Reverse Engineering Product Lines

KV Product Line Engineering (343.354)

Dr. Roberto Lopez-Herrejon
Dr. Rick Rabiser

JOHANNES KEPLER UNIVERSITY LINZ

ISSE

15.06.2016

Lesson Overview

- Motivation for Reverse Engineering Software Product Lines
  - Basic ideas and examples

- Four main challenges
  - Reverse Engineering feature models
  - Traceability
  - Safe composition and feature oriented refactoring
  - Maintenance and Evolution

And Software Product Lines?

- Software Product Line
  - A set of similar software systems distinguished by the set of features they implement.
- Feature
  - Increment in program functionality

Fact: software products are marketed by features

Software Marketed by Features – Example

MagicDraw® 17.0.3 FR features

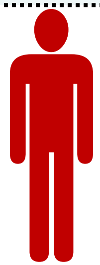
	Edition				
UML support	Personal	Standard	Professional	Architect	Enterprise
[Support for UML 2.4.1 metamodel and notation.]()	+	+	+	+	+
Support for UML 2 metamodel and notation.	+	+	+	+	+
Import of UML 1.4 metamodel.	+	+	+	+	+
Class diagram - includes Package and Objects diagrams.	+	+	+	+	+
Composite structure diagram.	+	+	+	+	+
Use Case diagram.	+	+	+	+	+

How are they built?

Business Process Modeling Notation support 1.x*	Personal	Standard	Professional	Architect	Enterprise
Support for Business Process Modeling Notation 1.x (BPMN).		+	+	+	+
Business Process Modeling Notation (BPMN) export to BPEL 1.1 (BEA flavor).				+	+
Business Process Modeling Notation support 2.0	Personal	Standard	Professional	Architect	Enterprise
Support for Business Process Modeling Notation 2.0 (BPMN).		+*	+*	+**	+**
UML extensions (profiles and diagrams)	Personal	Standard	Professional	Architect	Enterprise
[Generic numbering mechanism in DSL models.]()		+	+	+	+
WSDL profile and diagram.				+	+
XML schema profile and diagram.				+	+
CORBA IDL profile and diagram.				+	+
Database structure profile and diagram: Generic DDL and Oracle DDL diagram				+	+
Web Application Extensions (WAE) profile and diagram.		+	+	+	+
Content diagram.		+	+	+	+

