Technische Universität Braunschweig

ISF

How to test the Universe?

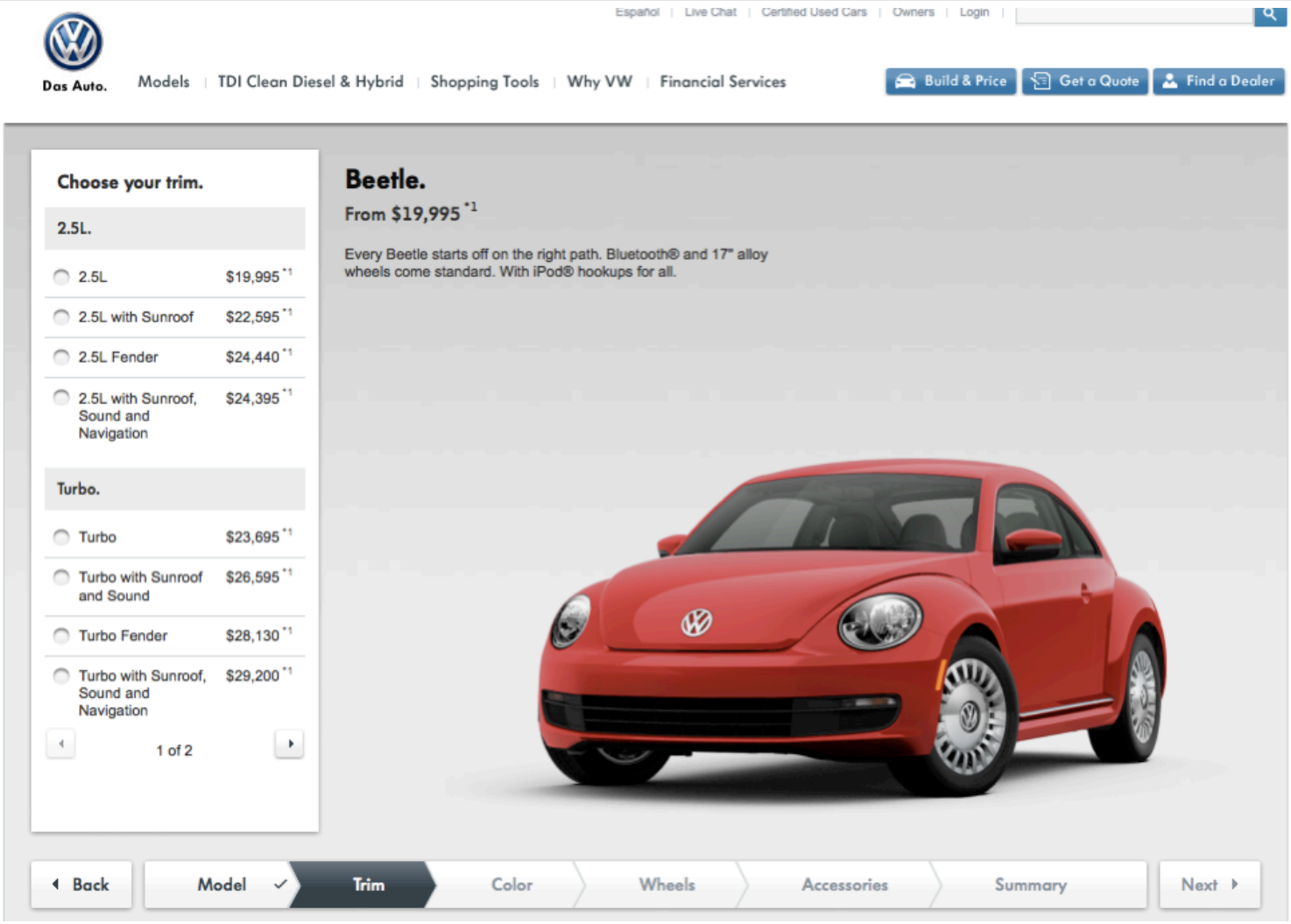
Ina Schaefer

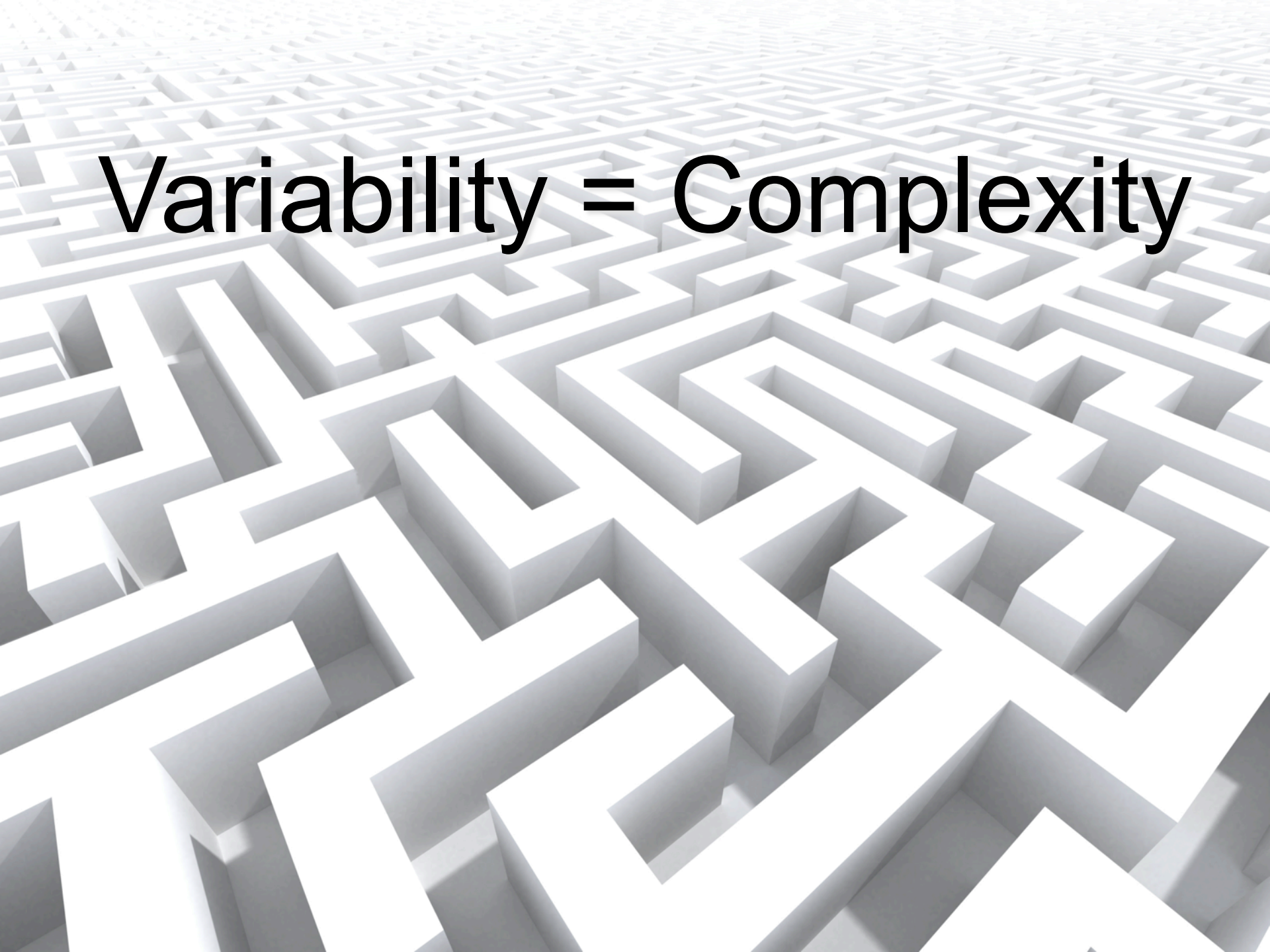
Nederlandse Testdag, 20 November 2019

“Any customer can have a car painted any color that he wants so long as it is black.”

Henry Ford (1909)

Variant-rich Software Systems – Modern Cars




Variability = Complexity

33 Features

Optional, independent

...one variant for each person on earth...

