innovationszentrum
applied system
modeling

Technische Universität Kaiserslautern

Fraunhofer

Analyzing similarity of multiple cloned software systems

Slawomir Duszynski

slawomir.duszynski@iese.fraunhofer.de

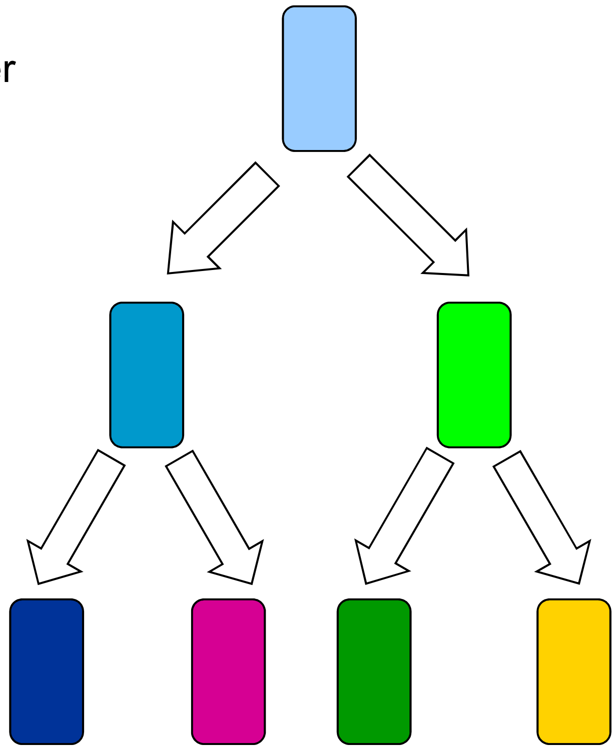
Fraunhofer IESE

Kaiserslautern, Germany

November 28, 2011
The 16th CREST Open Workshop
UCL London

Motivation for Multi-System Analysis

- The need for systematic software reuse is often recognized only after development of a group of similar software systems
  - Common practice: clone and adapt one of existing variants, no reuse mechanisms
  - “Software mitosis” (Faust 2003)
  - Variants are maintained independently from each other
  - Further variants emerge in the same way



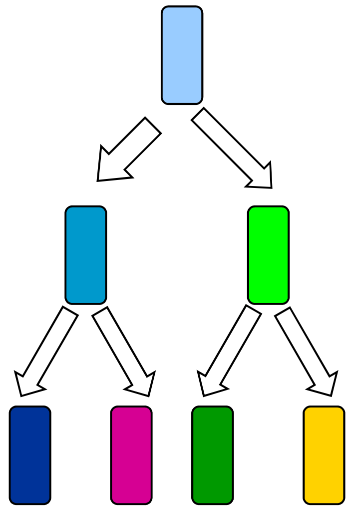
- Examples from the industry
  - 4 cloned variants, ca. 1.5 MLOC each
  - 14 cloned variants, ca. 200 KLOC each

- With a growing number of variants, maintenance becomes difficult
  - Redundant maintenance and QA effort

[D. Faust, C. Verhoef: Software Product Line Migration and Deployment. 2003]
[D. Beuche: Transforming Legacy Systems into Software Product Lines. SPLC 2010]

Motivation for Multi-System Analysis

- Having many similar variants, the company has two options:
  - **1: Develop** a new PL from scratch – costly, loss of past investment
  - **2: Migrate** the existing products – difficult, and costly too



- Typical migration problems
  - Variability in the existing code is not known
  - Code-level variability might differ from feature-level variability (Yoshimura 2006a)
  - High risk of incorrect reuse decisions (Garlan 1995; Kolb 2006)

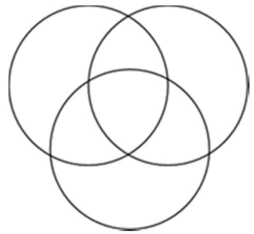
- Research problem: detailed information about the code variability is needed
  - variability needs to be recovered and understood
  - difficult for large systems and many variants

* [K. Yoshimura, D. Ganesan, D. Muthig: Assessing Merge Potential of Existing Engine Control Systems into a Product Line. SEAS 2006]

"*the portion of functional commonality among two products is about 60-75%; their implementations, however, share as little as around 30% of code"*

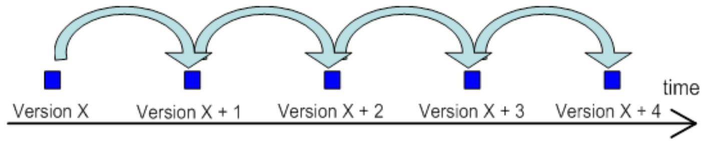
We need an analysis technique that:

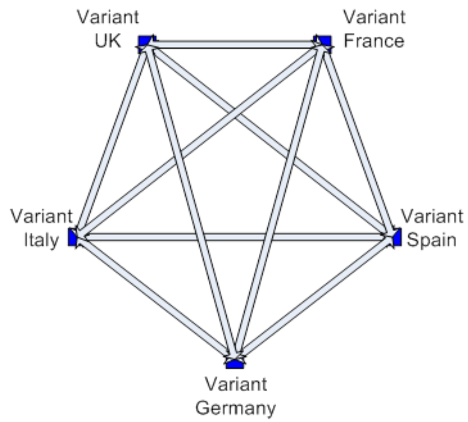
- Provides both abstract and detailed information
  - Available for any part of the code
  - Available for any variant or variant intersection



- Is scalable
  - High number of LOC
  - High number of variants
  - Suitable abstraction needed (providing just a flat list of similarities is not scalable!)