A tale of success…

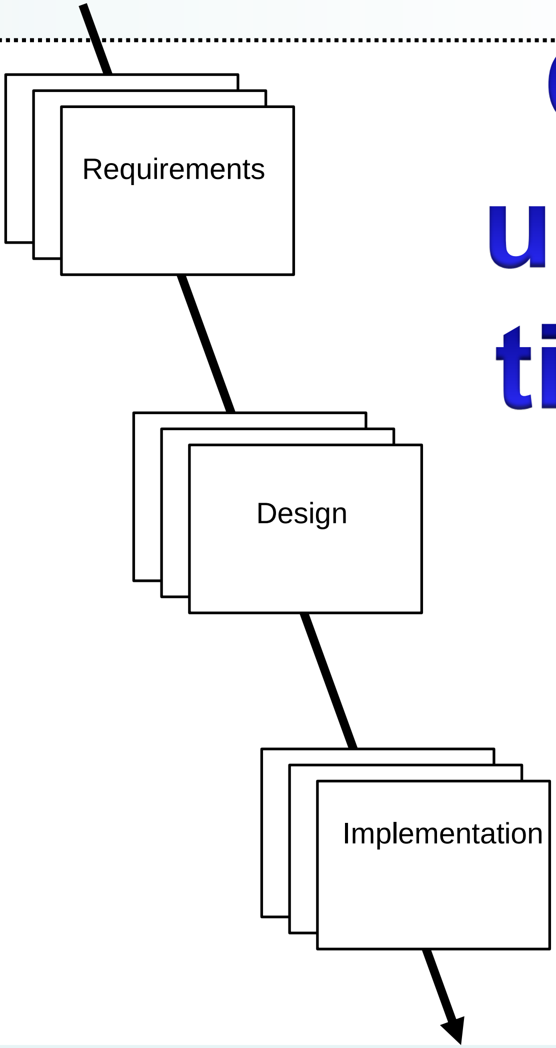
Client A



Requirements Specification

Design

Implementation

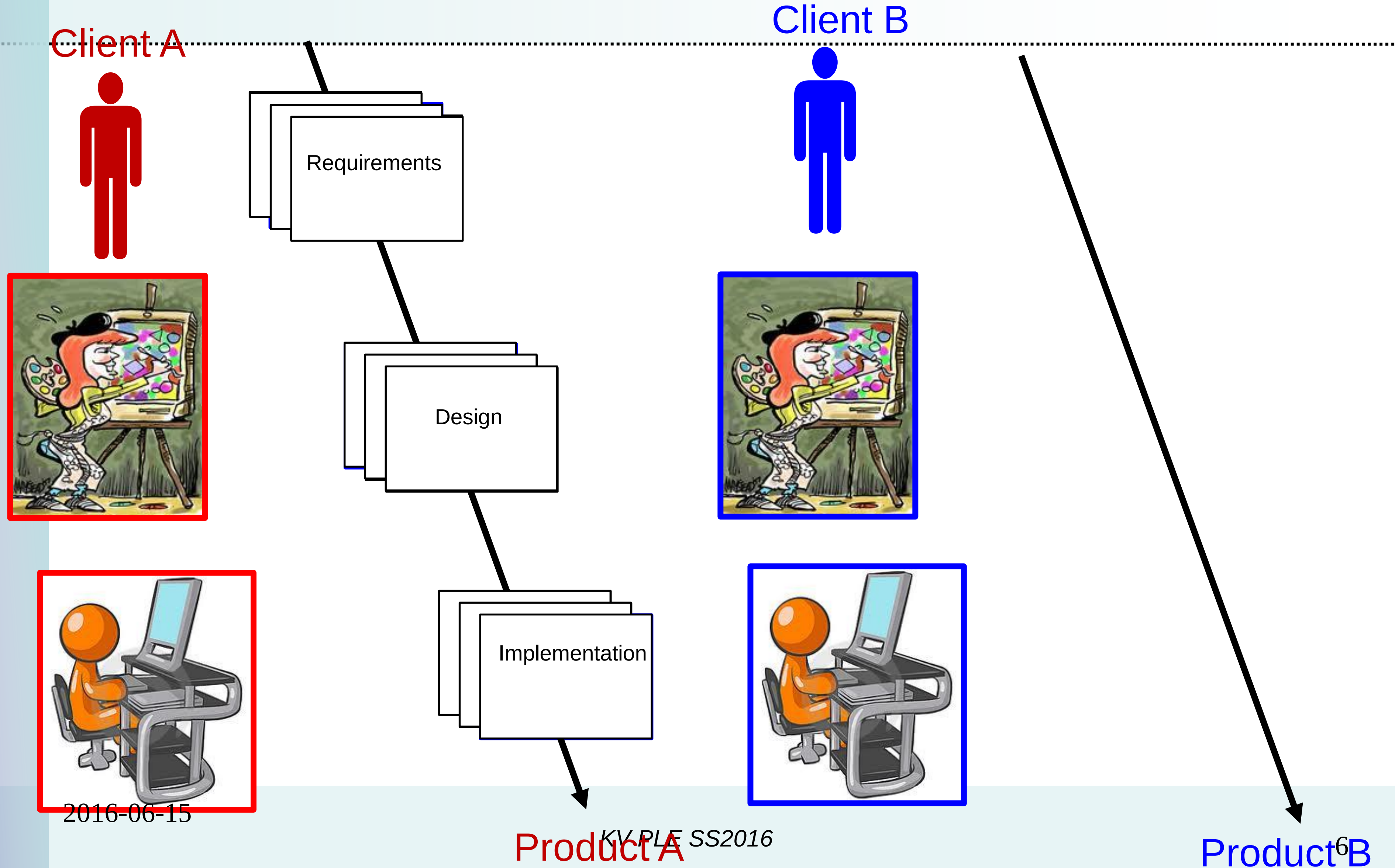


Product A

Once upon a time …

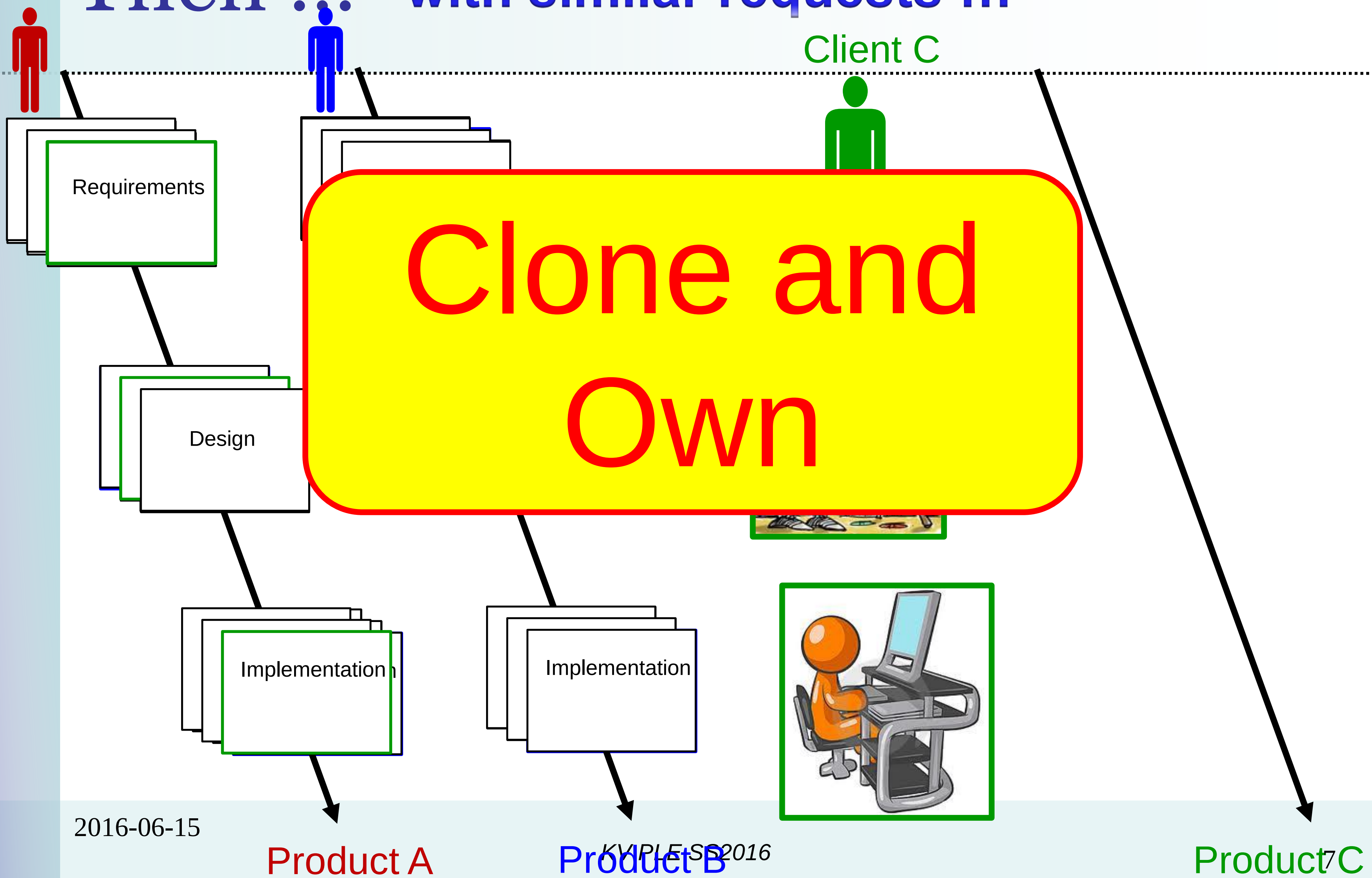
A second client came in …

Then … with similar requests …

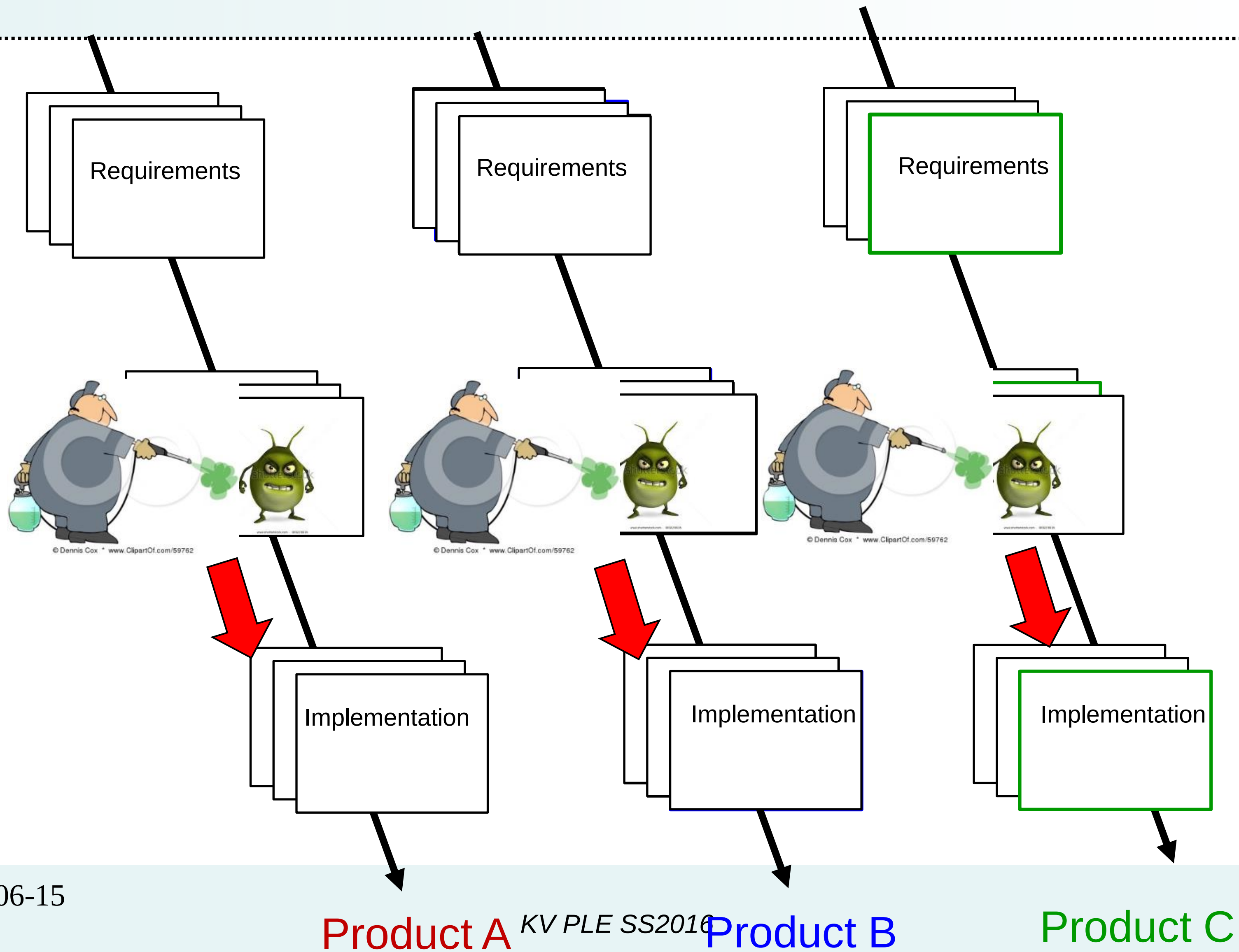


Then ...

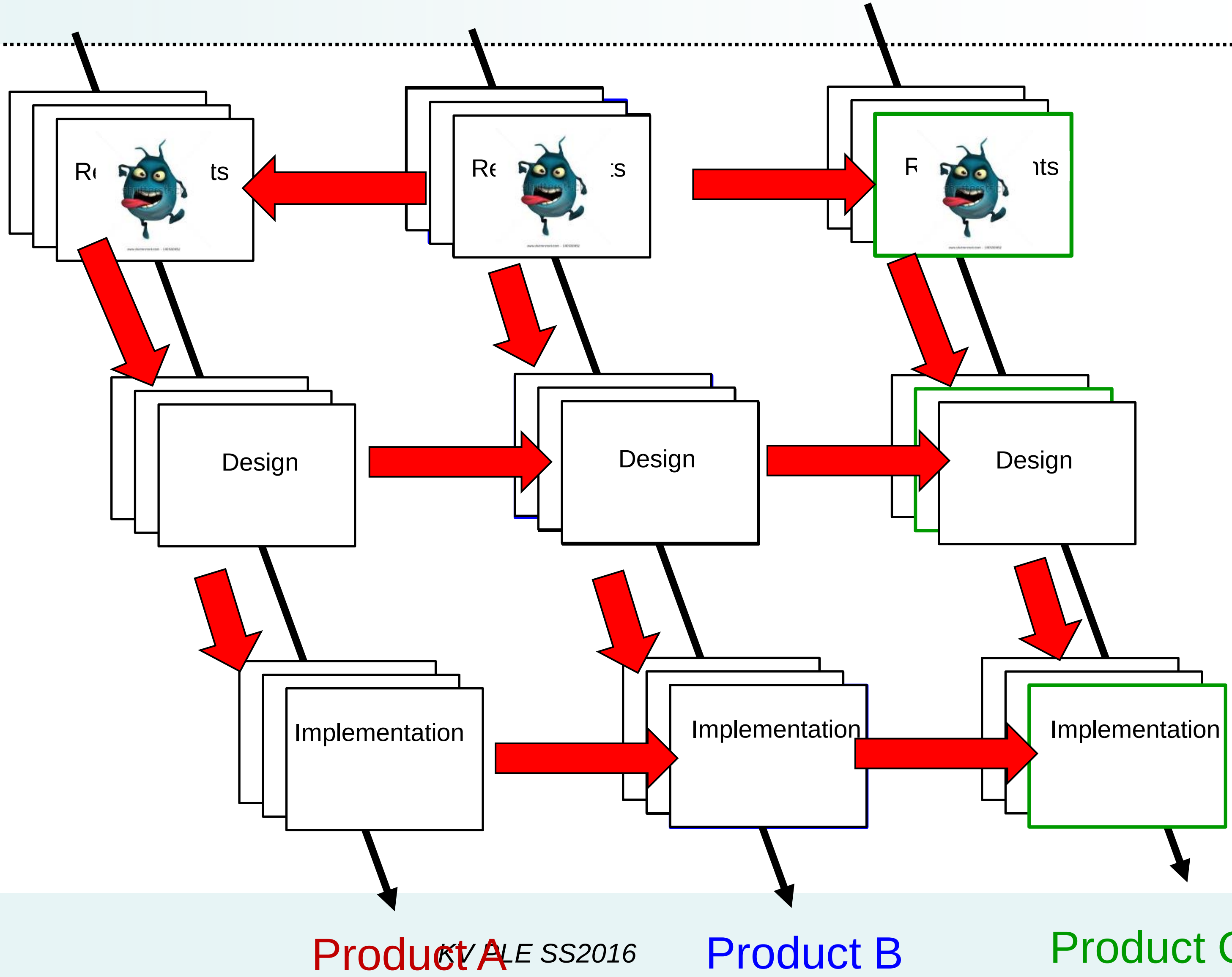
A third client came in …
with similar requests …



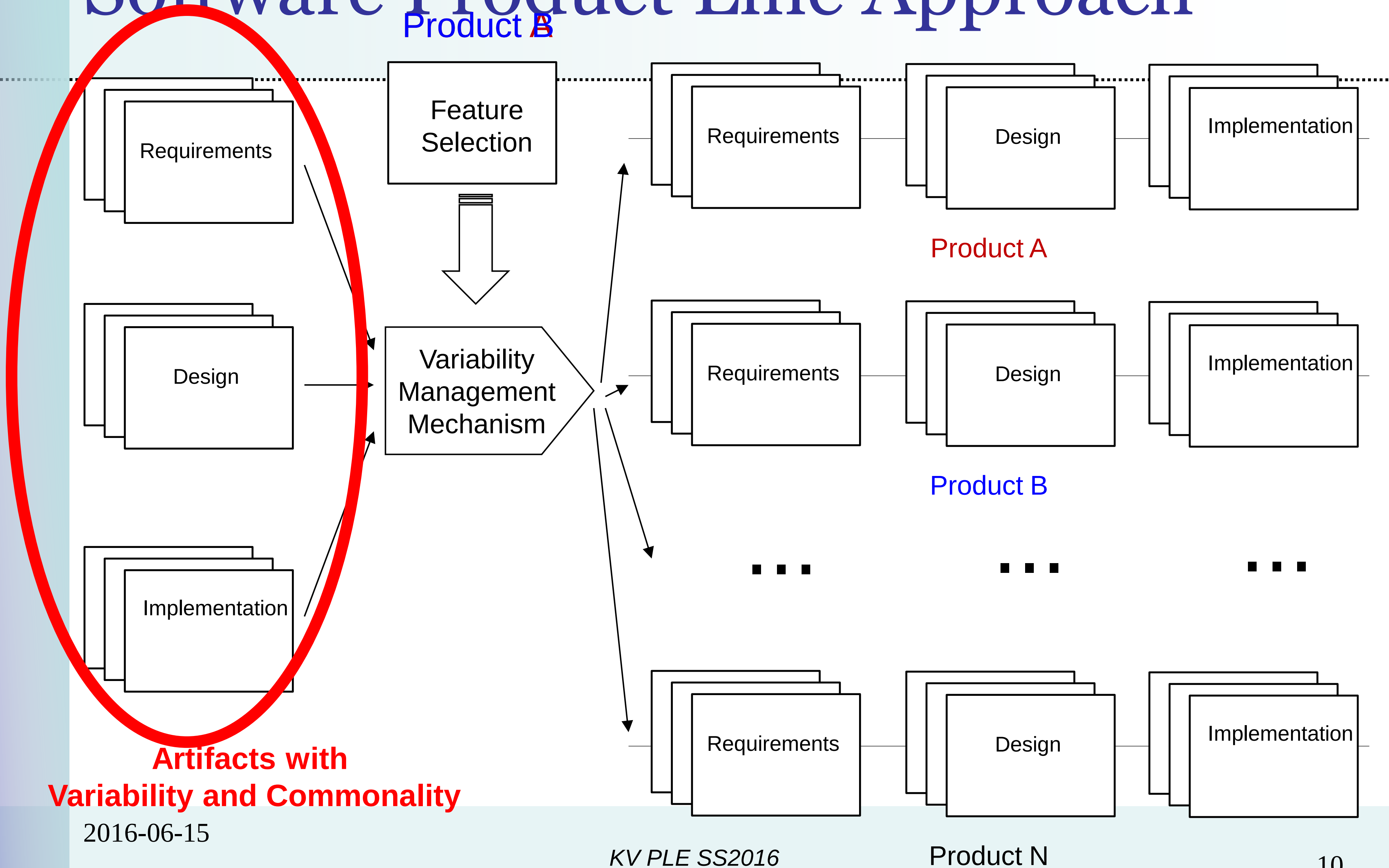
Then problems arise ...