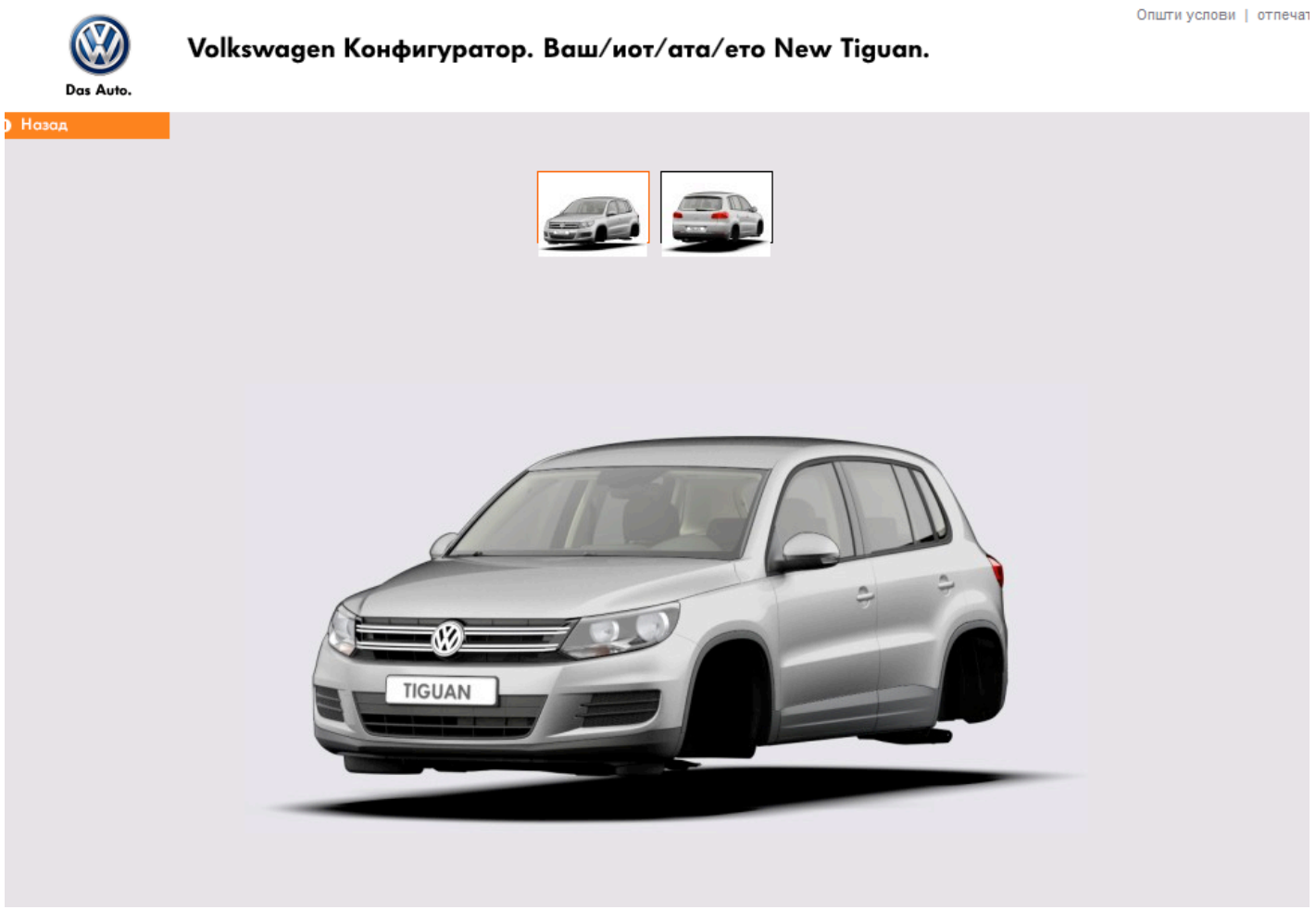
optionale, unabhängige

320 Features

... More variants than atoms in the universe ...


Correctness?
NCR
PAGE_FAULT_IN_NONPAGED_AREA
If this is the first time you've seen this Stop error screen,
restart your computer. If this screen appears again, follow
these steps:
Check to make sure any new hardware or software is properly installed.
If this is a new installation, ask your hardware or software manufacturer
for any windows updates you might need.
If problems continue, disable or remove any newly installed hardware
or software. Disable BIOS memory options such as caching or shadowing.
If you need to use Safe Mode to remove or disable components, restart
your computer, press F8 to select Advanced Startup options, and then
select Safe Mode.
Technical information:
*** STOP: 0x00000050 (0x800005F2,0x00000000,0x804E83C8,0x00000000)
Beginning dump of physical memory
Physical memory dump complete.
Contact your system administrator or technical support group for further
assistance.
1
2 ABC
3 DEF
CANCEL
CLEAR

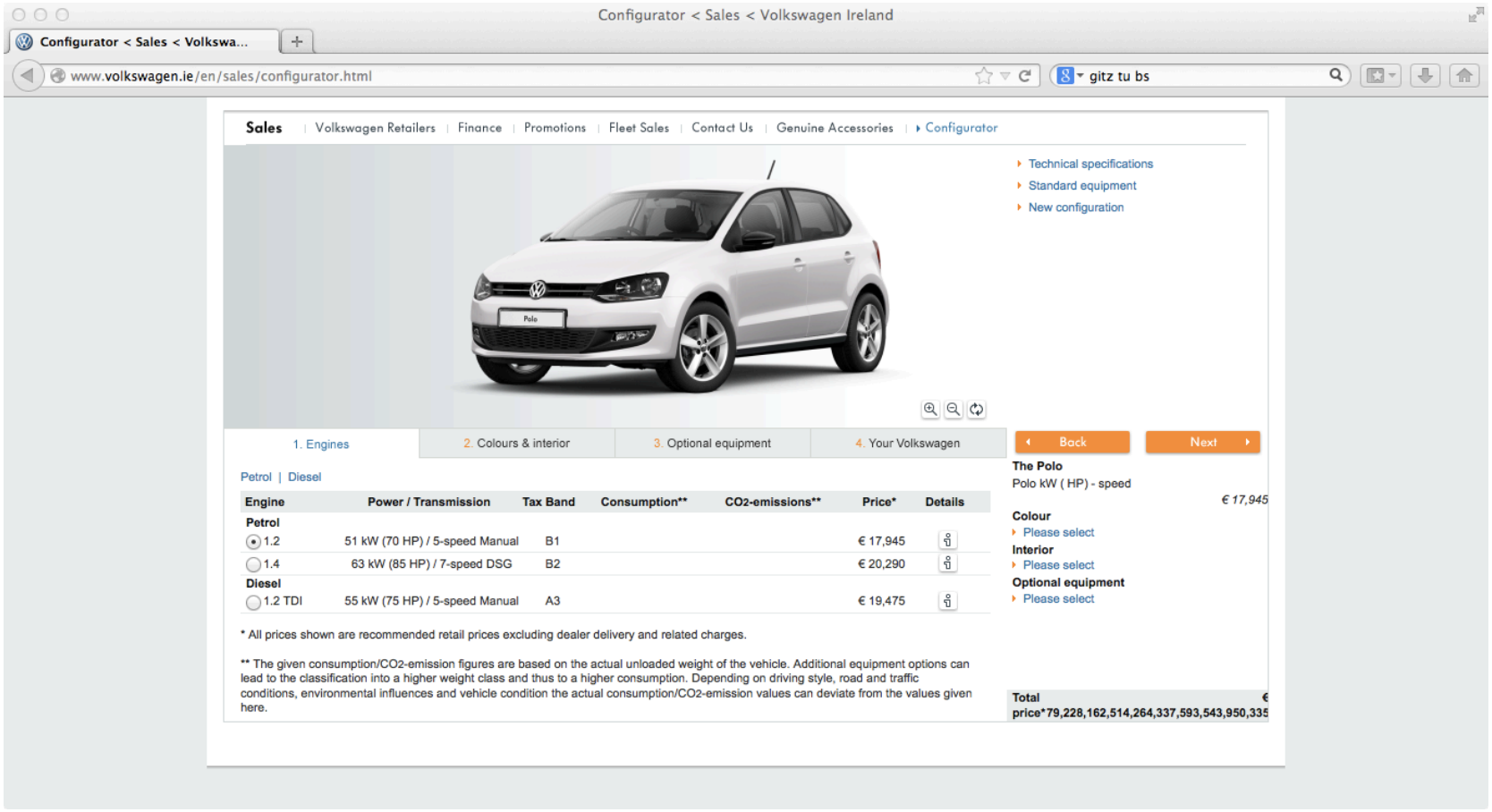
What is wrong here?



And here ...?

Quelle: mercedes-benz.de, 19. April 2013

And here …?


Configurator < Sales < Volkswagen Ireland
Configurator < Sales < Volkswa...
www.volkswagen.ie/en/sales/configurator.html
gitz tu bs
Sales
Volkswagen Retailers
Finance
Promotions
Fleet Sales
Contact Us
Genuine Accessories
Configurator
Technical specifications
Standard equipment
New configuration
1. Engines
2. Colours & interior
3. Optional equipment
4. Your Volkswagen
Back
Next
Petrol | Diesel
Engine
Power / Transmission
Tax Band
Consumption**
CO2-emissions**
Price*
Details
Petrol
1.2
51 kW (70 HP) / 5-speed Manual
B1
€ 17,945
1.4
63 kW (85 HP) / 7-speed DSG
B2
€ 20,290
Diesel
1.2 TDI
55 kW (75 HP) / 5-speed Manual
A3
€ 19,475
* All prices shown are recommended retail prices excluding dealer delivery and related charges.
** The given consumption/CO2-emission figures are based on the actual unloaded weight of the vehicle. Additional equipment options can lead to the classification into a higher weight class and thus to a higher consumption. Depending on driving style, road and traffic conditions, environmental influences and vehicle condition the actual consumption/CO2-emission values can deviate from the values given here.
The Polo
Polo kW (HP) - speed
€ 17,945
Colour
Please select
Interior
Please select
Optional equipment
Please select
Total price*79,228,162,514,264,337,593,543,950,335

Problem

- Assume we have safety-critical software (e.g., automobile ECUs)
  - → Every product variant should be safe!
- How to test every product variant? That's **impossible**!
- We need efficient strategies to reduce testing effort!