- Is specifically targeted at variants, not versions
  - Versions form a time-ordered list
    - It is enough to analyze n-1 pairs
  - Variants exist in parallel and cannot be ordered
    - Analysis of $\frac{n(n-1)}{2}$ pairs needed
    - Result cannot depend on any variant ordering





- [IESE context] Is understandable to practitioners

Existing Approaches

- Similarity metrics calculated on the whole systems (Yamamoto2005)
  - Only high-level information: it is known that there are differences, but it is not known where they are

- Clone detection and manual result analysis (Yoshimura2006b)
  - No scalability (lots of manual work, for just 2 variants)

- Clone detection and further result processing (Mende2008)
  - Unsuitable result presentation

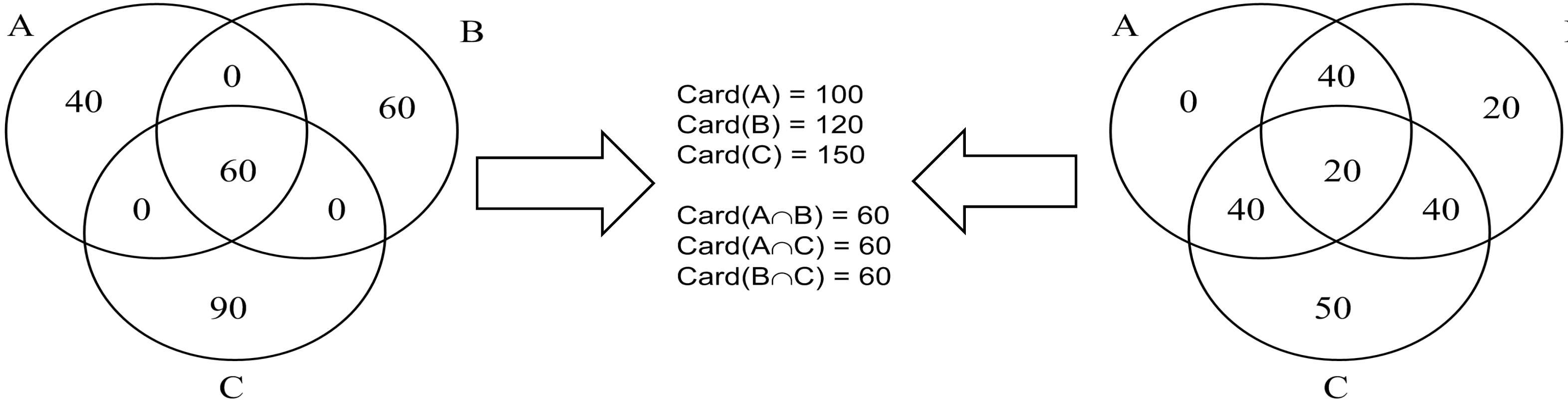
[T. Yamamoto, M. Matsushita, T. Kamiya, K. Inoue: Measuring similarity of large software systems based on source code correspondence. 2005]

[K. Yoshimura, D. Ganesan, and D. Muthig: Defining a strategy to introduce a software product line using existing embedded systems. EMSOFT 2006]

[T. Mende, R. Koschke: Supporting the Grow-and-Prune Model in Software Product Lines Evolution Using Clone Detection. 2008]

Existing Approaches

Information on Any Variant Intersection: Not Available



	FW	PF^1	FW^1	P_1^1	P_2^1	PF^2	P^3	P^4
FW		5 (0%)	1 (0%)	3 (0%)	1 (0%)	11 (1%)	4 (0%)	8 (0%)
PF^1	5 (0%)		11 (0%)	175 (8%)	173 (8%)	28 (1%)	69 (3%)	97 (4%)
▷ FW^1	1 (0%)	11 (3%)		–	11 (3%)	15 (4%)	17 (5%)	29 (8%)
▷ P_1^1	3 (0%)	162 (32%)	–		162 (32%)	11 (2%)	24 (4%)	30 (5%)
▷ P_2^1	1 (0%)	186 (16%)	11 (0%)	175 (15%)		2 (0%)	28 (2%)	38 (3%)
PF^2	11 (1%)	41 (3%)	16 (1%)	23 (2%)	5 (0%)		20 (1%)	18 (1%)
P^3	4 (0%)	57 (9%)	17 (2%)	21 (3%)	26 (4%)	15 (2%)		175 (30%)
P^4	9 (0%)	89 (4%)	27 (1%)	28 (1%)	39 (2%)	13 (0%)	185 (9%)	

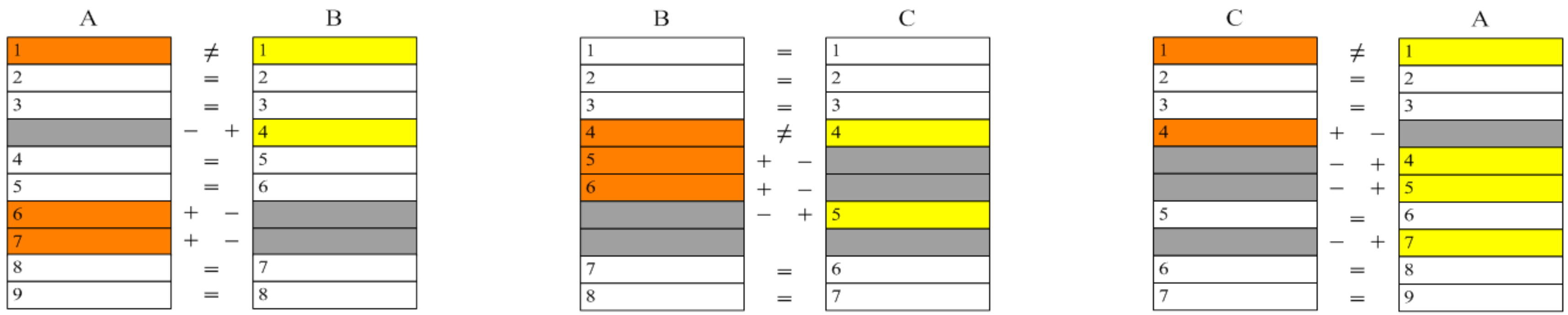
Result presentation in (Mende2008)

- Pair-wise result presentation
  - **Problem**: incomplete information
  - Example 1: Two different situations (above) cannot be distinguished as they provide the same pair-wise result
  - Example 2: impossible to answer questions such as “where is the core of my potential product line?”
  - **Problem:** complex result
    - $O(n^2)$ variant pairs!