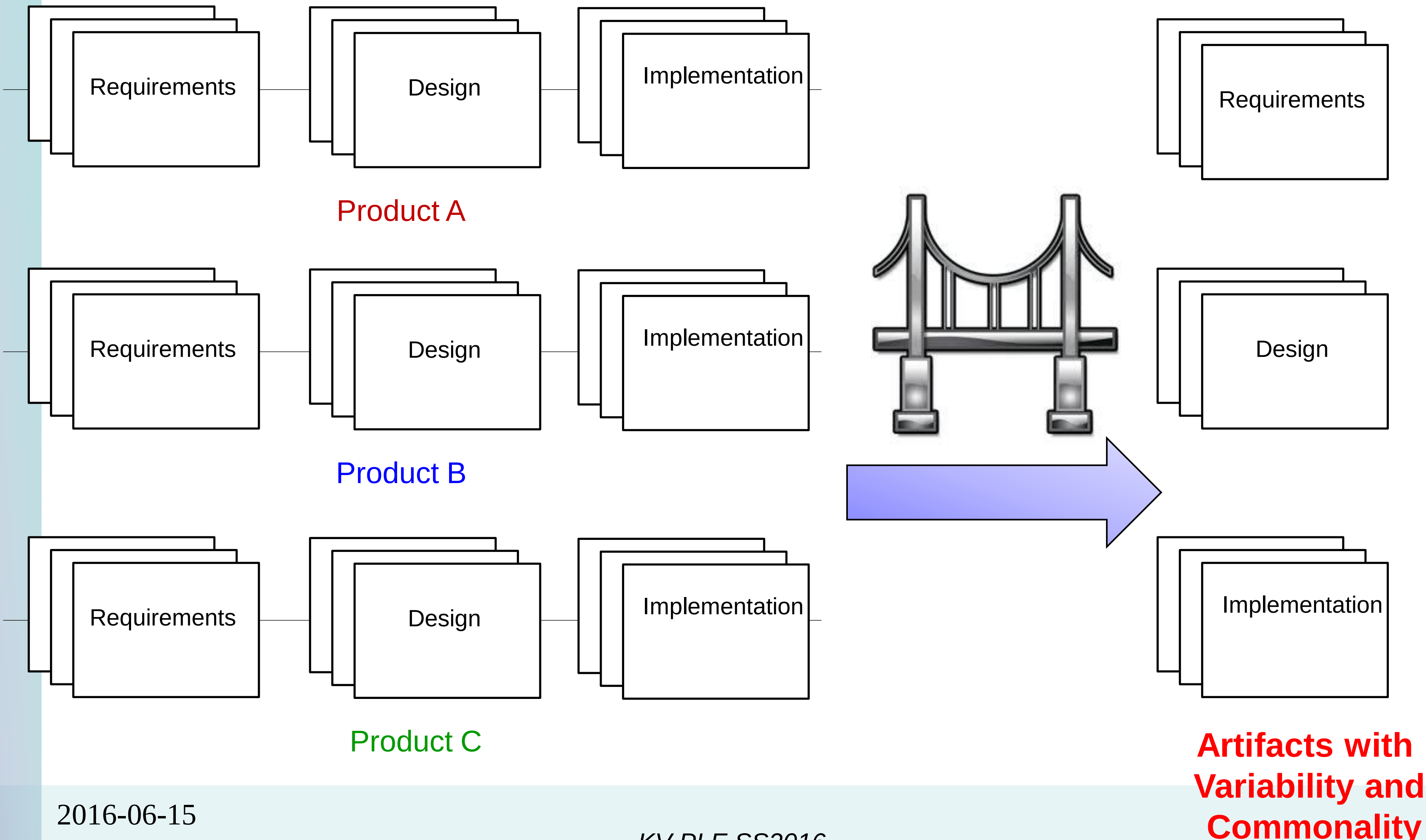
It gets even worse ...



Software Product Line Approach



Reverse Engineering Variability



VARiability In safety-critical Embedded Systems Project

SEARCH-BASED SOFTWARE ENGINEERING (SBSE) – REMINDER

Search-Based Software Engineering (SBSE)

- Search-Based Software Engineering focuses on the application of search-based optimization techniques to problems in software engineering [Harman10]

- Typical techniques are:
  - Basic searches, e.g. hill-climbing, simulated annealing
  - Techniques based on evolutionary computation

Hill Climbing Illustration

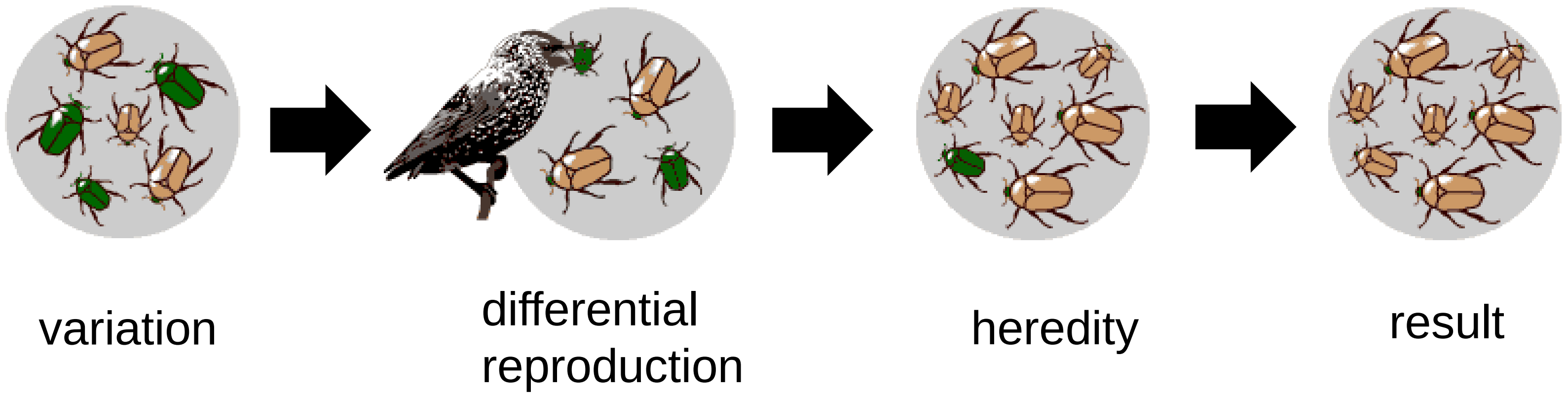
- Looks at a neighborhood of SampleSize states and selects the one with best fitness
- Main problem
  - Can get stuck in a local maxima

Algorithm 1 Steepest Ascent Hill Climbing

```
X ← random initial state
I ← 0
Best ← X
while (I<MaxIter) ∧ (evaluate(Best) ≠ BestFitness) do
    S ← 0
    while S<SampleSize do
        X′ ← move(Best)
        if evaluate(X′) better than evaluate(X) then
            X ← X′
        end if
        S ← S + 1
    end while
    if evaluate(X) better than evaluate(Best) then
        Best ← X
    end if
    I ← I + 1
end while
return Best
```

Basic Ideas [Eiben03]

- Darwinian evolution:
  - Given an enviroment able to host a limited number of individuals
  - Basic instinct of individuals is to reproduce
  - Natural Selection favours those that can compete for the available resources more effectively
    - a.k.a. survival of the fittest



Individuals, Mutation and Populations

- Phenotypic traits
  - Behavioural and physical features of an individual that affect the response to the enviroment → **fitness**
- Mutations
  - Random variations on the phenotypic traits
  - Can be accumulated to new combinations of traits
- Population
  - Consist on a number of individuals
  - After time pases, because of reproduction and mutation the population changes

Evolutionary Computation

- Evolutionary Computation
  - Includes several stochastic search methods which computationally simulate the natural evolutionary process

- Example techniques
  - Genetic algorithms
    - Individuals are typically represented as binary strings, commonly used in numerical optimization problems
  - Genetic programming
    - Individuals encode programs typically represented as tree-structures whose fitness function evaluate how well the programs execute a computational task

Evolutionary Computation Illustration

- Randomly creates an initial population
- Evaluates the initial population
- At each generation
  1. select the individuals with best fitness
  2. mutate their characteristics
  3. re-evaluate them

Algorithm 1 Basic Evolutionary Algorithm

$t \leftarrow 0$

initialize $P(t)$

evaluate $P(t)$

while not *termination* $-$ *condition* **do**

  $t \leftarrow t + 1$

  select $P(t)$ *from* $P(t-1)$

  mutate $P(t)$

  evaluate $P(t)$

end while

CHALLENGES IN RAISING A SOFTWARE FAMILY

Four Core Challenges

1. Know your family members
   - Reverse engineering feature models

2. Know the family members whereabouts
   - Traceability Feature-Artifact

3. Identify boundaries and enforce them
   - Safe composition and feature refactoring

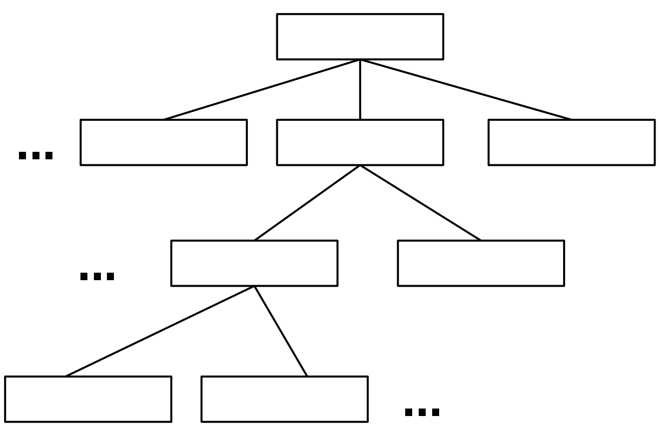
4. Cope with growing pains
   - Evolution and Maintenance