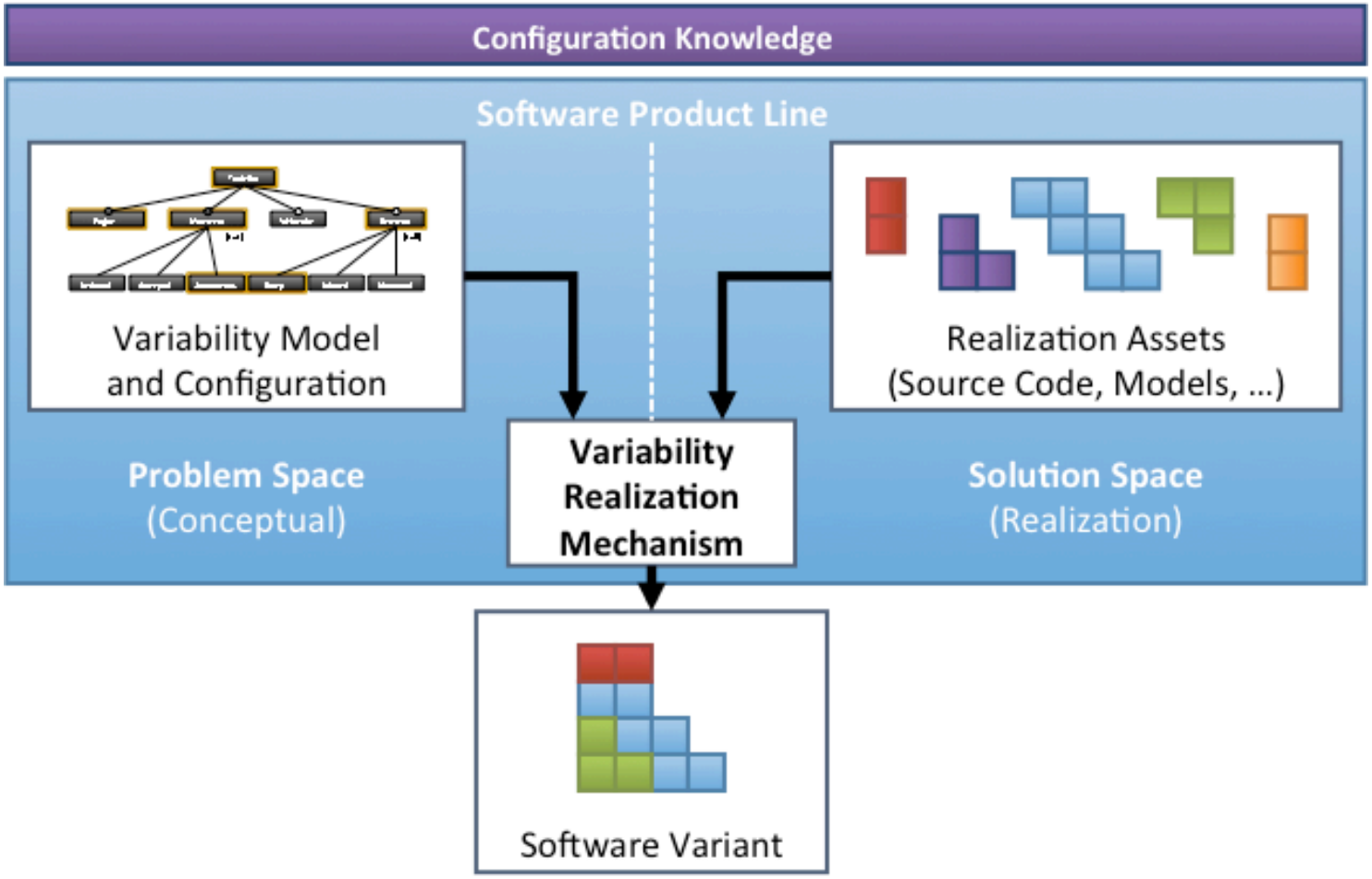
Agenda

1. Modeling Variant-rich Systems

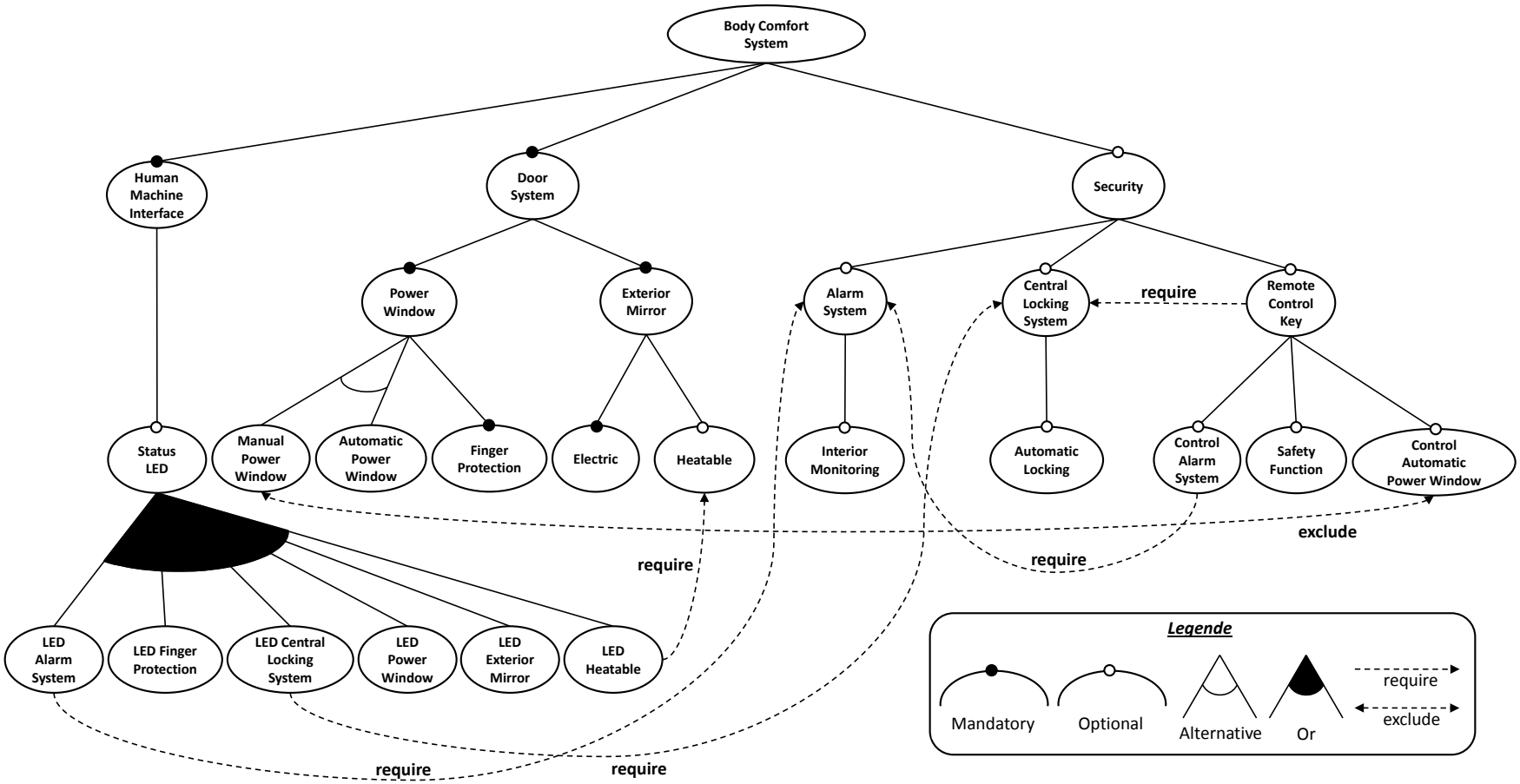
2. Efficient Analysis of Variant-Rich Systems
   a. Variant Selection Techniques
   b. Test effort reduction by incremental testing

Modeling Variant-rich Systems

Modeling (and Managing) Variant-rich Systems



Feature Models



Variability Realization Techniques

Annotative

(Subtractive/Negative)

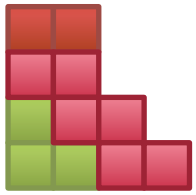
- Large „150% model“ for all variants
- Remove unneeded variable parts

E.g., pure::variants, Gears, FeatureMapper, conditional compilation

Compositional

(Additive/Positive)

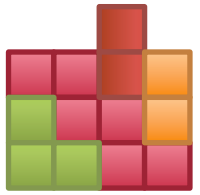
- Small common core
- Add variable parts



E.g., Feature-Oriented Programming (FOP), component systems

Transformational

- Valid basic variant of system as start
- Transformation by adding/removing/ modifying



E.g., Delta Modeling, CVL

Efficient Analysis of Variant-Rich Systems

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

Efficient Testing of Variant-Rich Software Systems

We use a combination of two approaches:

Combinatorial Testing:

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

Incremental Regression-based Testing:

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

Variant Selection Techniques

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 Covering Arrays

- S = {a,b,c,d,e} — SPL features
- M = {{a,b,c}, {b,d}, {c,d}, {d,e}} — Valid product configurations
- Minimal Covering Array L = M_1 + M_4 — Feature combinations to test

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

Heuristic Solution by Chvátal (1979)

- **Idea of the algorithm:**

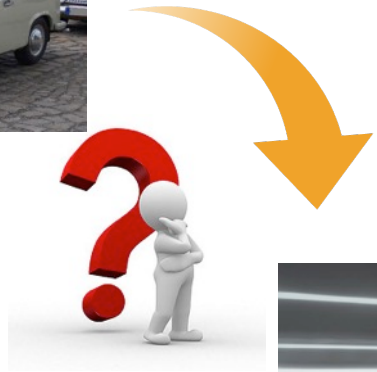
1. $Set\ L = \emptyset$
2. $If\ M_i = \emptyset, \forall\ i, i \in \{1, 2, \dots, n\}\ END.$
   $ELSE\ find\ M', where\ \#\ of\ uncovered\ elements\ is\ max$
3. $Add\ M'\ to\ L\ and\ replace\ elements\ in\ M_i\ by\ M_i - M'$
4. $Goto\ Step\ 2$

- **Worst Case:** M contains only subsets with different elements
- Best solution not guaranteed
- Adaptation for pairwise CA generation is easy!