Technische Universität Kaiserslautern Fraunhofer

Variant Analysis
Example Situation

- Consider three source code files A, B and C
  - The task: recognize and characterize the commonalities and variabilities
  - A human could use the diff tool to understand the differences



- Practical problems in a product line context:
  - Scalability problem: for n systems there are *n(n-1)/2* pairs. Hard to understand for a human (e.g. n=6 –> 15 different pairs to be related to each other)
  - Comparison delivers pair-wise results such as “same” and “different”: but for the product line, we want to know which lines are **core** and which are **unique**

Variant Analysis
Occurrence Matrices

- For each variant, list its elements in a matrix
- Add union matrix to represent the total analyzed code
- Fill the matrix
  - Rows: variant elements
  - Columns: all the existing variants; additionally: number of variants where the element occurs
  - Cells: occurrence of the elements in the variants (1: occurrence, 0: no occurrence)

A

	A	B	C	Sum
1	1	0	0	1
2	1	1	1	3
3	1	1	1	3
4	1	1	0	2
5	1	1	0	2
6	1	0	1	2
7	1	0	0	1
8	1	1	1	3
9	1	1	1	3

B

	A	B	C	Sum
1	0	1	1	2
2	1	1	1	3
3	1	1	1	3
4	0	1	0	1
5	1	1	0	2
6	1	1	0	2
7	1	1	1	3
8	1	1	1	3

C

	A	B	C	Sum
1	0	1	1	2
2	1	1	1	3
3	1	1	1	3
4	0	0	1	1
5	1	0	1	2
6	1	1	1	3
7	1	1	1	3

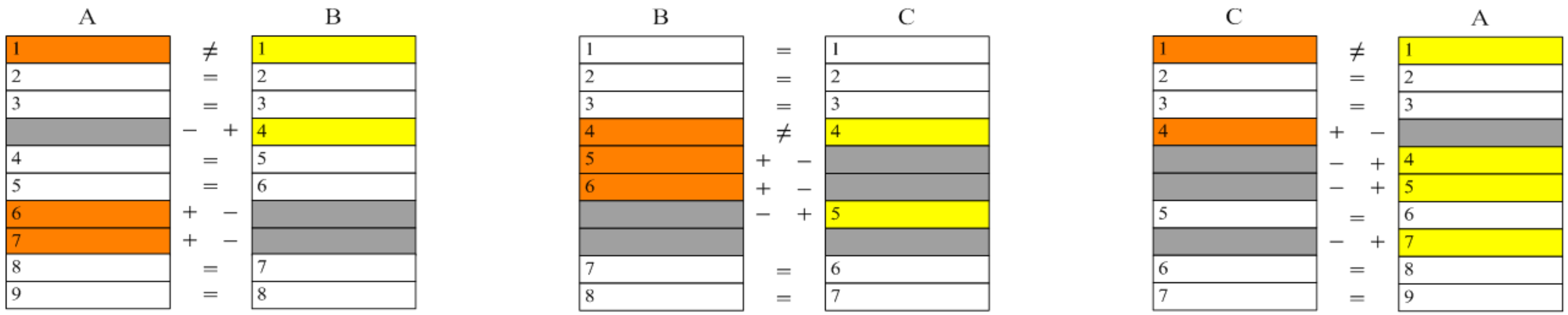
All

	A	B	C	Sum
A1	1	0	0	1
A2	1	1	1	3
A3	1	1	1	3
A4	1	1	0	2
A5	1	1	0	2
A6	1	0	1	2
A7	1	0	0	1
A8	1	1	1	3
A9	1	1	1	3
B1	0	1	1	2
B4	0	1	0	1
C4	0	0	1	1

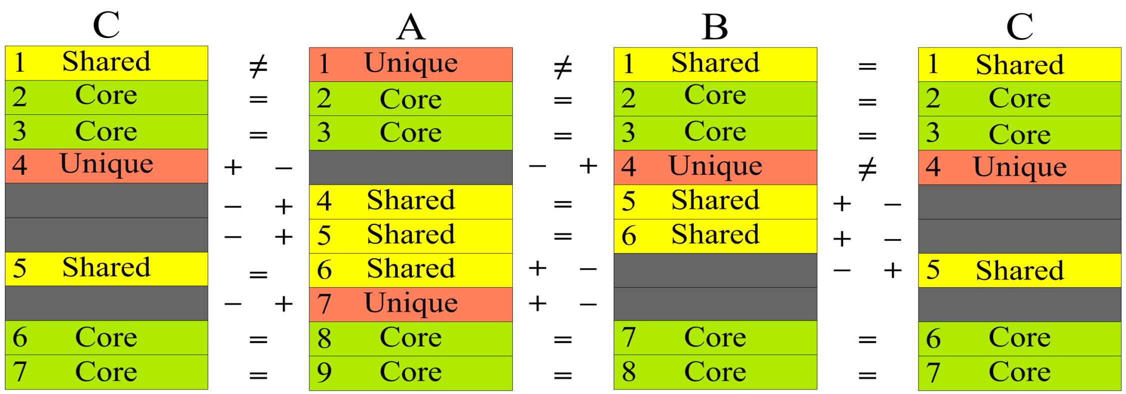
- Redefine the line status to make it appropriate for product lines
  - Not “same” and “different”, but “**core**” (Sum=n), “**shared**”, “**unique**” (Sum=1)

Variant Analysis
n-ary Diff Results

- Instead of a group of diff-ed pairs…



- … the result is a n-ary diff performed on all the involved variants:

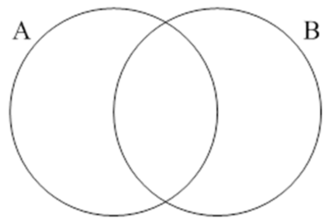


- Using the same principle, a comparison for any number of variants is possible

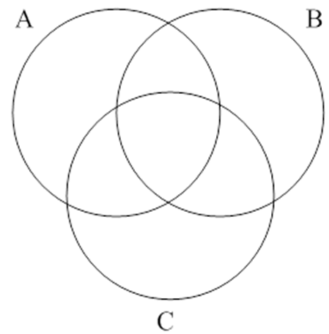
Variant Analysis – Visualization

Venn Diagrams: Not the way to go…

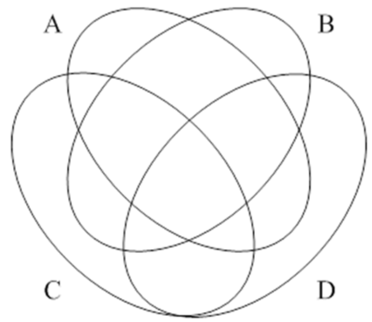
- Venn diagrams: very useful for small number of sets