CHALLENGE 1.
KNOW YOUR FAMILY MEMBERS

Big Picture

- Goal: Model all the products of a SPL and their features

Feature Model



Reverse Engineering

non trivial

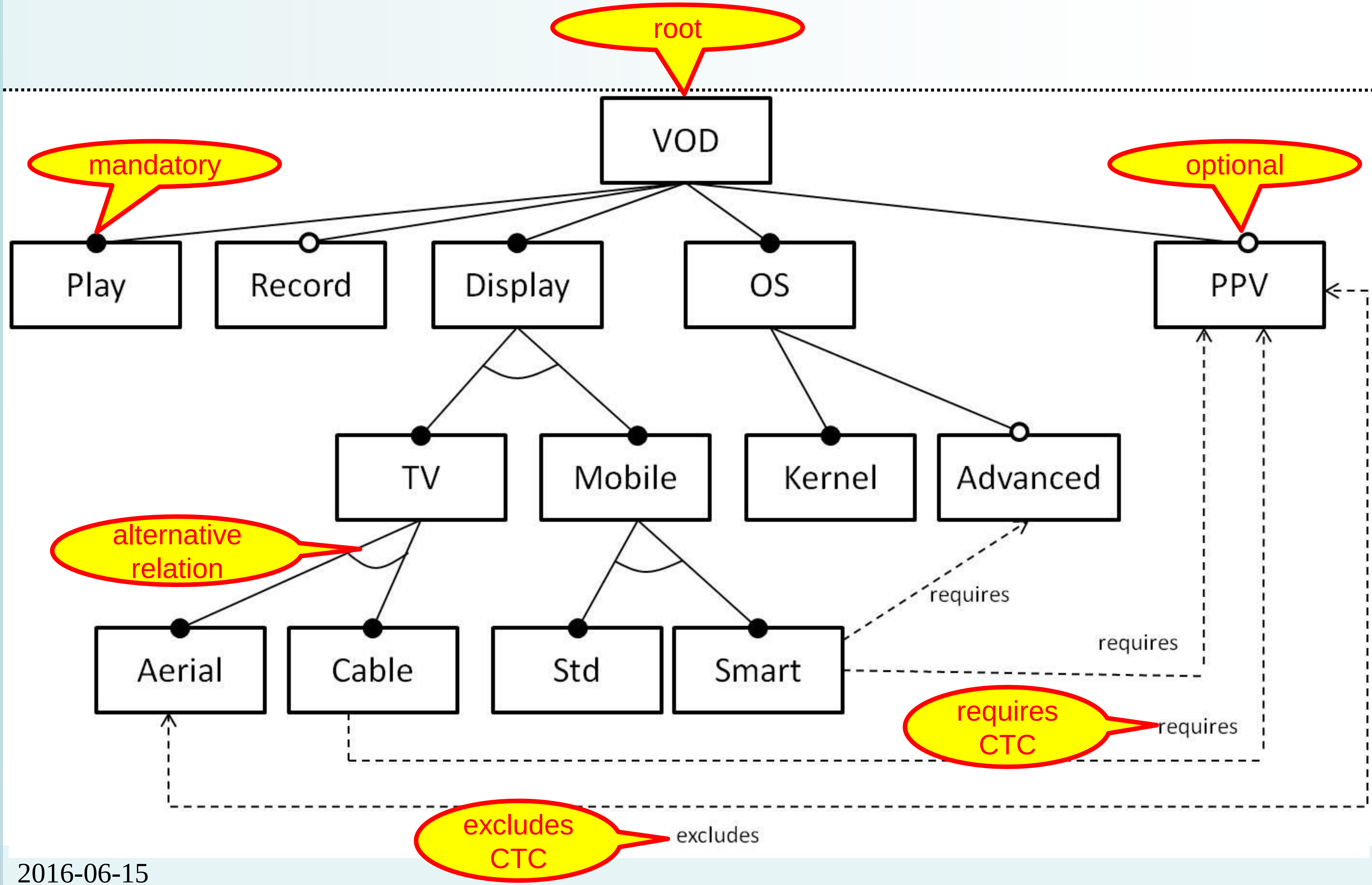
error prone

non unique

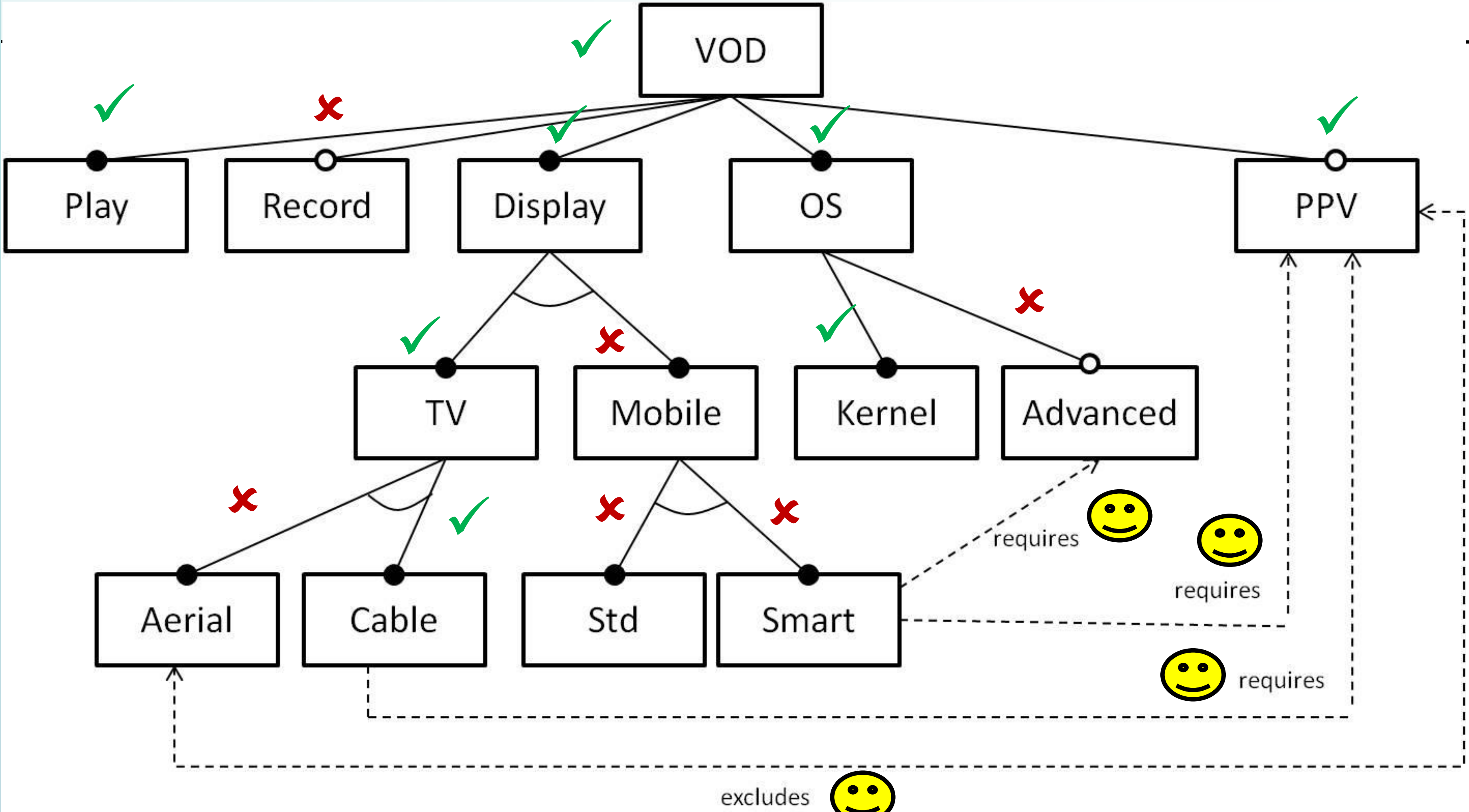
Feature Sets

A	B	C	...	N
✓		✓	…	
	✓		…	✓
✓	✓	✓	…	
		✓	…	✓
	✓	✓	…	✓
✓	✓	✓	…	
…	…	…	…	…
✓	✓	✓	…	✓

Feature Model – Notation Example



Feature Set Example



feature set

Selected {VOD, Play, Display ,OS, TV, Cable, Kernel, PPV}

Not Selected {Record, Mobile, Aerial, Mobile, Std, Smart, Advanced}

Running Example -- Feature Set Table

FSet	VOD	Play	Rec	Disp	OS	TV	Mob	Sm	Std	Ker	Adv	Aer	Cab	PPV
FSI	✓	✓	✓	✓	✓	✓				✓			✓	✓
FS2	✓	✓		✓	✓	✓				✓			✓	✓
FS3	✓	✓	✓	✓	✓	✓				✓		✓		
FS4	✓	✓		✓	✓	✓				✓		✓		
FS5	✓	✓	✓	✓	✓		✓		✓	✓				✓
FS6	✓	✓	✓	✓	✓		✓		✓	✓				
FS7	✓	✓		✓	✓		✓		✓	✓				
FSS	✓	✓		✓	✓		✓		✓	✓				✓
FS9	✓	✓	✓	✓	✓	✓				✓	✓		✓	✓
FS1O	✓	✓		✓	✓	✓				✓	✓		✓	✓
FS11	✓	✓	✓	✓	✓	✓				✓	✓	✓		
FS12	✓	✓		✓	✓	✓				✓	✓	✓		
FS13	✓	✓	✓	✓	✓		✓		✓	✓	✓			✓
FS14	✓	✓	✓	✓	✓		✓		✓	✓	✓			
FS15	✓	✓		✓	✓		✓		✓	✓	✓			✓
FS16	✓	✓		✓	✓		✓		✓	✓	✓			
FS17	✓	✓	✓	✓	✓		✓	✓		✓	✓			✓
FS18	✓	✓		✓	✓		✓	✓		✓	✓			✓

Our contributions

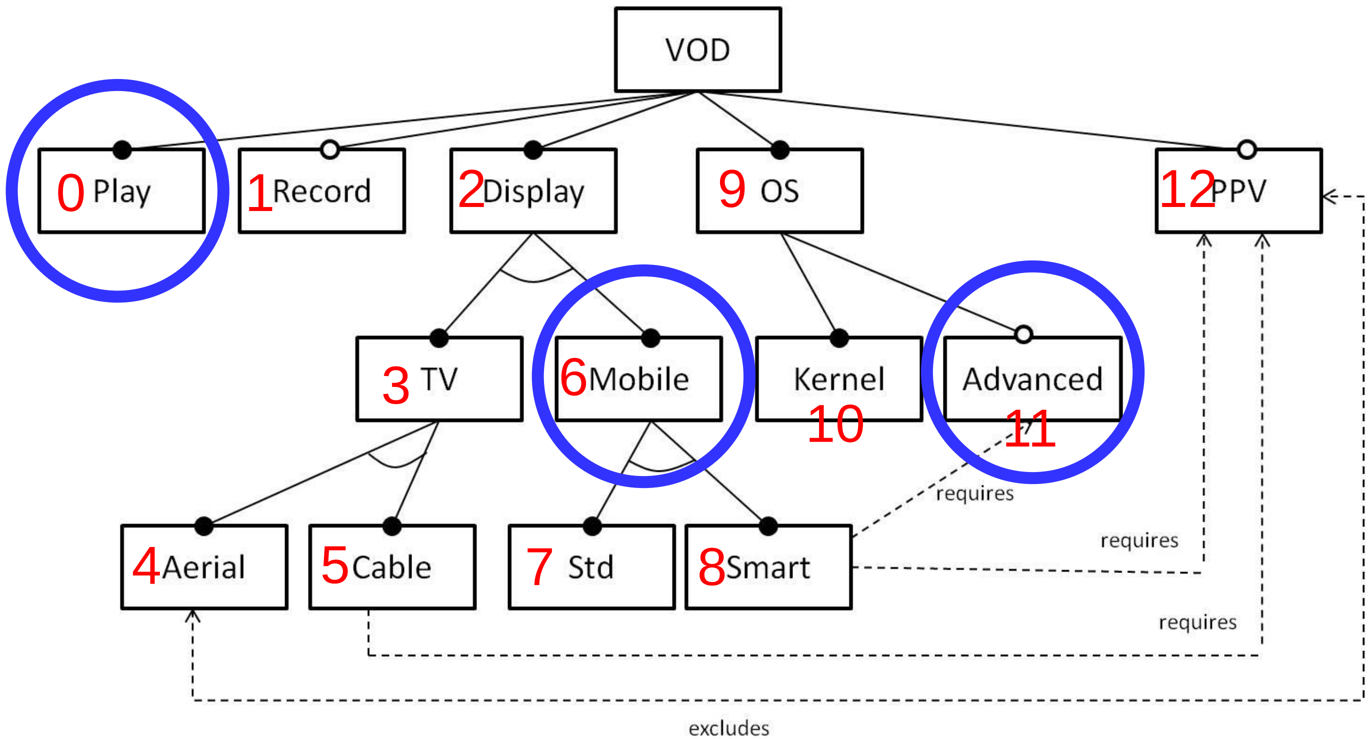
- An ad hoc algorithm [WCRE11, FASE2013]
  - Provides *one* feature model solution with arbitray feature hierarchy