Vasek Chvátal: *A Greedy Heuristic for the Set-Covering Problem*. Math. Oper. Res. 4(3): 233-235 (1979)

ICPL

```
input  : arbitrary FM
output: t-wise covering array
 1  S ← all t-tuples
 2  while S ≠ ∅ do
 3      k ← new and empty configuration
 4      counter ← 0
 5      foreach tuple p in S do
 6          if FM is satisfiable with k ∪ p then
 7              k ← k ∪ p
 8              S ← S\{p}
 9              counter ← counter + 1
10          end
11      end
12      if counter > 0 then
13          L ← L ∪ (FM satisfy with {k})
14      end
15      if counter < # of features in FM then
16          foreach tuple p in S do
17              if FM not satisfiable with p then
18                  S ← S\{p}
19              end
20          end
21      end
22  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…

Martin Fagereng Johansen, Øystein Haugen, Franck Fleurey: *An algorithm for generating t-wise covering arrays from large feature models*. SPLC (1) 2012: 46-55

ICPL - Evaluation

- ICPL can handle large-scale SPLs
- Compute 2-wise Covering Array with „normal“ hardware
- 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

Regression-based Delta-oriented Testing

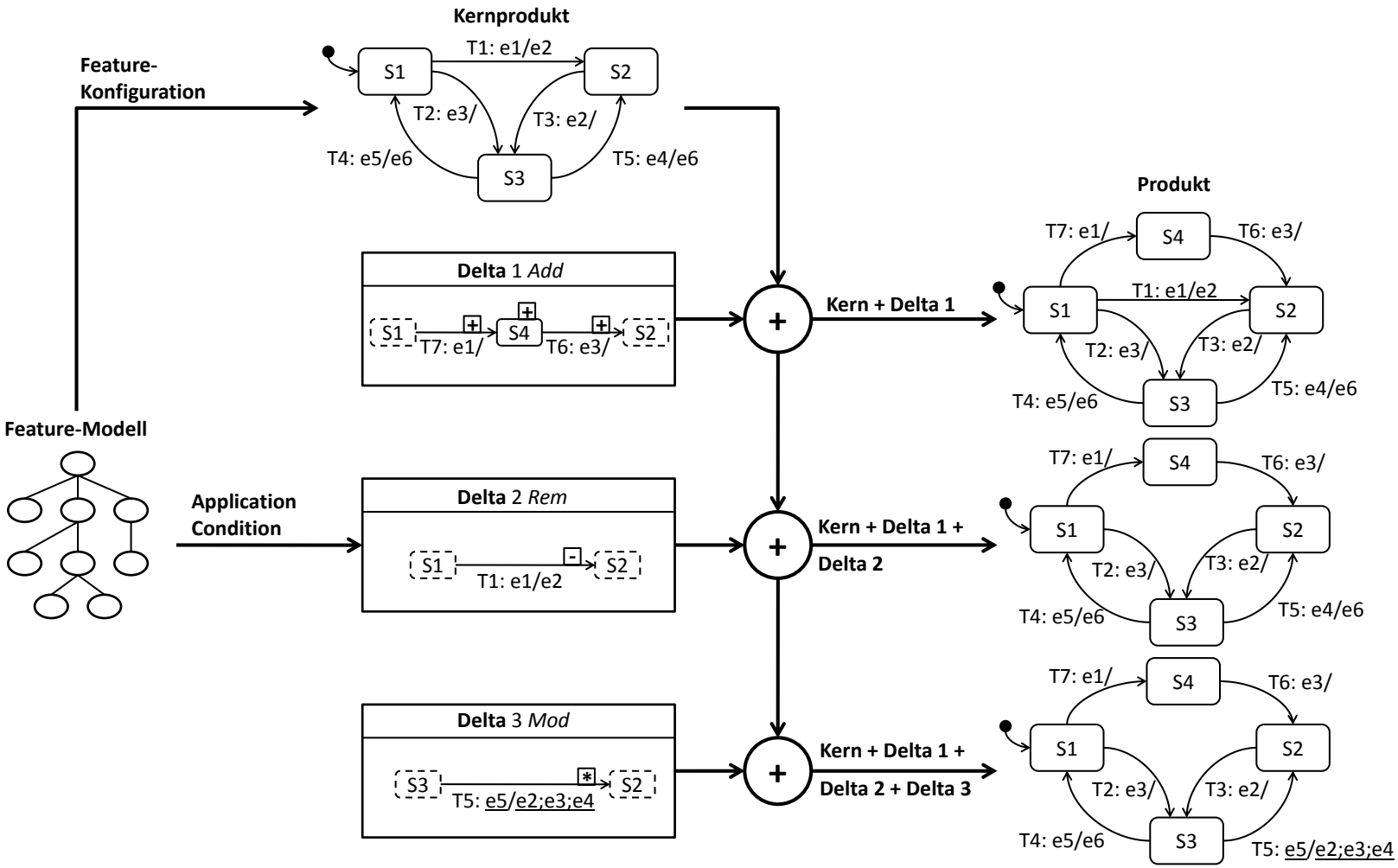
Delta-oriented Testing Approaches

- **Goal**: Reduce testing effort by only testing differences between product variants

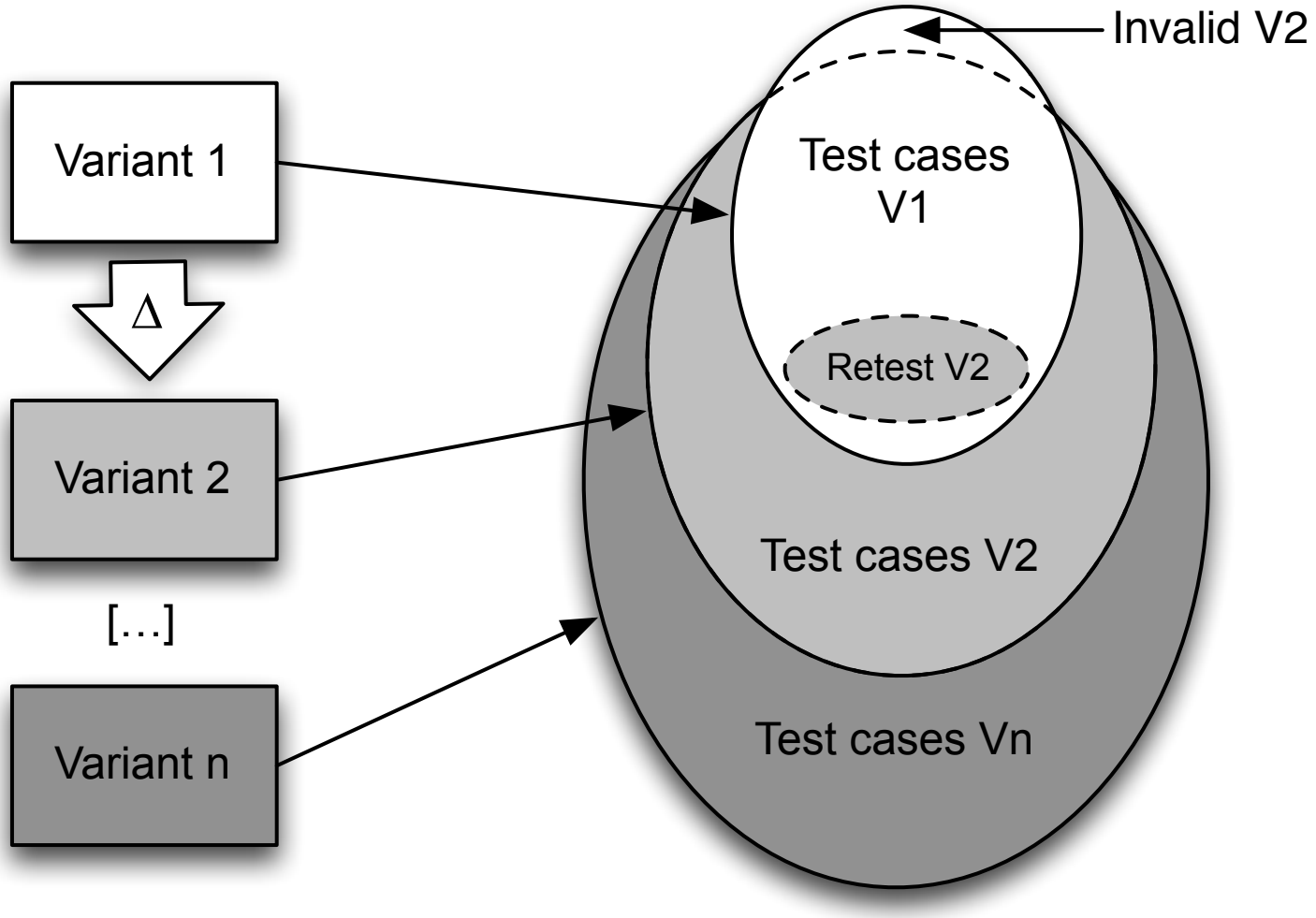
- **Define Deltas on test-models in model-based testing**
  - State Machines
  - Activity Diagrams
  - Architectures

- **Define Deltas on requirements** in natural language

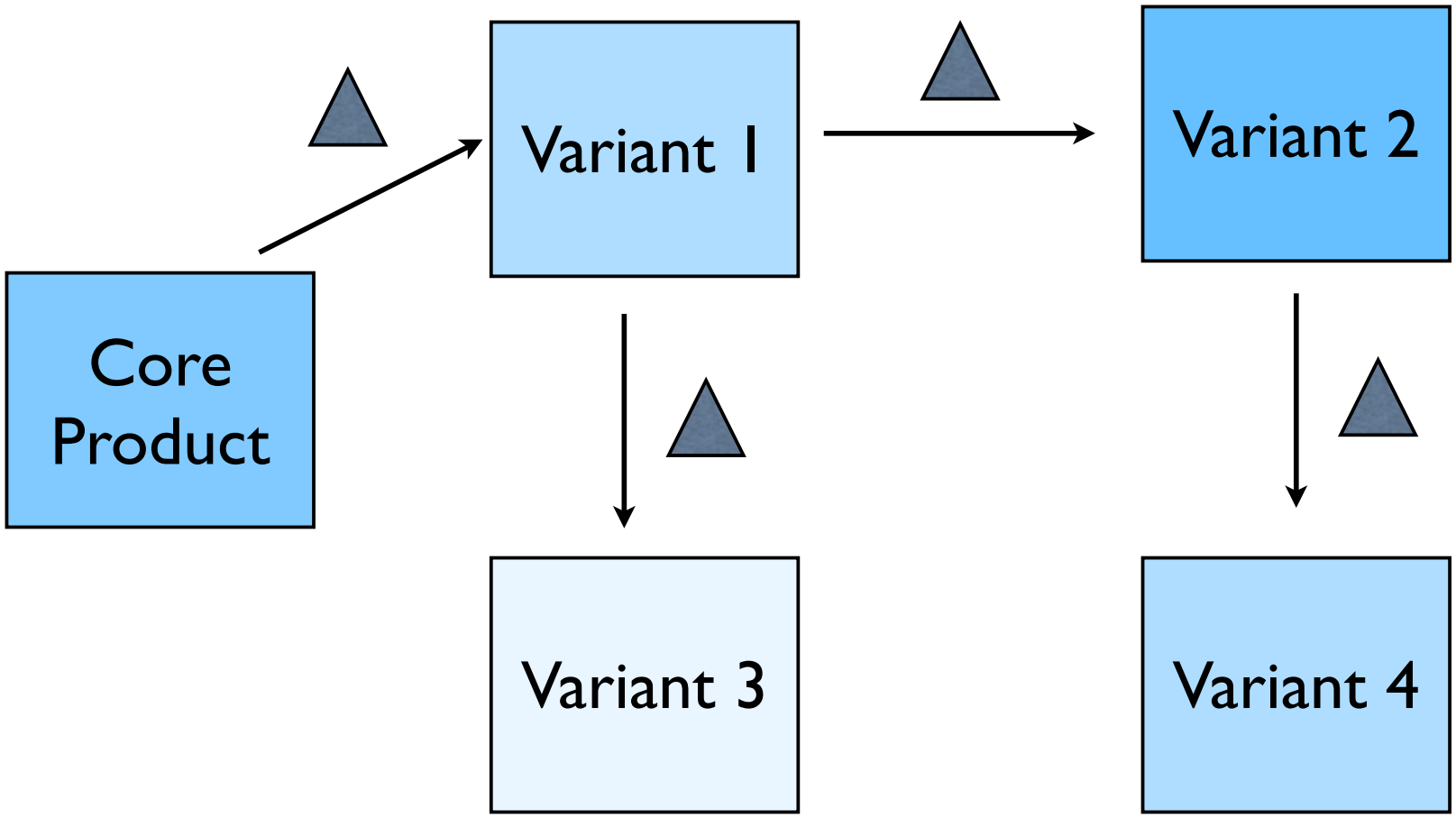
Delta-oriented Test Modeling



Classification of Test Cases by Delta-Analysis



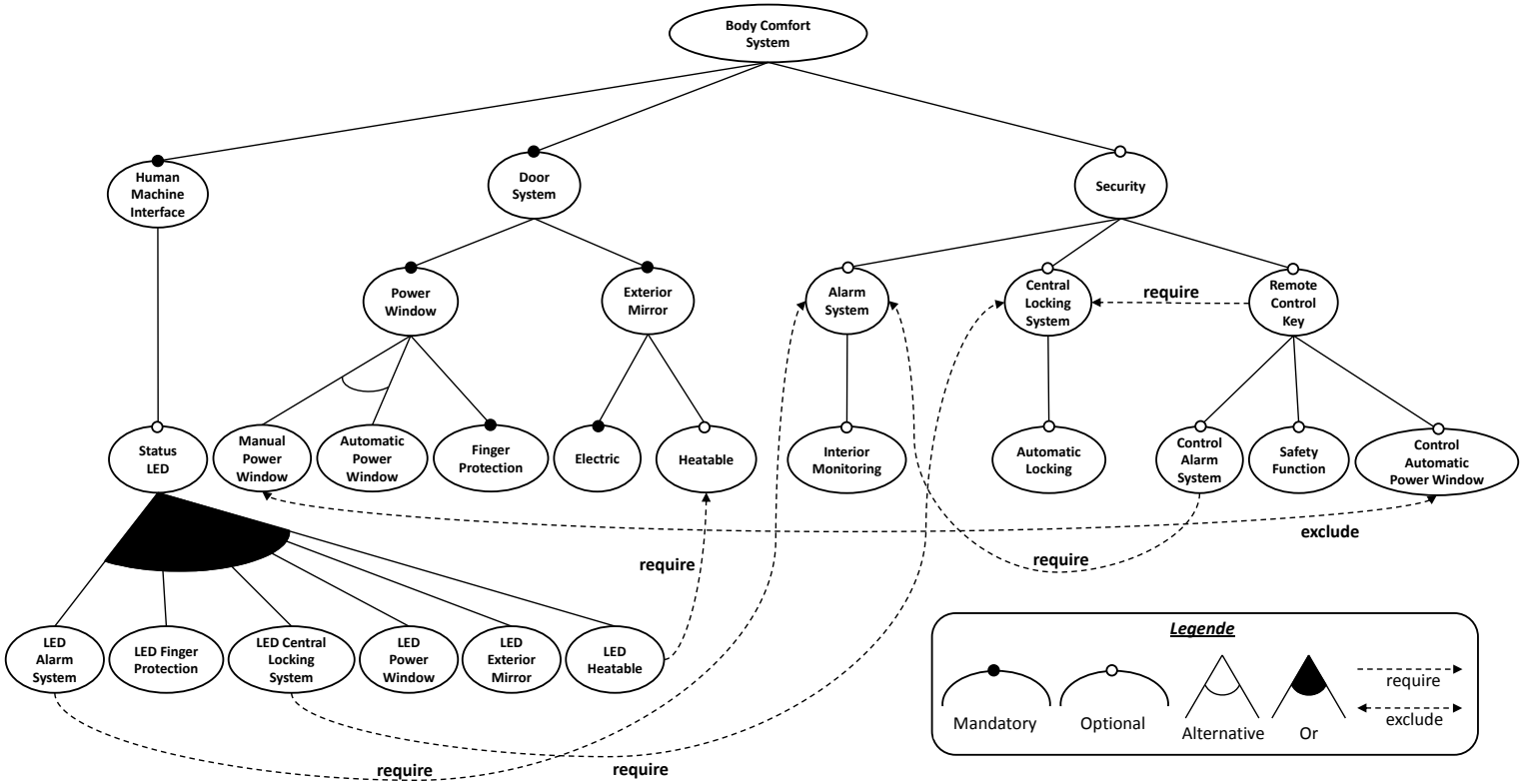