n = 2

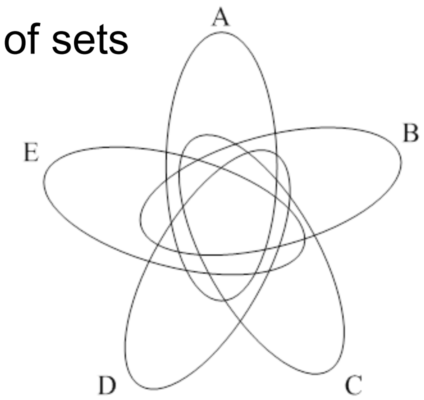


n = 3

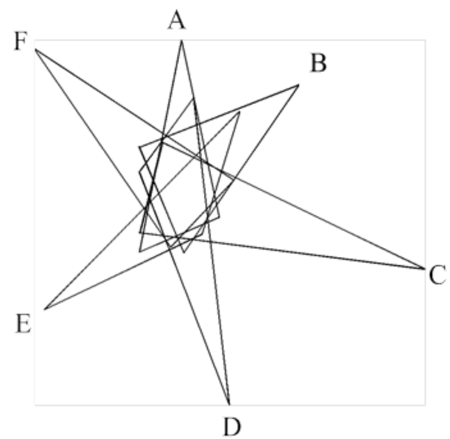


n = 4

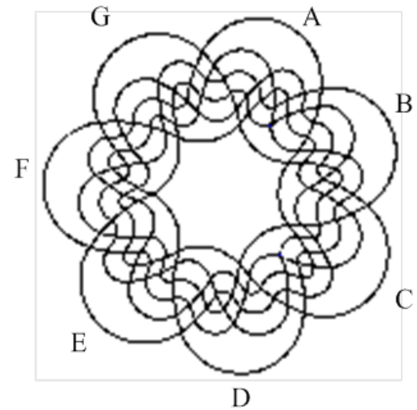
- Harder to understand for larger number of sets



n = 5



n = 6



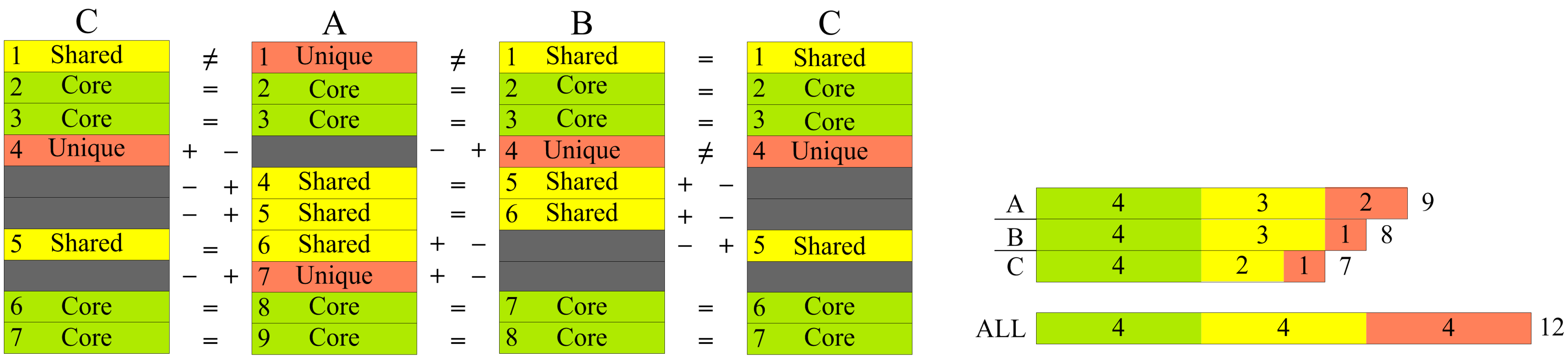
n = 7

Number of diagram areas = 2^n

Variant Analysis

Visualization: Bar Diagrams

- Bar diagrams are a way to visualize occurrence matrices
  - One bar created for each occurrence matrix (in total: n+1 bars)

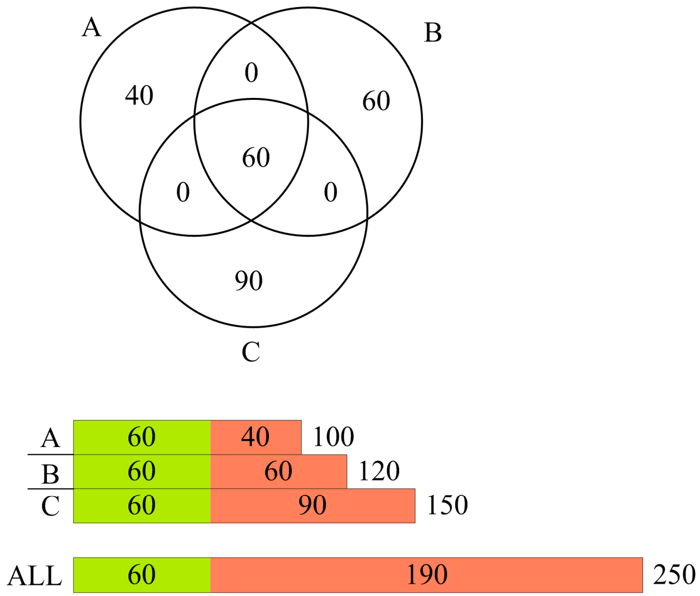


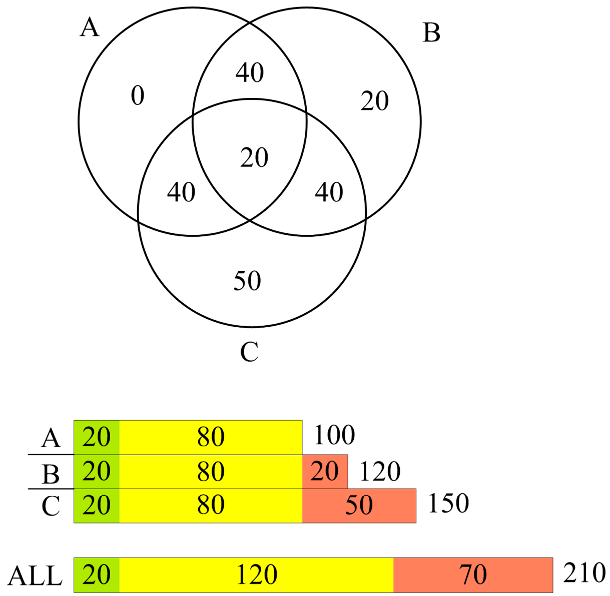
- Size of the bar = number of elements in the matrix
- Bar parts symbolize the core, shared and unique elements in the variants
- Sizes of the particular parts reflected in the diagram