- Search-based approach based on genetic algorithm [SBSE12]
  - Can provide more than one solution alternatives

Feature Model Encoding (1)

- First part encodes the structure of the feature model
- Each chromosome is a tuple <PR, CN>
  - PR relation with parent
    - M – mandatory
    - Op – optional
    - Alt – alternative relation
    - Or – or relation
  - CN denotes the number of children
- A Depth-First Traversal determines the tuple order
  - Starting from the root of FM tree
  - The root is not encoded

Structural Encoding Example



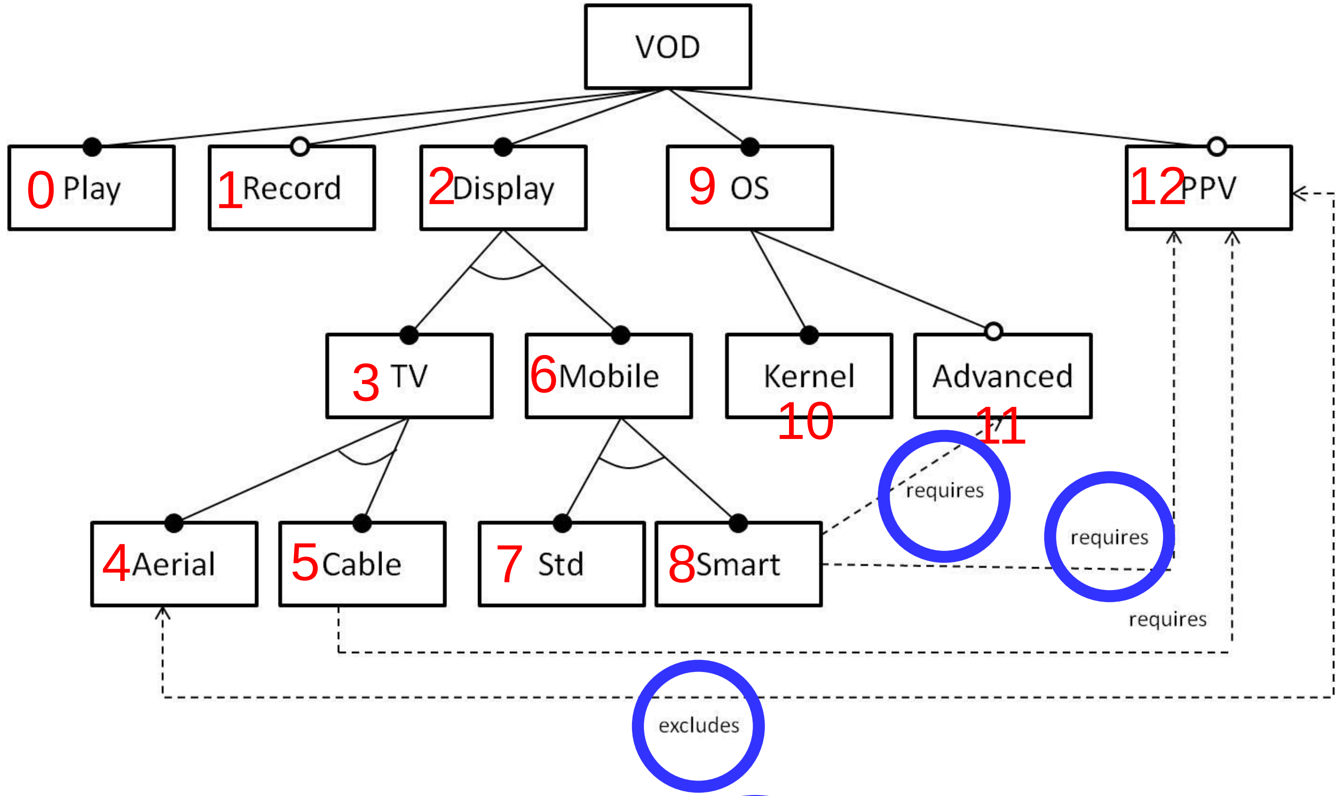
0	1	2	3	4	5	6	7	8	9	10	11	12
M,0	Op,0	M,2	Alt,2	Alt,0	Alt,0	Alt,2	Alt,0	Alt,0	M,2	M,0	Op,0	Op,0

Feature Model Encoding (2)

- Second part encodes the Cross-Tree Constraints

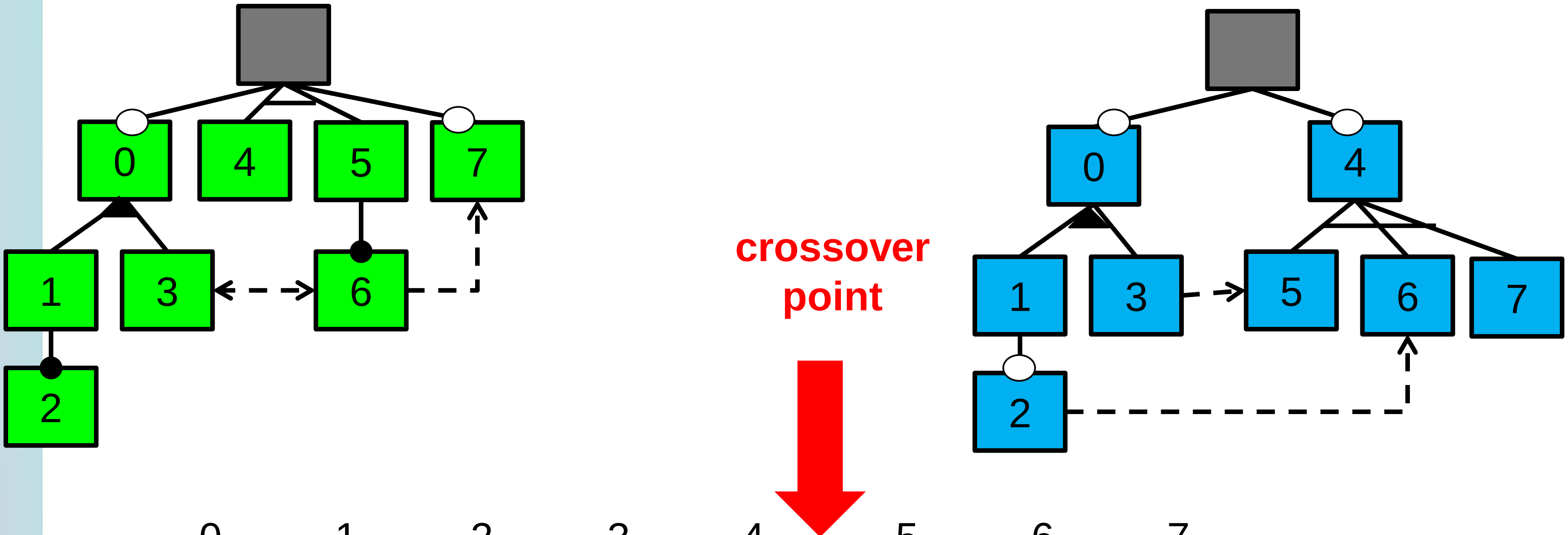
- Each chromosome is a tuple <TC, O, D>
  - TC – type of constraint
    - R – requires
    - E – excludes
  - O – origin feature denoted with DFT value
  - D – destination feature denoted with DFT value

CTC Encoding Example



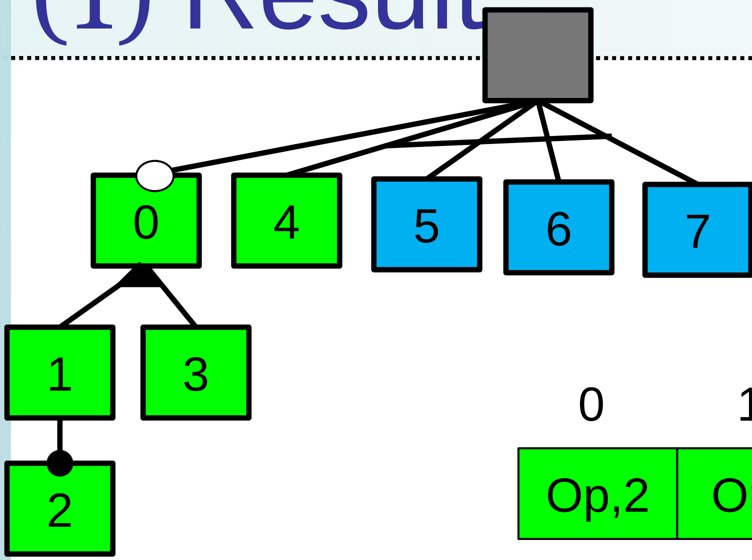
E,4,12	R,5,12	R,8,12	R,8,11

Crossover — One point
(1) Feature Diagram

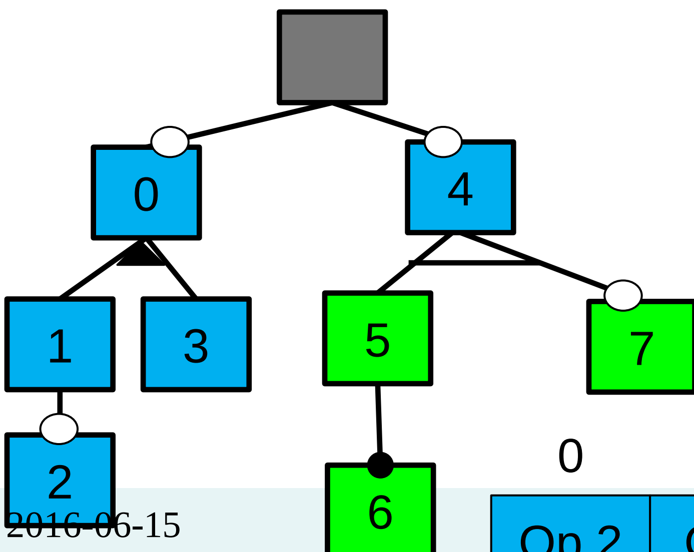


0	1	2	3	4	5	6	7
Op,2	Or,1	M,0	Or,0	Alt,0	Alt,1	M,0	Op,0
Op,2	Or,1	Op,0	Or,0	Op,3	Alt,0	Alt,0	Alt,0