Delta Testing Strategy



Case Study – Body Comfort System

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

16 Products for Pair-Wise Feature Coverage

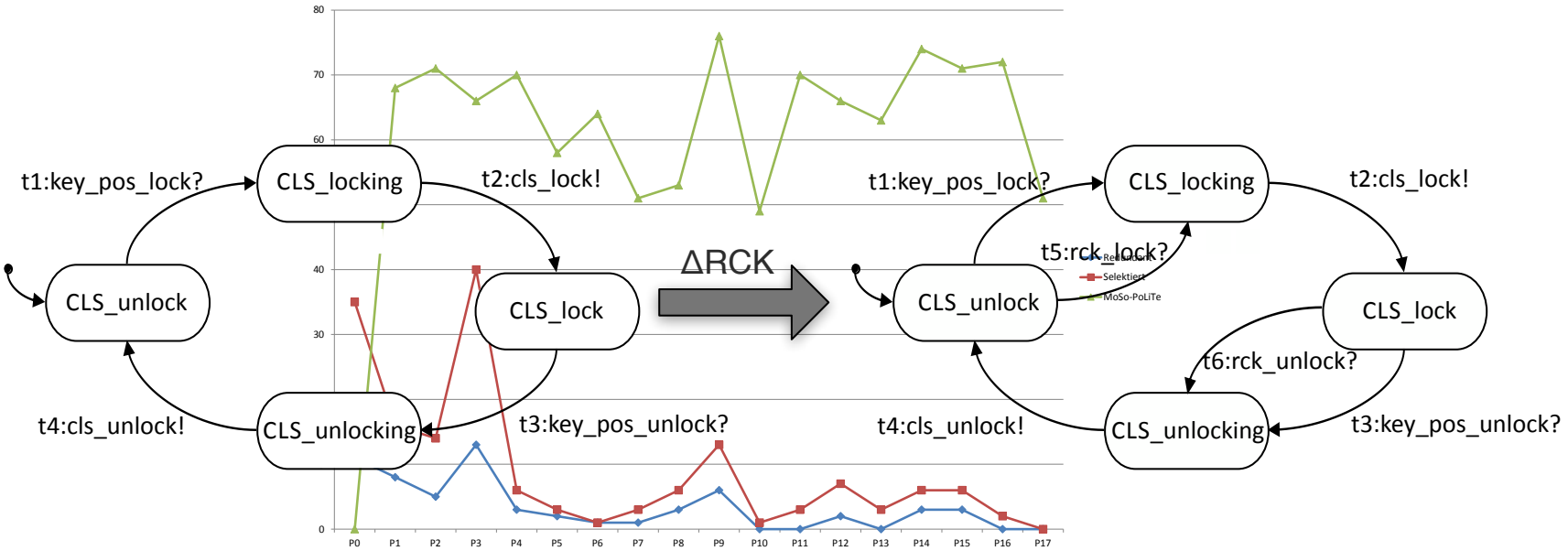


Sascha Lity, Remo Lachmann, Malte Lochau, Ina Schaefer: *Delta-oriented Software Product Line Test Models – The Body Comfort System Case Study*, Technische Universität Braunschweig, 2012

Deltas based on State Machines

- State machines describe component behavior
  - Used as test models for **component testing**

- Deltas describe *add*, *remove* and *modify* transformations of states, transitions and signals
- **Select test cases** based on coverage of changes

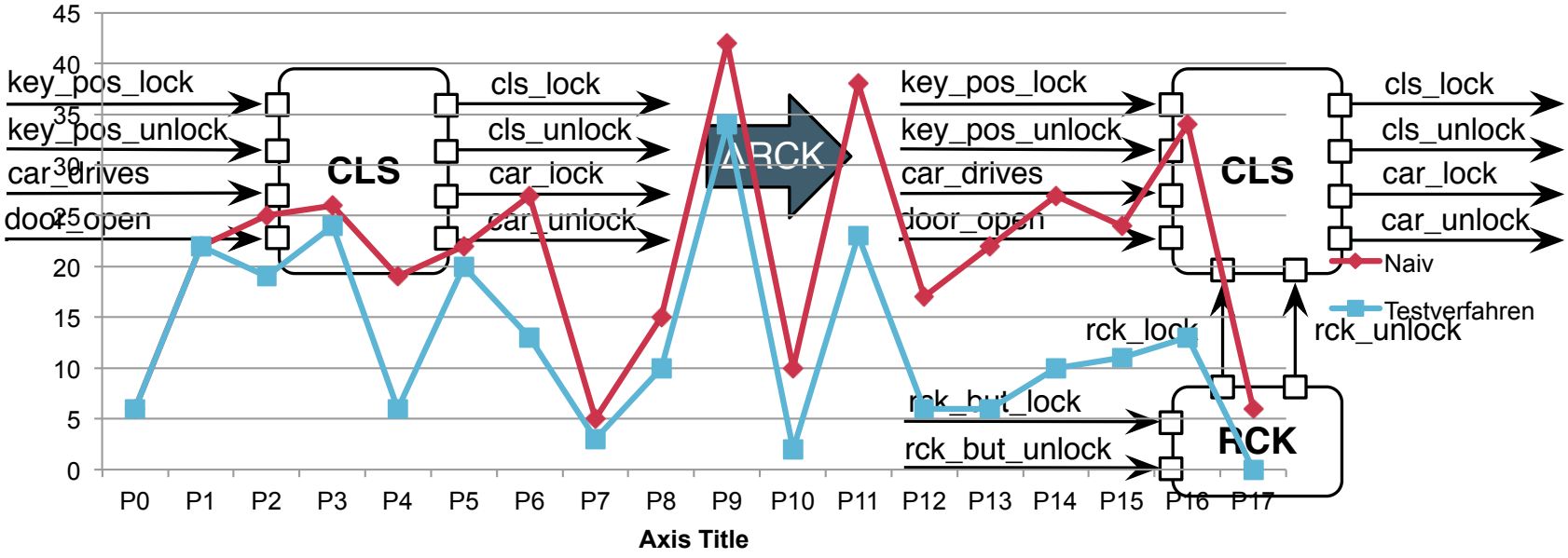


Malte Lochau, Sascha Lity, Remo Lachmann, Ina Schaefer, Ursula Goltz: *Delta-oriented model-based integration testing of large-scale systems*. Journal of Systems and Software 91: 63-84 (2014)

Deltas based on Software Architectures

- Architectures describe the structure of software and **component interaction**
  - Used as test models for **integration testing**

- Deltas describe *add* and *remove* transformations for components, connectors, ports and signals
- **Select test cases** based on coverage of changes

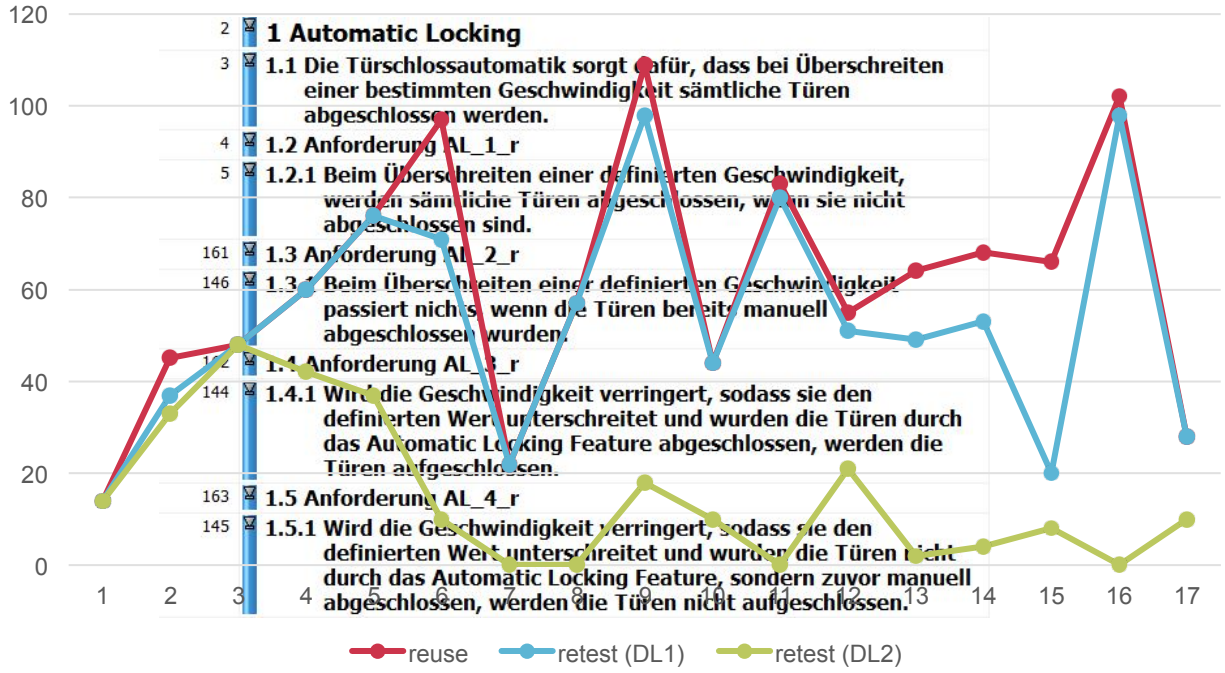


Malte Lochau, Sascha Lity, Remo Lachmann, Ina Schaefer, Ursula Goltz: *Delta-oriented model-based integration testing of large-scale systems*. Journal of Systems and Software 91: 63-84 (2014)

Deltas based on Requirements

- Requirements describe desired properties of software on system level
  - Used as specification for **system testing**

- Deltas describe *add* and *remove* transformations of requirements
- **Select test cases** based on requirements coverage