Technische Universität Kaiserslautern Fraunhofer

Variant Analysis

Information on Any Variant Intersection Available

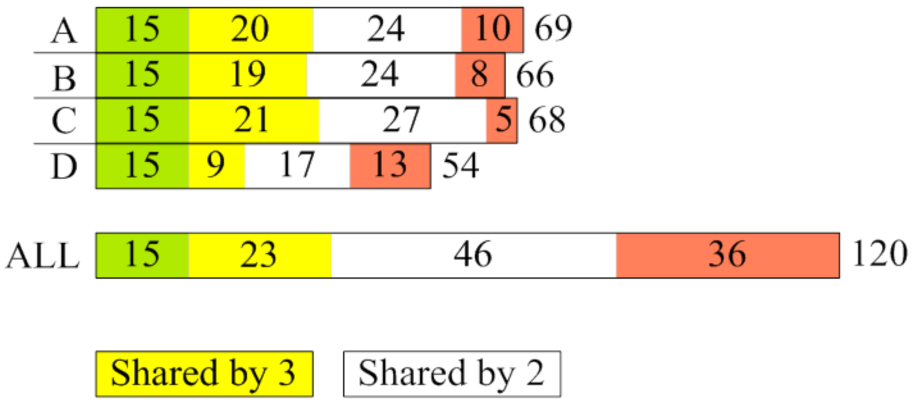


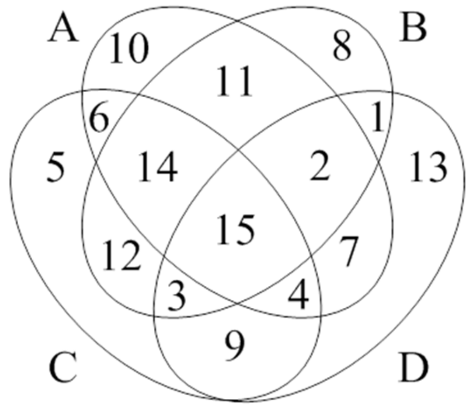


- The information provided by Variant Analysis is complete
  - Two example situations easily distinguishable
  - Any set intersection can be obtained using subset calculations
  - It is know <u>how much</u> elements fulfill a criterion and <u>which</u> elements they are
- Information can be easily presented even for a high number of variants

Variant Analysis
Subset Calculations

- Sometimes a specific subset of the analyzed system group is interesting, e.g.:
  - All elements shared by at least k systems
  - Elements common for a given system and other systems
  - Subsets such as A ∩ ¬B ∩ ¬C ∩ D



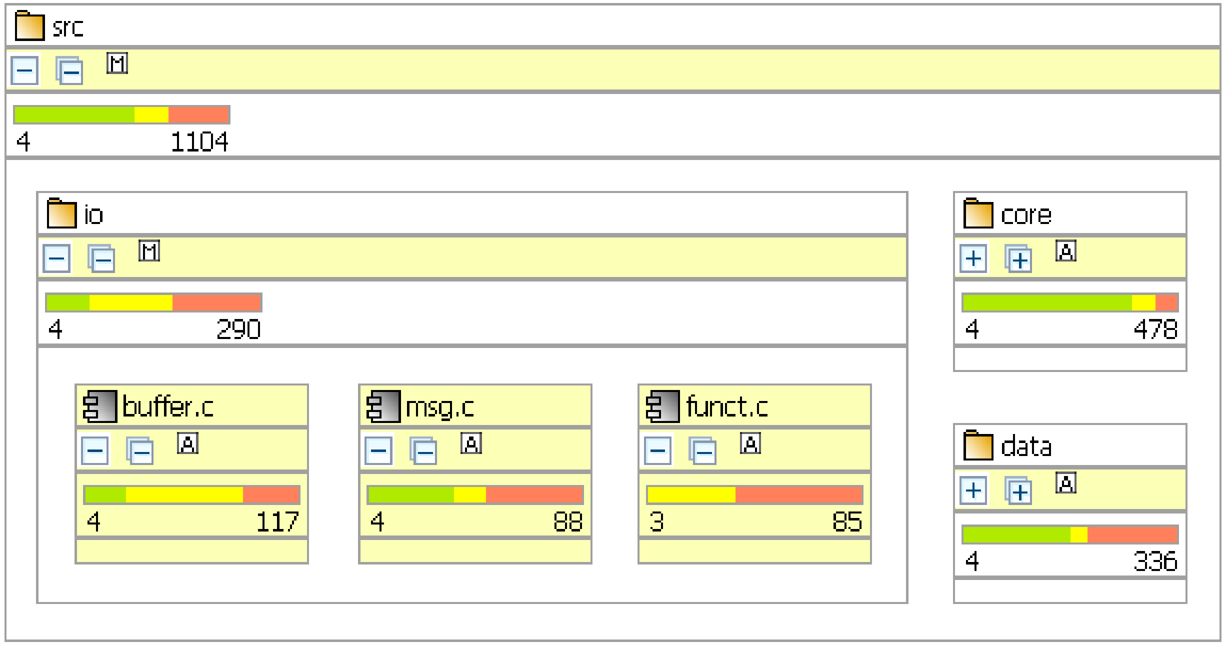


- Subset elements can be found by evaluating the element occurrences in the matrix
- Visualization on a bar diagram: display relevant bar parts and associated numbers
- Visualization in text editor: highlight relevant text lines in the text editor

Variant Analysis

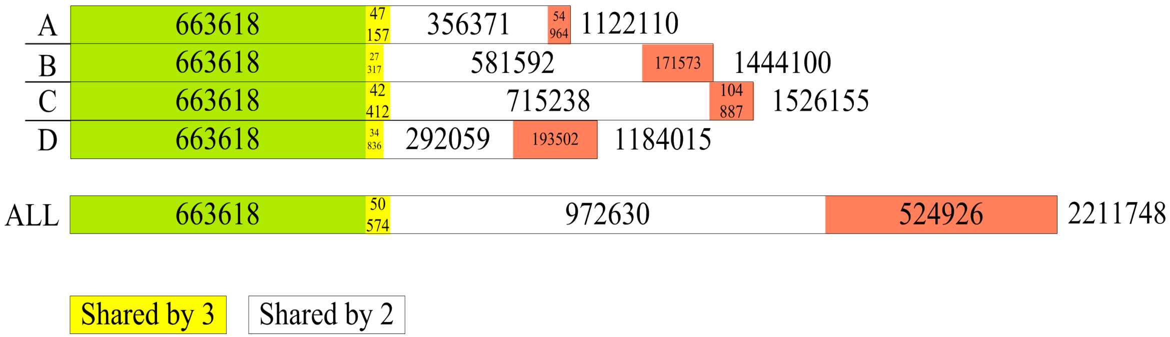
Scalable Result Abstraction and Navigation