Crossover — One point (1) Result

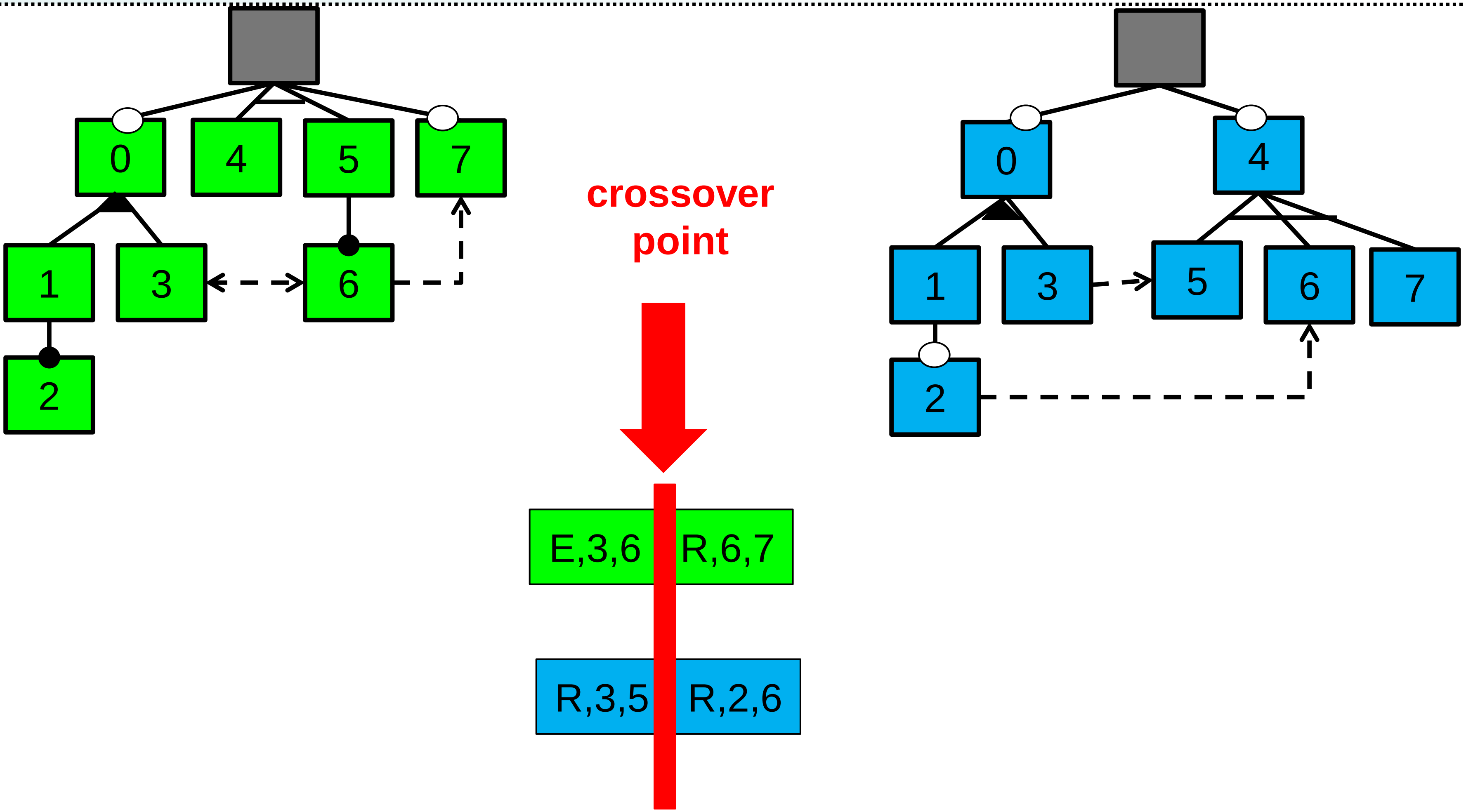


0	1	2	3	4	5	6	7
Op,2	Or,1	M,0	Or,0	Alt,0	Alt,0	Alt,0	Alt,0

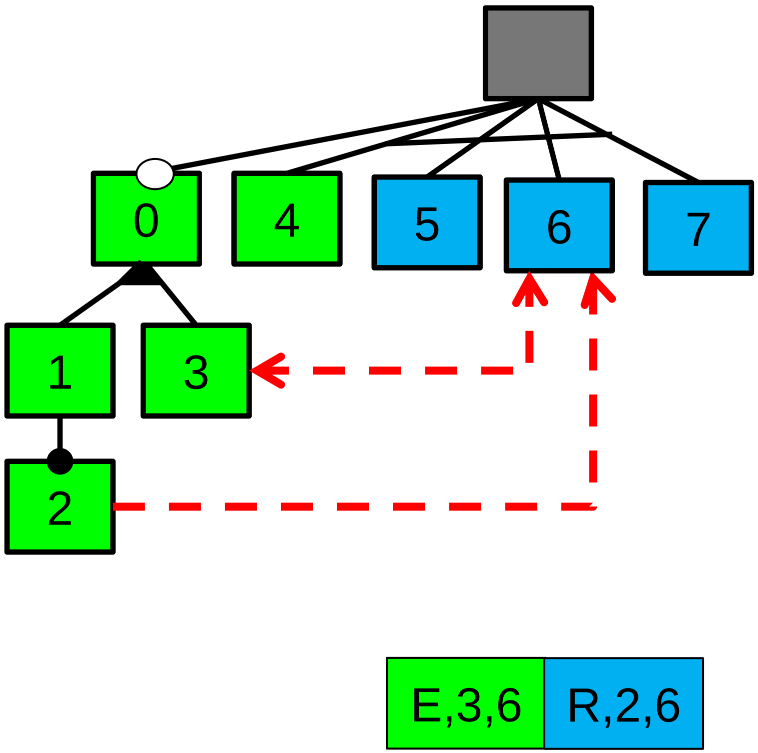


0	1	2	3	4	5	6	7
Op,2	Or,1	Op,0	Or,0	Op,3	Alt,1	M,0	Op,0

Crossover – One point (2) Cross-Tree Constraints



Crossover — One point (2) Result



Mutation Operators

- Four operators applied with a configurable probability
  - Operator 1. Changes randomly a relation between two features from one kind to any other kind. For example, from mandatory (M) to optional (Op) or from Op to Alternative (Alt).
  - Operator 2. Changes the number of children CN, to a number selected from 0 to a maximum branching factor parameter set up.
  - Operator 3. Changes the type of cross-tree constraint, from excludes to requires and vice versa.
  - Operator 4. Changes either the origin or destination feature (with equal probability) of a cross-tree constraint.
- Validity checks
  - Identification and repair of infeasible feature models
  - E.g. A CTC does not have the same origin and destination values.

