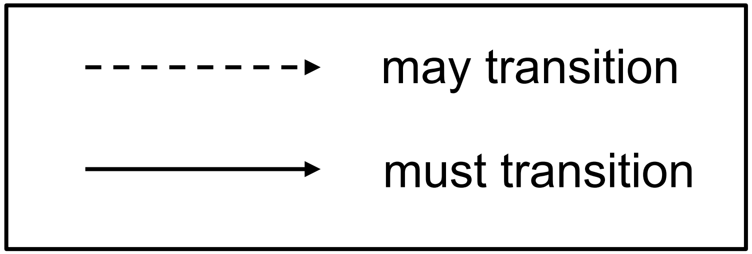
Implementation variability in SPL IOCO testing

- Distinction between **mandatory** and **possible** input/ouput behaviors

- SPL specification with explicit transition **modality**

Modal I/O Transition Systems





I/O MTS

Modal I/O Labeled Transitionsystem: $(\boldsymbol{Q}, \boldsymbol{q_0}, \boldsymbol{I}, \boldsymbol{U}, \rightarrow_{\diamond}, \rightarrow_{\square})$, where

- $\boldsymbol{Q}$ is a countable set of states,
- $\boldsymbol{q_0} \in \boldsymbol{Q}$ is the initial state,
- $\boldsymbol{I}$ and $\boldsymbol{U}$ are disjoint sets of input actions and output actions,
- $\rightarrow_{\diamond} \subseteq \boldsymbol{Q} \times \boldsymbol{act} \times \boldsymbol{Q}$ is a labeled may-transition relation, and
- $\rightarrow_{\square} \subseteq \boldsymbol{Q} \times \boldsymbol{act} \times \boldsymbol{Q}$ is a labeled must-transition relation.

Syntactical Consistency of Transition Modality

Mandatory behaviors are always also possible:
$\rightarrow_{\Box} \subseteq \rightarrow_{\Diamond}$

Modal Trace Semantics

The set of modal traces of an MTS m is defined as

$$Tr_\gamma(m) := \{\sigma \in (I \cup U)^* \mid \exists s \in Q : q_0 \overset{\sigma}{\Longrightarrow}_\gamma s\} \quad \text{where} \quad \gamma \in \{\Box, \Diamond\}$$

From syntactical consistency of MTS it follows that

$$Tr_\Box(s) \subseteq Tr_\Diamond(s)$$

Modal IOR

Modal I/O Conformance holds iff

- all **possible** behaviors of a product line implemenation are **allowed** by the specification
- all **mandatory** behaviors of a product line implementation are **required** by the specification

Modal Refinement I/O Conformance holds iff the product line implementation shows

- **at least** all **mandatory** behaviors
- **at most** all **allowed** behaviors

of the product line specification.

Modal IOCO

Let s,i be an MTS, where i is may-input-enabled.

i **mioco** s $:\Leftrightarrow$

1. $\forall \sigma \in Straces_{\Diamond}(s) : Out_{\Diamond}(i \textbf{ after}_{\Diamond}\ \sigma) \subseteq Out_{\Diamond}(s \textbf{ after}_{\Diamond}\ \sigma)$ *and*
2. $\forall \sigma \in Straces_{\Box}(i) : Out_{\Box}(i \textbf{ after}_{\Box}\ \sigma) \subseteq Out_{\Box}(s \textbf{ after}_{\Box}\ \sigma)$.

i $\textbf{mioco}_{\leqslant}$ s $:\Leftrightarrow$

1. $\forall \sigma \in Straces_{\Diamond}(s) : Out_{\Diamond}(i \textbf{ after}_{\Diamond}\ \sigma) \subseteq Out_{\Diamond}(s \textbf{ after}_{\Diamond}\ \sigma)$ *and*
2. $\forall \sigma \in Straces_{\Box}(i) : Out_{\Box}(s \textbf{ after}_{\Box}\ \sigma) \subseteq Out_{\Box}(i \textbf{ after}_{\Box}\ \sigma)$.

Conclusion

- Describing and Managing Variant-rich Systems
- Testing Strategies for Software Product Lines
  - Sample-based Testing of SPLs
  - Regression-based Testing of SPLs
  - Family-based Testing of SPLs

Literature

- [BCS12] - S. Lity, R. Lachmann, M. Lochau, I. Schaefer: *Delta-oriented Software Product Line Test Models - The Body Comfort System Case Study*, Technische Universität Braunschweig, 2012

- [Kang90] - Kyo C. Kang, Sholom G. Cohen, James A. Hess, William E. Novak, A. Spencer Peterson - *Feature-Oriented Domain Analysis (FODA) Feasibility Study,* Technical Report, 1990

- [Lity13] - S. Lity, R. Lachmann, M. Lochau, M. Dukaczewski, I. Schaefer: *Delta-orientiertes Testen von variantenreichen Systemen*, ObjektSpektrum, 2013

- Malte Lochau, Sven Peldszus, Matthias Kowal, Ina Schaefer: Model-Based Testing. SFM 2014: 310-342