- Variant Analysis integrated into Fraunhofer SAVE tool (Eclipse plug-in)
- Top-down result exploration possible using structural architectural views
  - Detect interesting areas on the high level structure
  - Go to details only where relevant results exist
  - Example: the folders “core” vs. “data” in the figure

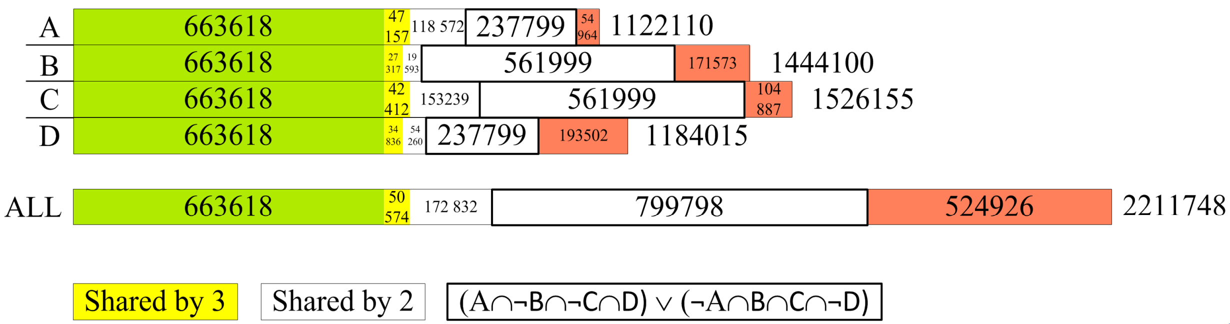


Variant Analysis
Industrial Application

- Good scalability and performance
  - Four 1.5 MLOC variants (implemented in C++) analyzed in **7 minutes**

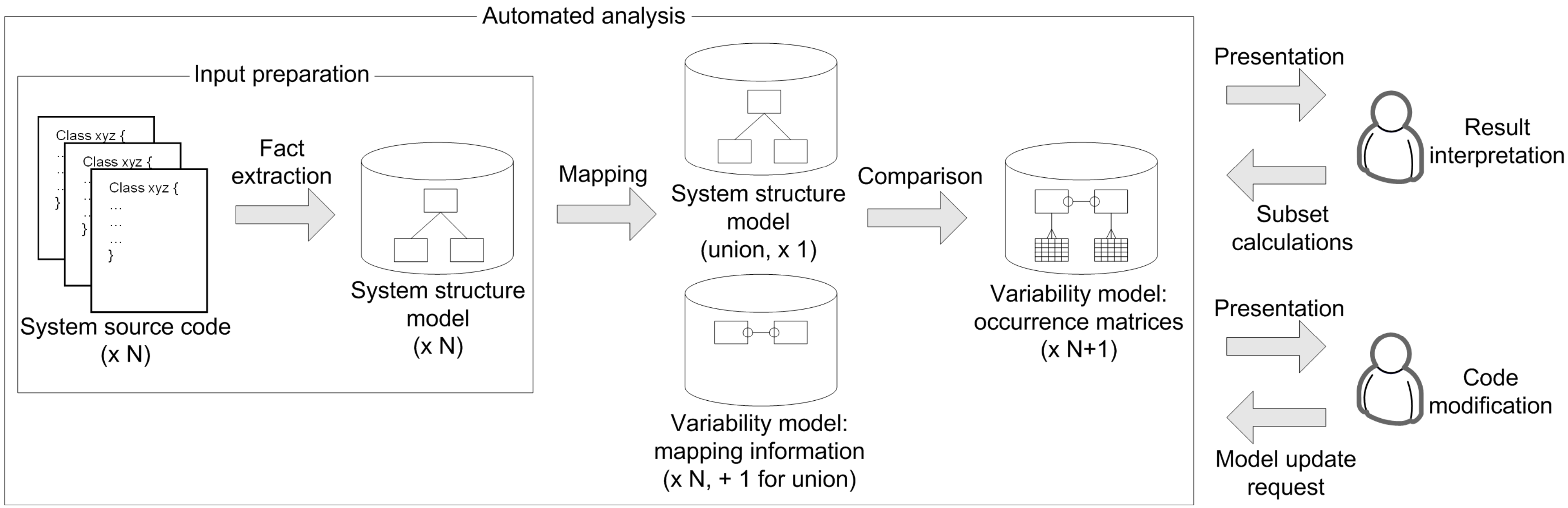


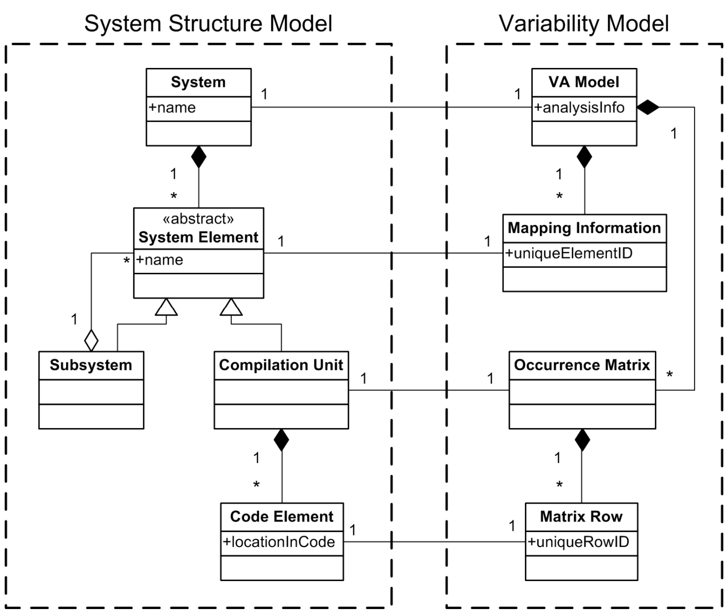
  - Subset calculations on all rows time range **from 312ms to 328ms**



Technische Universität Kaiserslautern Fraunhofer

Diff is just an example data source!