Michael Dukaczewski, Ina Schaefer, Remo Lachmann, Malte Lochau: *Requirements-based delta-oriented SPL testing*. PLEASE@ICSE 2013: 49-52

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

Drawback of Test Case Selection

Remember the 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

There is one drawback → Selection of test cases might still be huge!

Solution? Prioritize test cases to always test important parts first!

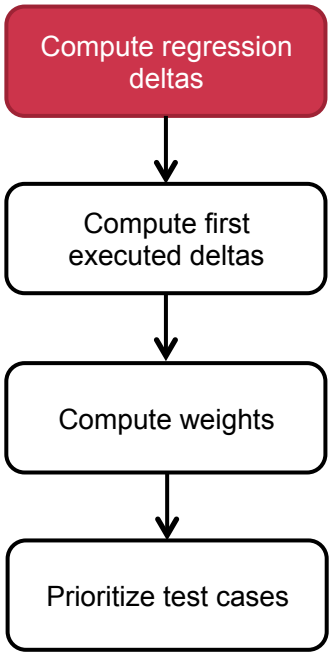
Delta-Oriented Test Case Prioritization

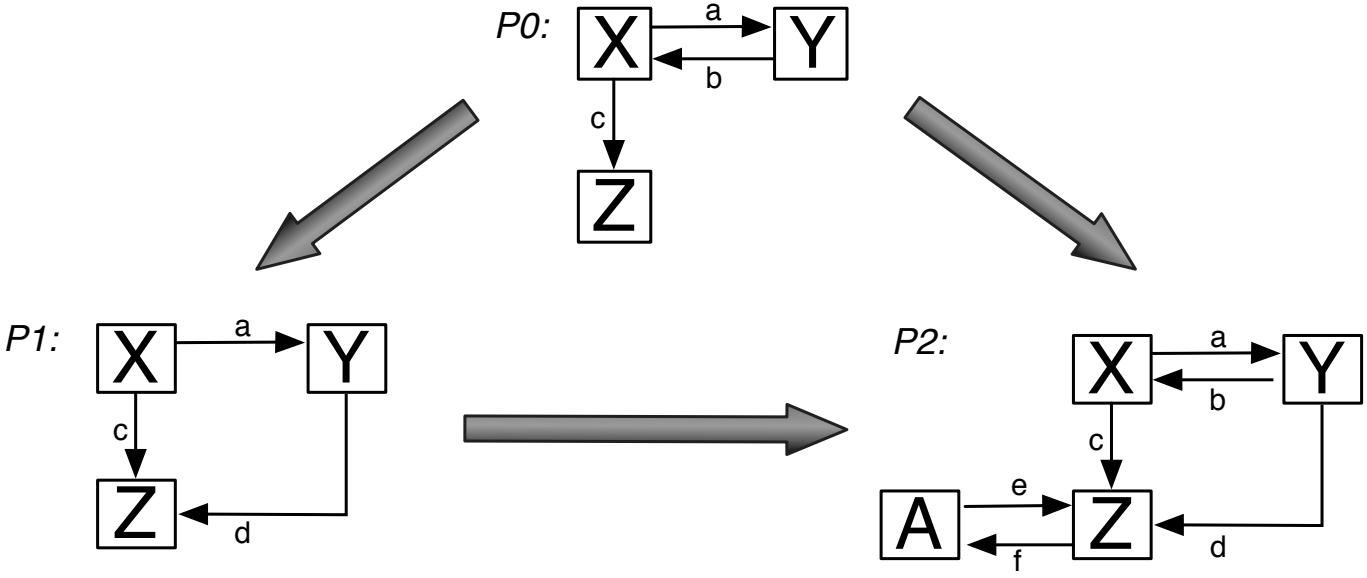
- We propose a test case prioritization approach for SPLs to find failures early

Remo Lachmann, Sascha Lity, Sabrina Lischke, Simon Beddig, Sandro Schulze, Ina Schaefer: *Delta-oriented test case prioritization for integration testing of software product lines*. SPLC 2015: 81-90

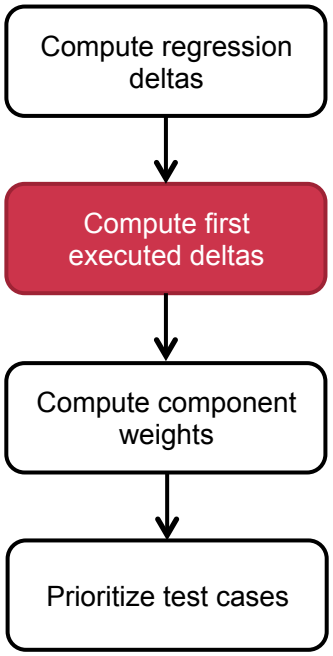
Test Prioritization Technique

Applicable to any type of test model with set of test model elements $\Omega = \{\omega_1, \ldots, \omega_n\}$ (edges and nodes)





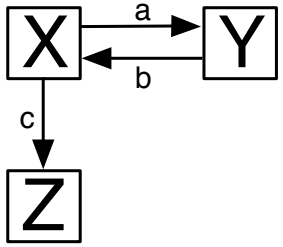
Test Prioritization Technique



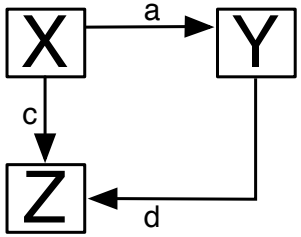
$\delta_{P_1} = \{rem\ b_{Y \to X}, add\ d_{Y \to Z}\}$

$\delta_{P_2} = \{add\ A, add\ d_{Y \to Z}, add\ f_{Z \to A}, add\ e_{A \to Z}\}$

P0:

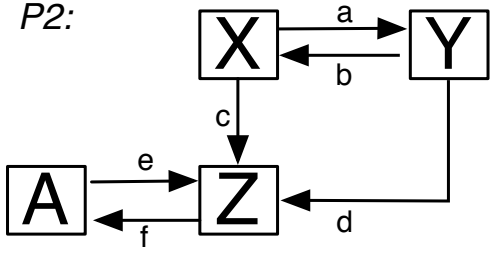


P1:



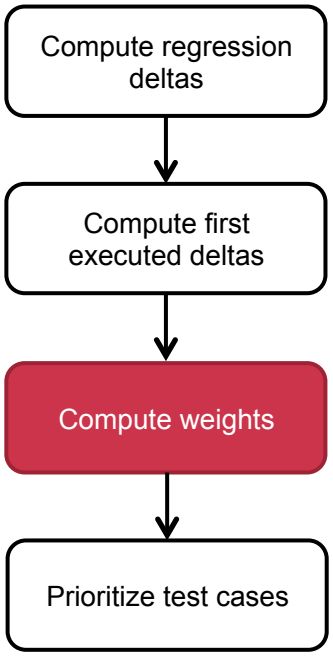
$\delta_{P_1,P_2} = \{add\ A, add\ b_{Y \to X}, add\ e_{Z \to A}, add\ f_{A \to Z}\}$

P2:



- $\delta_{new}(p_1) = \{rem\ b_{Y \to X}, add\ d_{Y \to Z}\}$
- $\delta_{new}(p_2) = \{add\ A, add\ e_{A \to Z}, add\ f_{Z \to A}\}$

Test Prioritization Technique



Define Test Model Weights

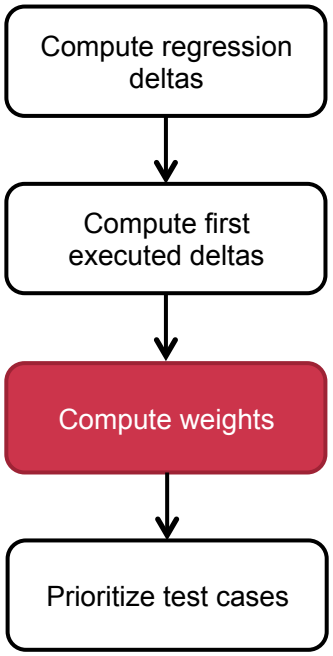