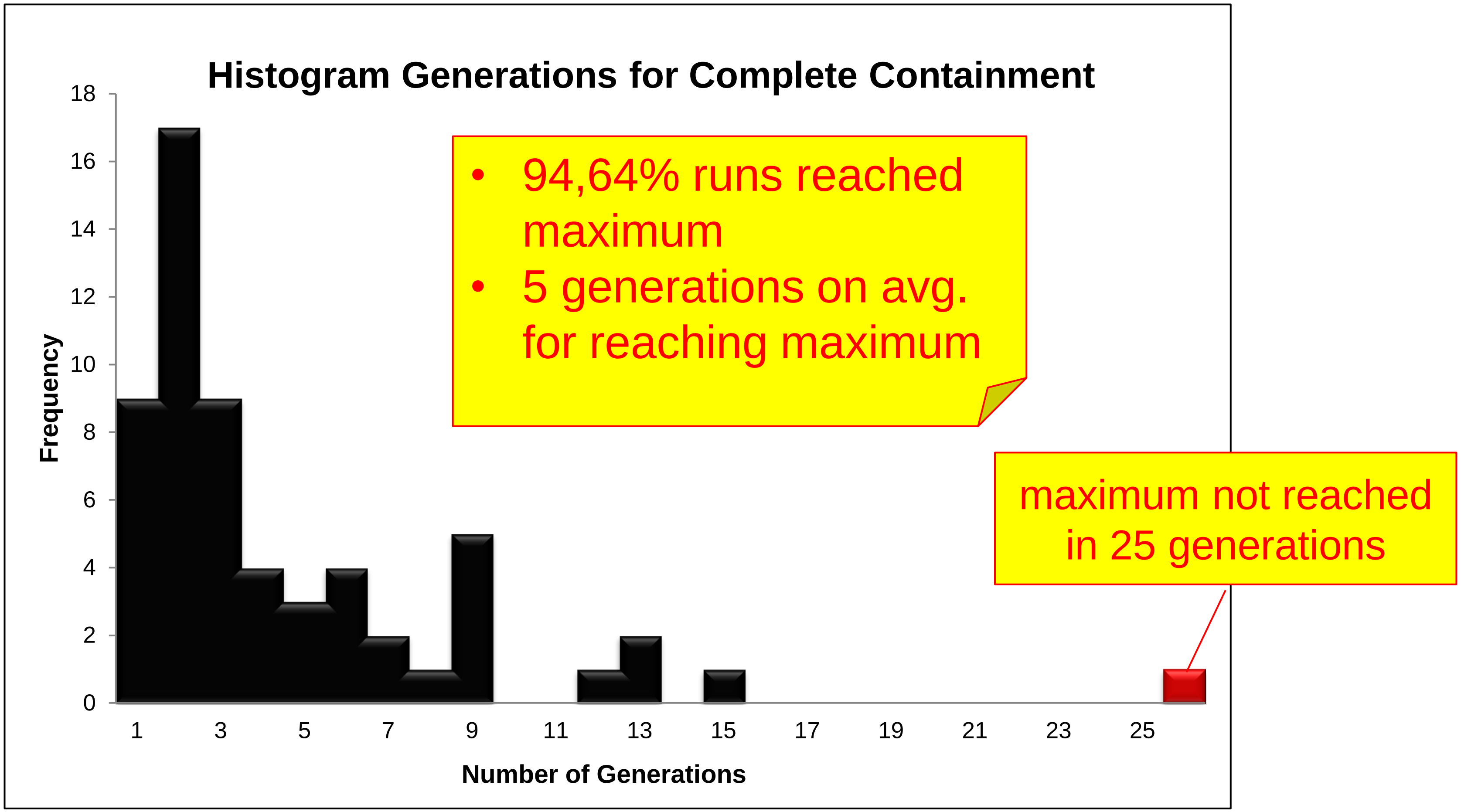
Evaluation Overview

$$FFRelaxed(sfs, fm) = containedFSets(sfs, fm)$$

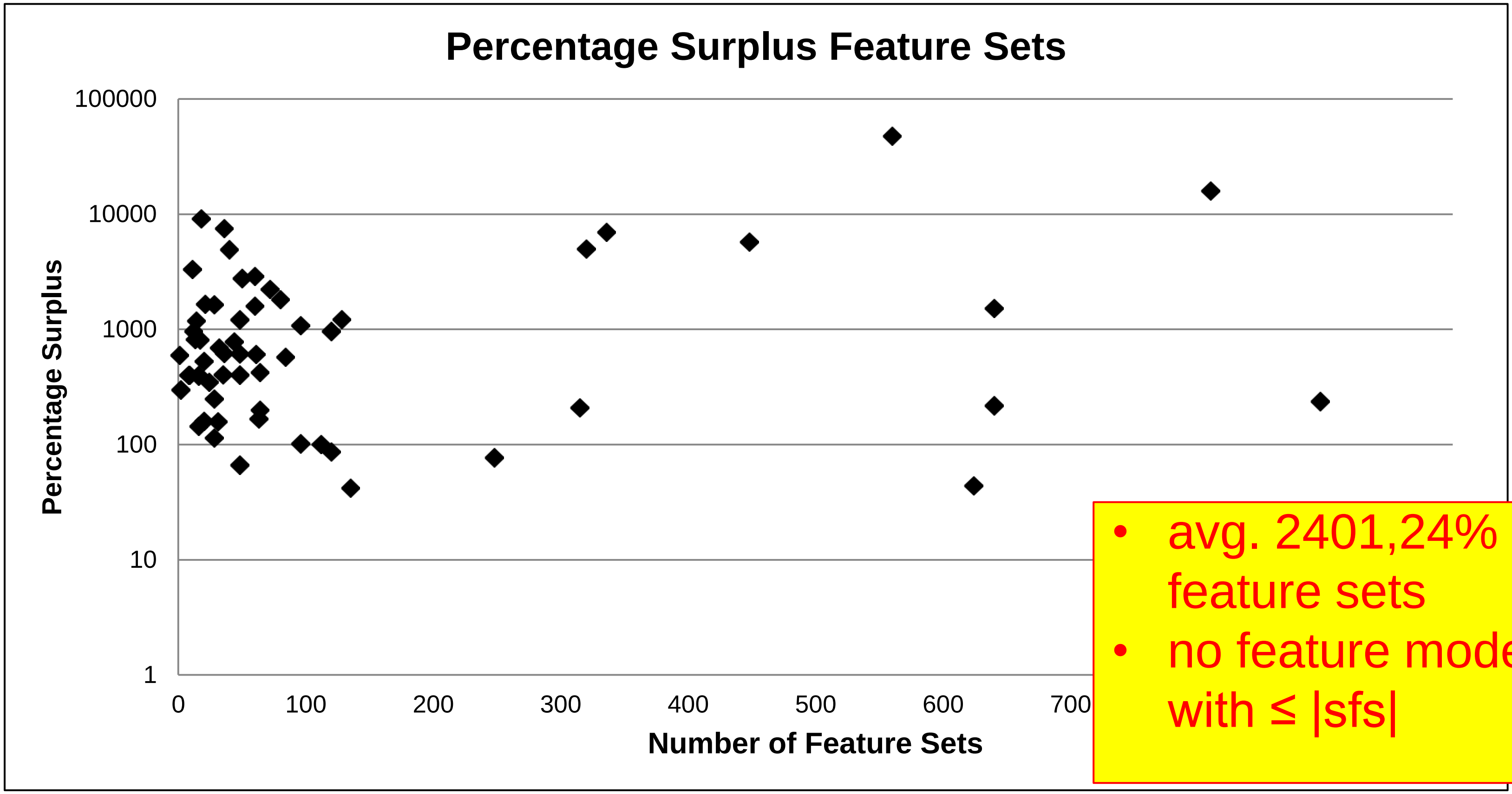
- Case studies
  - 59 feature models from SPLOT repository
  - No. products 1...896
  - No. features 9 ... 27
- Executions
  - 10 runs for each feature model
  - 16 cores at 2.40 GHz, 25GB RAM, Cent OS, Java 1.6

Parameter	Value
Selection strategy	Roulette-wheel
Crossover strategy	One-point
Crossover probability	0.7
Mutation probability	0.01
Initial population size	100
Infeasible individuals	Replace
Maximum generations	25

FFRelaxed Results (1)



FFRelaxed Results (2)



- avg. 2401,24% more feature sets
- no feature model with ≤ |sfs|

$$\text{Surplus}(sfs,fm) = \frac{\text{products}(fm) - |sfs|}{|sfs|} \times 100$$

Open questions

- Non-binary feature property combinations
  - Not only yes/no but other real values, e.g. non-functional properties
- Effective use of domain knowledge to structure the feature hierarchy
  - For example based on ontologies
- More expressive feature model representations and operators
  - Genetic programming, variability-aware operators

CHALLENGE 2.
KNOW THE FAMILY MEMBERS WHEREABOUTS

Big Picture

- Goal:
  - Compute traces between features and the realization artifacts
- Our contribution
  - Basic algorithm to incrementally trace features and feature interactions to artifact fragments [SPLC13]

First Variant Example – V1 (LINE)

```
class Line {
   Point startPoint, endPoint;
   Line(Point start) {...}
   void paint(Graphics g) {  …  }
   void setEnd(Point end) {...}
}
class Canvas {
   List<Line> lines =  new LinkedList<Line>();
   void paintComponent(Line  l)   { … }
}
```

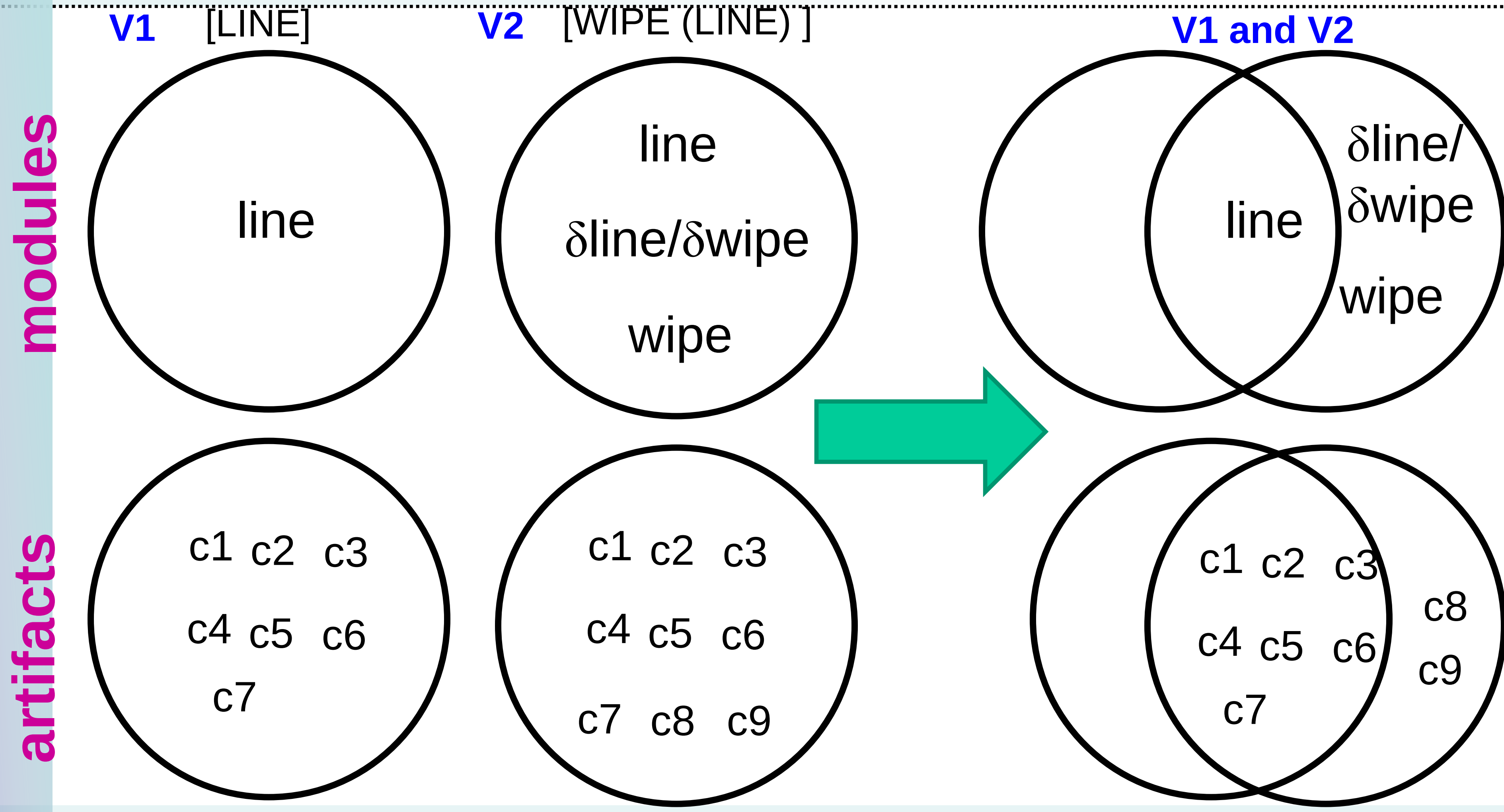
	Variant V1
c1	Point Line.startPoint
c2	Point Line.endPoint
c3	Line.Line(Point)
c4	void Line.paint(Graphics)
c5	void Line.setEnd(Point)
c6	List Canvas.lines
c7	void Canvas.paintComponent(Line)

Second Variant Example – V2 (LINE, WIPE)

```
class Line {
   Point startPoint, endPoint;
   Line(Point start) {...}
   void paint(Graphics g) {…}
   void setEnd(Point end) {... }
}
class Canvas {
   List<Line> lines =  new LinkedList<Line>();
   void paintComponent(Line l)       {…}
  void wipe() {
    lines.clear();
    repaint();
   }
}
```

	Variant V2
c1	Point Line.startPoint
c2	Point Line.endPoint
c3	Line.Line(Point)
c4	void Line.paint(Graphics)
c5	void Line.setEnd(Point)
c6	List Canvas.lines
c7	void Canvas.paintComponent(Line)
c8	void Canvas.wipe()
c9	lines.clear()

First Traceability Refinement

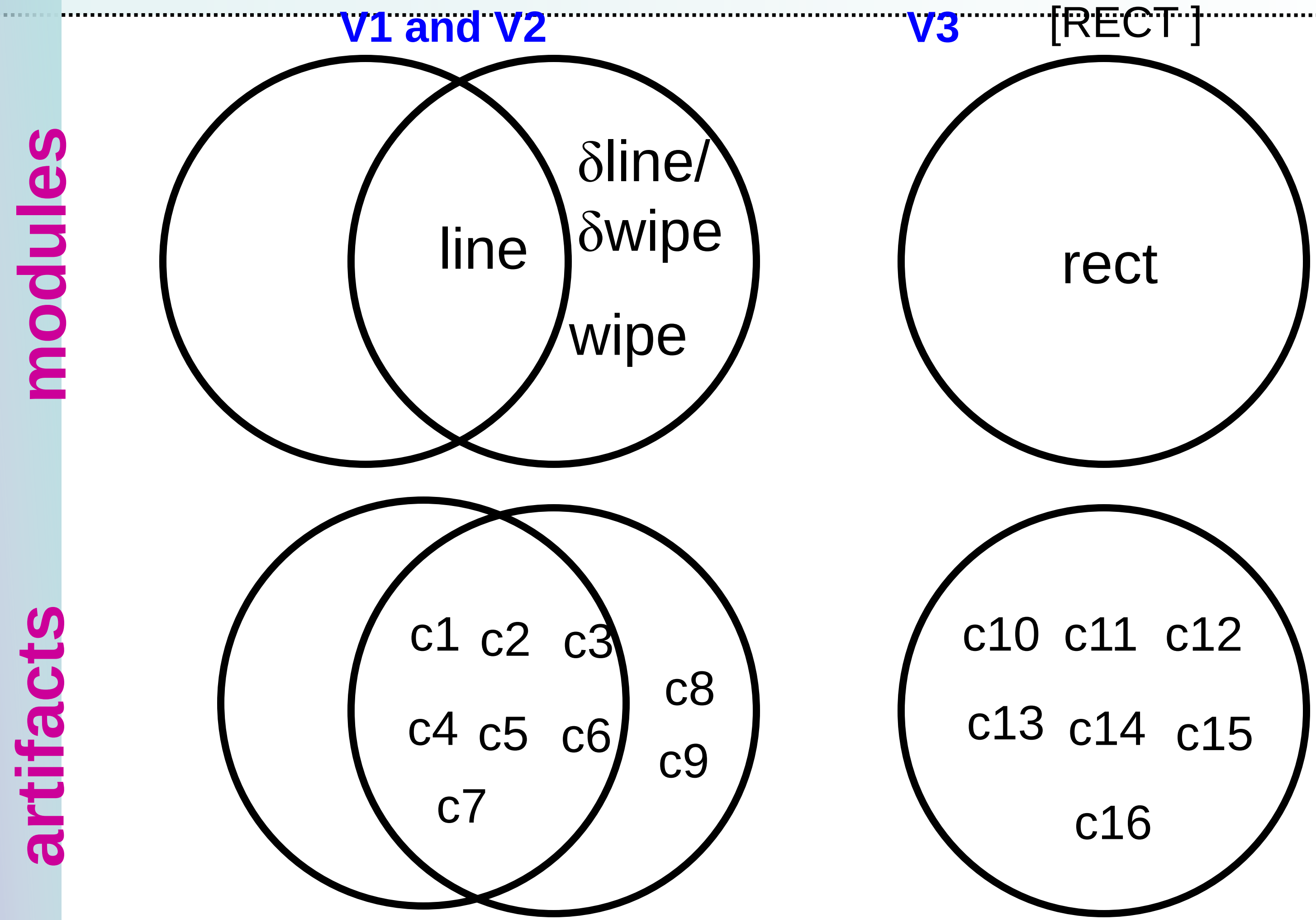


Third Variant Example – V3 (RECT)

```
class Rectangle {
   Point  upperPoint, lowerPoint;
   Rectangle(Point  x,  Point y) { ... }
   void paint(Graphics g) { … }
   void setEnd(Point x) { ... }
}
class Canvas {
 List<Rectangle> rectangles = new
LinkedList<Rectangle>();
 void paintComponent(Rectangle rect)  { ... }
}
```