- The Variant Analysis model is <u>generic</u>
  - Different system representations possible
- Analysis phases can be adapted to specific needs
  - Different similarity detection algorithms possible

Generalization

Equivalence Relation and Unambiguous Assignment

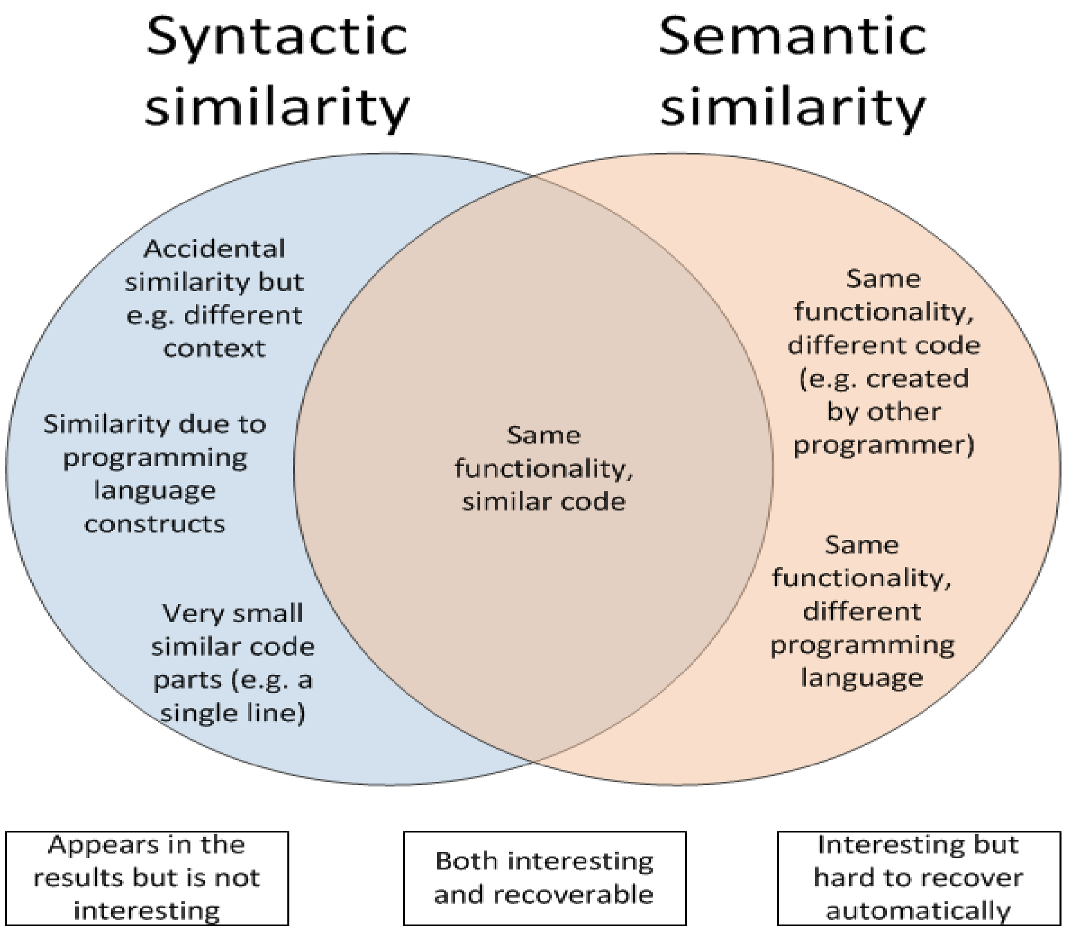
- Bar diagrams and occurrence matrices can be applied to analyze and visualize any kind of variability
  - Code, non-code artifacts, model elements, features, …
- The prerequisite for using the technique is a "correct" filling of the occurrence matrix
  - **Equivalence relation** across the variants' elements needed
    - Reflexive $\forall x \in S$: $\mathbf{x}\ \mathit{rel}\ \mathbf{x}$ == true
    - Symmetric $\forall x,y \in S$: $\mathbf{x}\ \mathit{rel}\ \mathbf{y} \Rightarrow \mathbf{y}\ \mathit{rel}\ \mathbf{x}$
    - Transitive $\forall x,y,z \in S$: $\mathbf{x}\ \mathit{rel}\ \mathbf{y} \wedge \mathbf{y}\ \mathit{rel}\ \mathbf{z} \Rightarrow \mathbf{x}\ \mathit{rel}\ \mathbf{z}$
  - **Unambiguous assignment** of equivalent elements across variants
    - Necessary if more than one element from variant A is equivalent to a given element of variant B

[S. Duszynski: Visualizing and Analyzing Software Variability with Bar Diagrams and Occurrence Matrices. SPLC 2010]

[S. Duszynski, J. Knodel, M. Becker: Analyzing the Source Code of Multiple Software Variants for Reuse Potential. WCRE2011]