- Weights are defined for test model elements $\omega \in \Omega$
- Weight function of signature: $w : \Omega \rightarrow \mathbb{R}$
- No restriction in their semantics
- Can be combined

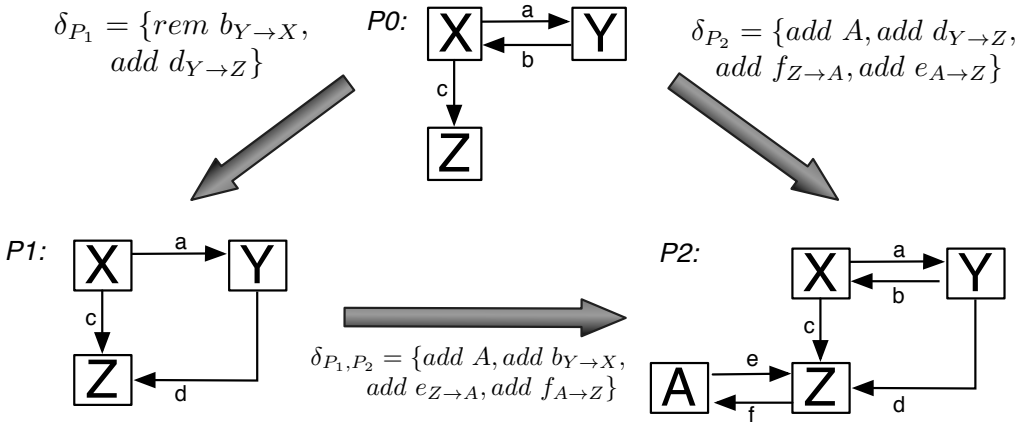
- Examples: Complexity of model elements, direct and indirect delta influences, interface changes, e.g., :

$$w(\omega) = \alpha \cdot \frac{|IC(\omega)|}{|I(\omega)|} + \beta \cdot \frac{|OC(\omega)|}{|O(\omega)|}$$

such that $\alpha, \beta \in \mathbb{R}, \alpha + \beta = 1$ holds.

Test Prioritization Technique





Example weight instantiation for components $c \in C$:

$$w(c) = \alpha \cdot \frac{|IC(c)|}{|I(c)|} + \beta \cdot \frac{|OC(c)|}{|O(c)|}$$

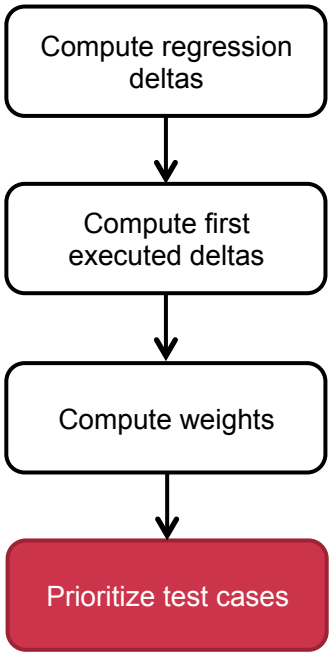
Weights for components X,Y and Z in P2 with $\alpha = 0.6$ and $\beta = 0.4$

$$w(X) = 0.6 \cdot \frac{0}{1} + 0.4 \cdot \frac{0}{2} = 0$$

$$w(Y) = 0.6 \cdot \frac{0}{1} + 0.4 \cdot \frac{0}{2} = 0$$

$$w(Z) = 0.6 \cdot \frac{1}{3} + 0.4 \cdot \frac{1}{1} = 0.6$$

Test Prioritization Technique



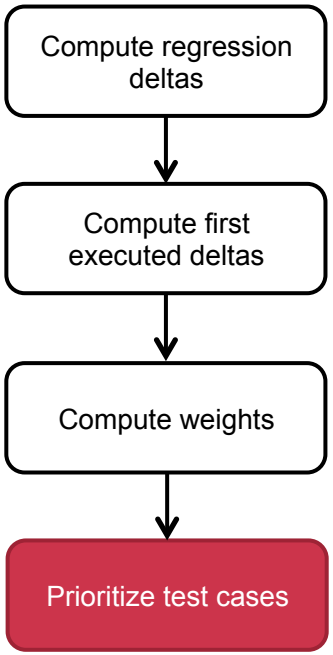
Define Prioritization Functions

- Definition based on weight functions
- Goal: Prioritize individual test cases $tc \in \mathcal{TC}$ for individual product variants $p \in P_{SPL}$
- Thus, signature is $prio : \mathcal{TC} \times P_{SPL} \rightarrow \mathbb{R}$

Similar to test case selection, we only prioritize *reusable* test cases for product variants

Process is repeated for all product variants under test

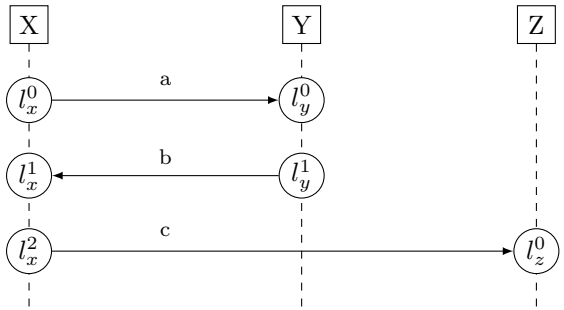
Test Prioritization Technique

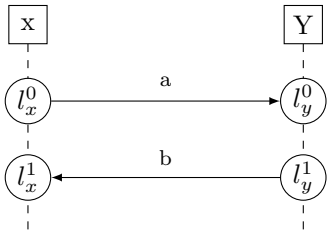


Example Prioritization Function for Architectures:

- Test cases defined as message sequence charts
- Prioritize test cases according to covered components