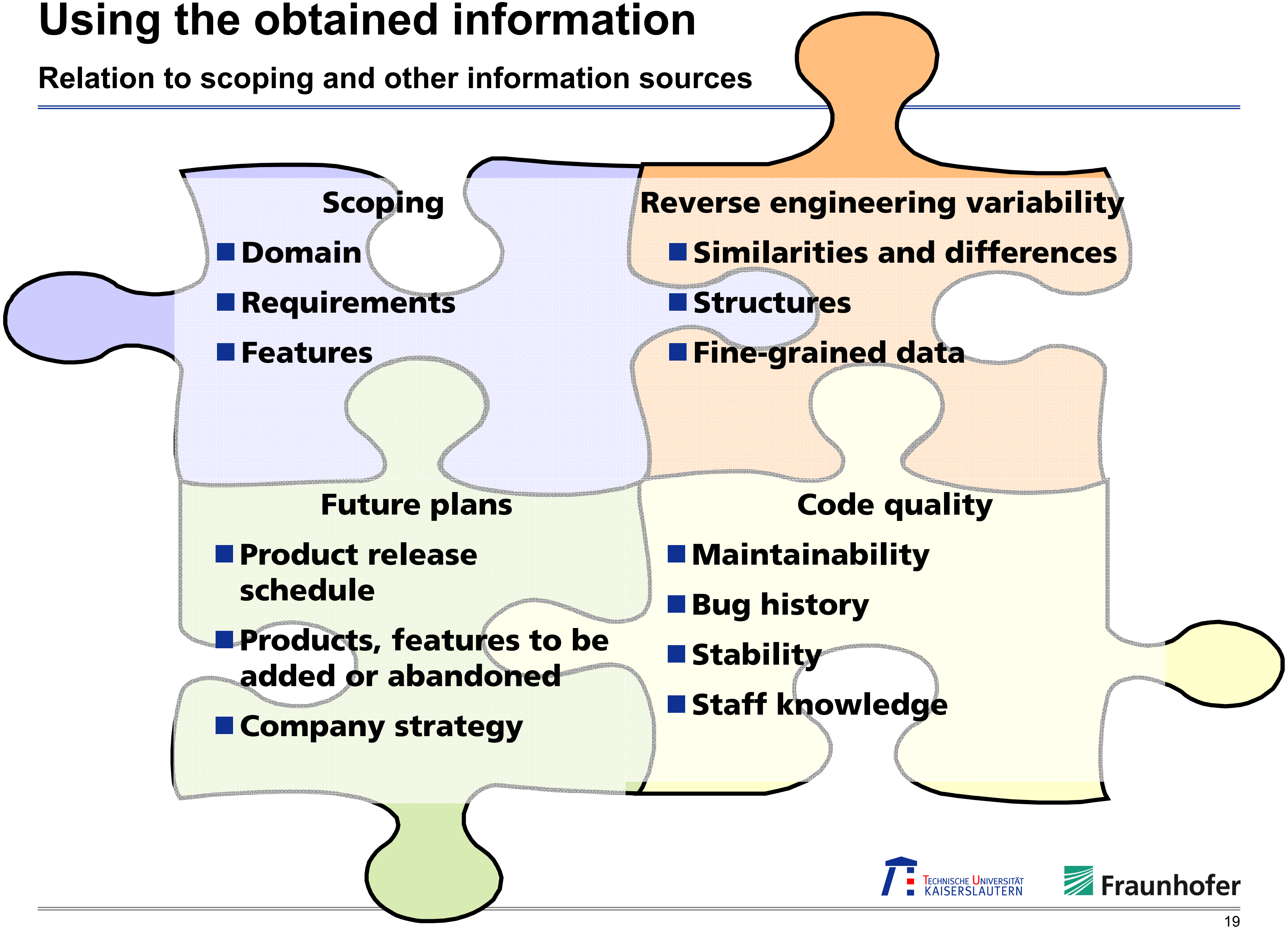
Limitations

- Typical situation in reverse engineering:
  - Use syntax-level approaches…
  - … trying to derive meaningful (semantic-level) results
- Variant Analysis retrieves just the syntactic similarity
- It also depends on the structure similarity: comparing non-cloned system does not deliver interesting results



Using the obtained information

Relation to scoping and other information sources



Summary

- Occurrence matrices: a data structure to store detailed variability information
- Matrix construction algorithm
  - Scalable: works for any number of variants
  - Generic: supports any element types
  - Flexible: equivalence relations enable customized definitions of similarity

A

	A	B	C	Sum
1	1	0	0	1
2	1	1	1	3
3	1	1	1	3
4	1	1	0	2
5	1	1	0	2
6	1	0	1	2
7	1	0	0	1
8	1	1	1	3
9	1	1	1	3

B

	A	B	C	Sum
1	0	1	1	2
2	1	1	1	3
3	1	1	1	3
4	0	1	0	1
5	1	1	0	2
6	1	1	0	2
7	1	1	1	3
8	1	1	1	3

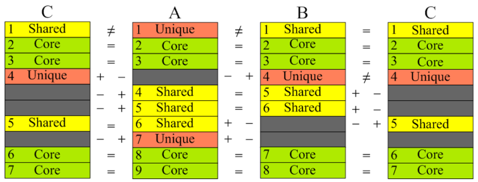
C

	A	B	C	Sum
1	0	1	1	2
2	1	1	1	3
3	1	1	1	3
4	0	0	1	1
5	1	0	1	2
6	1	1	1	3
7	1	1	1	3

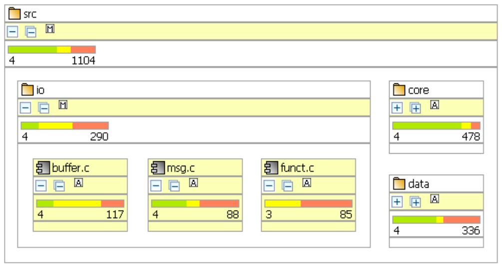
All

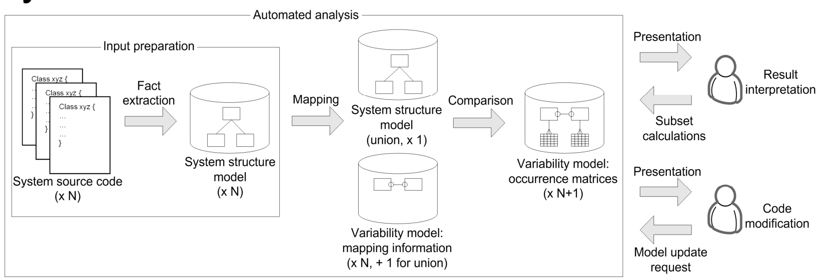
	A	B	C	Sum
A1	1	0	0	1
A2	1	1	1	3
A3	1	1	1	3
A4	1	1	0	2
A5	1	1	0	2
A6	1	0	1	2
A7	1	0	0	1
A8	1	1	1	3
A9	1	1	1	3
B1	0	1	1	2
B4	0	1	0	1
C4	0	0	1	1

- Bar diagrams: visualization technique for variability information
- Subset calculations: on-demand retrieval of variant intersections



- Generalized framework for analysis of cloned systems





Further work

- Attach a data source more advanced than diff
  - Clone detection results
  - Model-based comparison
- Define further analyses on the rich data set available
  - E.g. variability metrics: granularity, # different configurations needed, …
- Try to obtain more semantic-level results
  - Mapping features to code, traceability, …
- Perform (publishable) case studies

Thank you!

A

	A	B	C	Sum
1	1	0	0	1
2	1	1	1	3
3	1	1	1	3
4	1	1	0	2
5	1	1	0	2
6	1	0	1	2
7	1	0	0	1
8	1	1	1	3
9	1	1	1	3

B

	A	B	C	Sum
1	0	1	1	2
2	1	1	1	3
3	1	1	1	3
4	0	1	0	1
5	1	1	0	2
6	1	1	0	2
7	1	1	1	3
8	1	1	1	3

C

	A	B	C	Sum
1	0	1	1	2
2	1	1	1	3
3	1	1	1	3
4	0	0	1	1
5	1	0	1	2
6	1	1	1	3
7	1	1	1	3

All

	A	B	C	Sum
A1	1	0	0	1
A2	1	1	1	3
A3	1	1	1	3
A4	1	1	0	2
A5	1	1	0	2
A6	1	0	1	2
A7	1	0	0	1
A8	1	1	1	3
A9	1	1	1	3
B1	0	1	1	2
B4	0	1	0	1
C4	0	0	1	1