- Summarize component weights and normalize by total number of covered components n

$$prio_{comp}(tc, p) = \{\frac{\sum_{j=1}^{n} w(c_j)}{n} \mid c_j \in C_{tc}, tc \in \mathcal{TC}_{reuse_p}\}$$

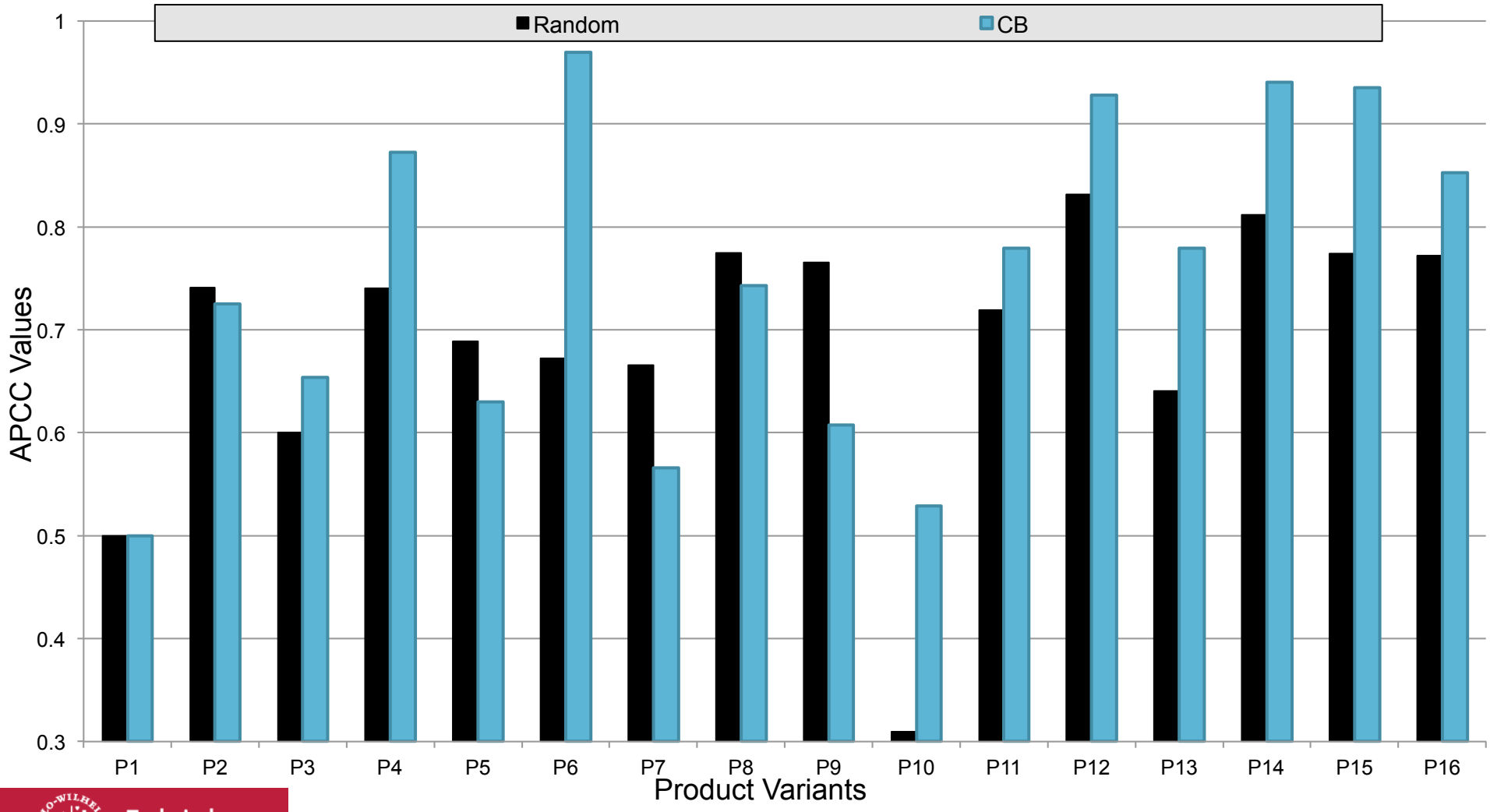




$$prio_{comp}(tc_1, p_2) = \frac{0 + 0 + 0.6}{3} = 0.2$$

$$prio_{comp}(tc_2, p_2) = \frac{0 + 0}{2} = 0$$

Evaluation Results



A Selection of Further Contributions in SPL Testing

- Dissimilarity-based testing in combination with delta-oriented testing

  Remo Lachmann, Sascha Lity, Mustafa Al-Hajjaji, Franz Fürchtegott, Ina Schaefer: *Fine-grained test case prioritization for integration testing of delta-oriented software product lines*. FOSD@SPLASH 2016: 1-10

- Optimize product variant orderings for incremental SPL testing

  Sascha Lity, Mustafa Al-Hajjaji, Thomas Thüm, Ina Schaefer: *Optimizing product orders using graph algorithms for improving incremental product-line analysis*. VaMoS 2017: 60-67

- Scalable risk-based testing for SPLs

  Remo Lachmann, Simon Beddig, Sascha Lity, Sandro Schulze, Ina Schaefer: *Risk-based integration testing of software product lines.* VaMoS 2017: 52-59

- Change impact analysis using incremental model slicing for SPL testing

  Sascha Lity, Thomas Morbach, Thomas Thüm, Ina Schaefer: *Applying Incremental Model Slicing to Product-Line Regression Testing*. ICSR 2016: 3-19

Conclusion

- **Variant-rich systems** lead to an exponential increasing number of products and test cases
- **Subset selection heuristics** reduce number of product variants to a representative subset which is tested.
- **Incremental delta-testing** reduces test effort between product variants by identifying effective test cases for reuse