	Variant V3
c10	Point Rectangle.upperPoint
c11	Point Rectangle.lowerPoint
c12	Rectangle.Rectangle(Point, Point)
c13	void Rectangle.paint(Graphics)
c14	void Rectangle.setEnd(Point)
c15	List Canvas.rectangles
c16	void Canvas.paintComponent(Rectangle)

Second Traceability Refinement

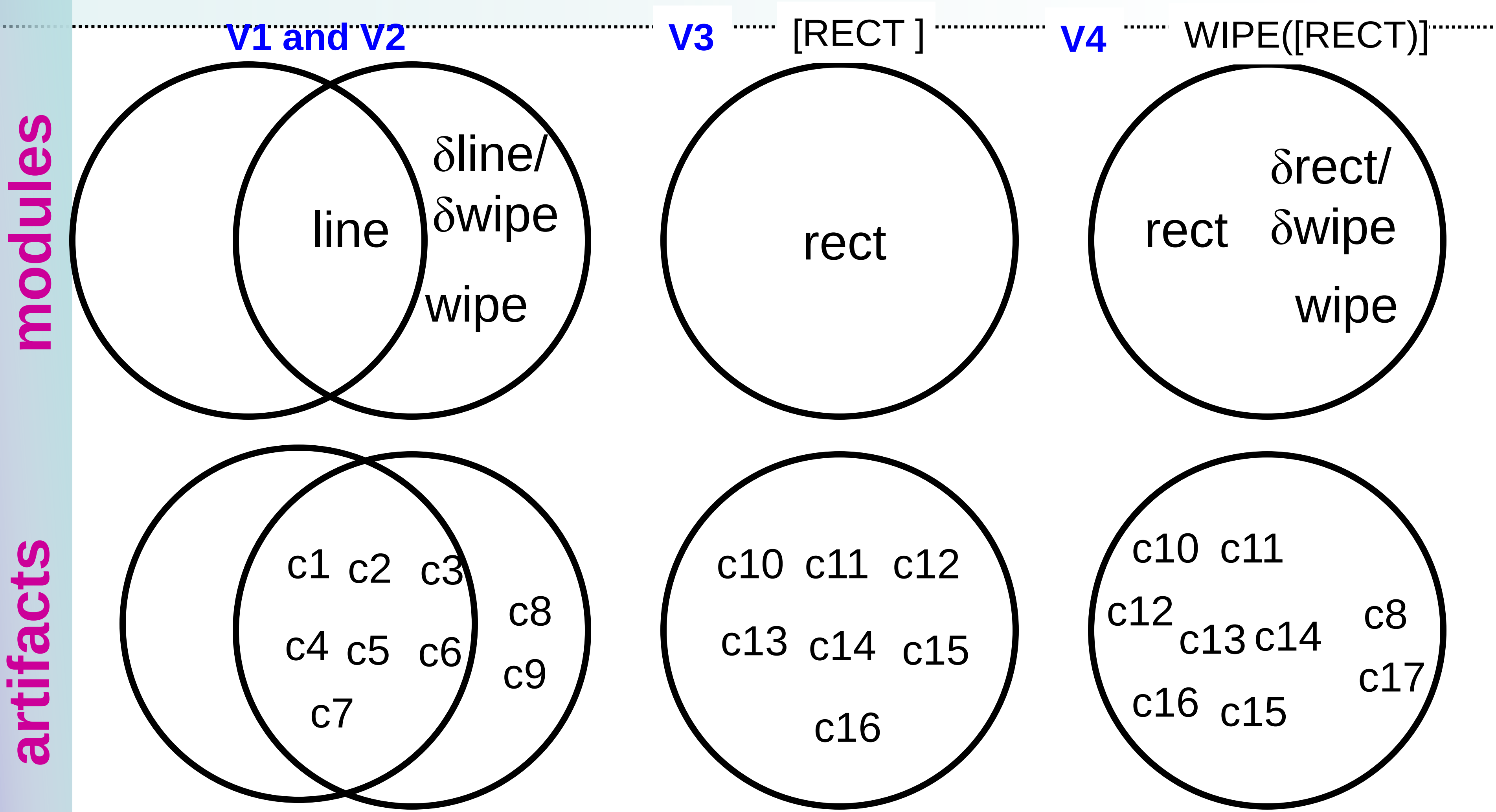


Fourth Variant Example – V4 (RECT, WIPE)

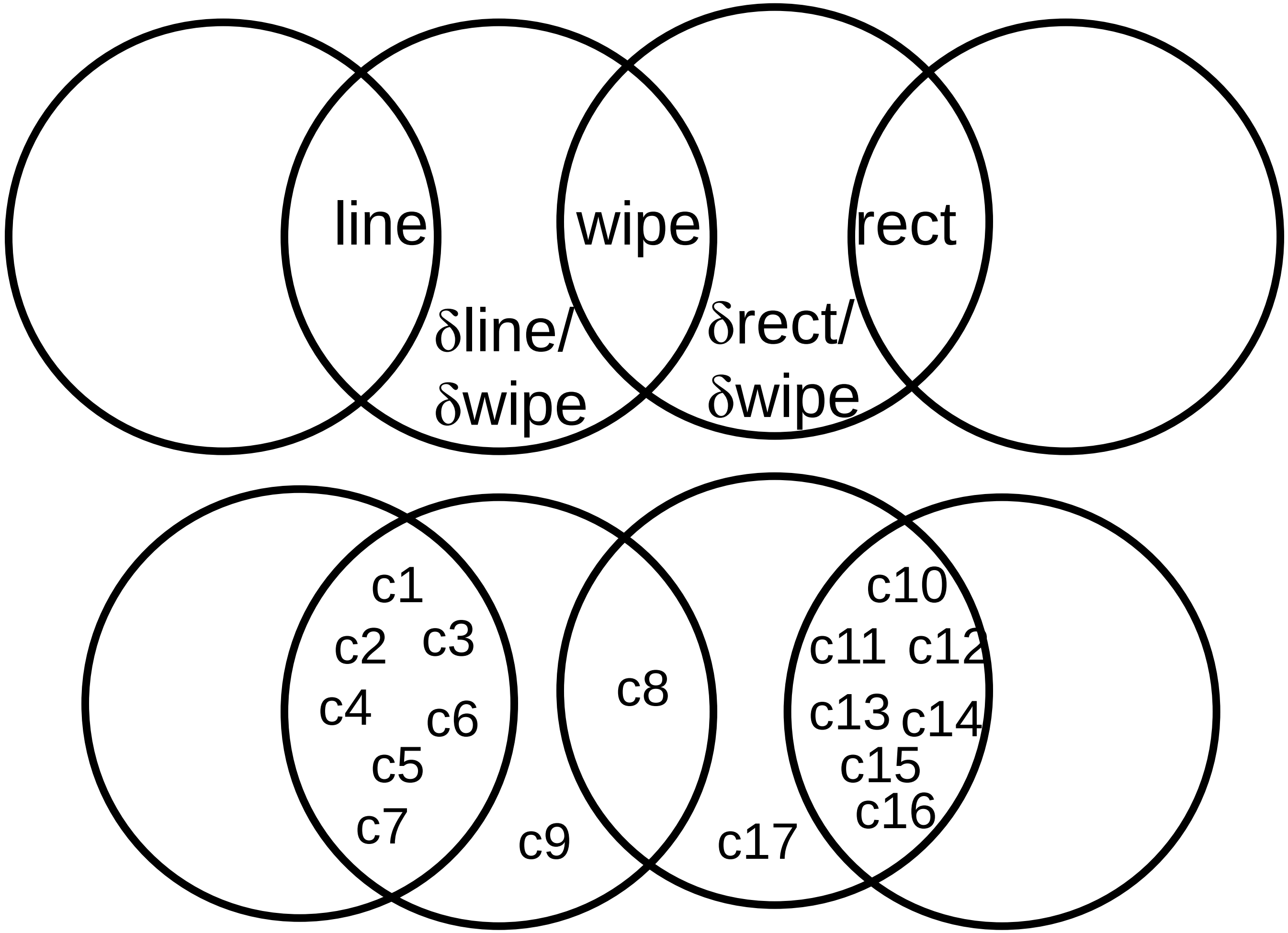
```
class Rectangle {
   Point  upperPoint, lowerPoint;
   Rectangle(Point  x,  Point y) { ... }
   void paint(Graphics g) { … }
   void setEnd(Point x) { ... }
}
class Canvas {
 List<Rectangle> rectangles = new
LinkedList<Rectangle>();
 void paintComponent(Rectangle rect)  { ... }
 void wipe() {
     rectangles.clear();
     repaint();
   }
}
```

	Variant V4
c10	Point Rectangle.upperPoint
c11	Point Rectangle.lowerPoint
c12	Rectangle.Rectangle(Point, Point)
c13	void Rectangle.paint(Graphics)
c14	void Rectangle.setEnd(Point)
c15	List Canvas.rectangles
c16	void Canvas.paintComponent(Rectangle)
c8	void Canvas.wipe()
c17	rectangles.clear();

Third Traceability Refinement (1)



Third Traceability Refinement (2)



Evaluation Examples

	VOD	ArgoUML	MM
Mandatory Features	6	3	6
Optional Features	5	8	8
Number of Variants	32	256	7
Lines of Code	5.3K+	340K+	5K+
Classes	42	1915	50
Fields	392	4452	223
Methods	249	16676	422
Unique Code Pieces	641	21128	645
Correctness %	100	99.4	99.6

Limitations

- Current evaluation
  - Based on synthesized variants from annotated programs
  - More „realistic“ and larger examples are being collected
- Coarse grain level – field, methods, basic interactions
  - Type 1 – clones
- Current implementation only for Java-based systems
  - Working on extensions for EMF-based representation
- Current algorithm does no exploit
  - Runtime information – execution traces
  - Development history
  - More advanced diffing and clone detection technologies

CHALLENGE 3.
IDENTIFY BOUNDARIES AND ENFORCE THEM

Big Picture

- We want to answer the following questions:

1. How can we find if there is an error in a product?
   - For example, that we do not have references to non-existing elements?
2. What can we do if we find an error in a product?
   - Can we „fix“ it? How?

Our contributions

- Safe composition for multi-view models [ECMFA10, ICSR11]
  - Detection of inconsistency in models with variability
- Catalogue of Feature Oriented Refactoring patterns [SPLC11]
  - Patterns for moving code fragments across feature boundaries

Safe Composition in MVM (1)

- Multi-View Modeling (MVM)
  - Common modeling practice
  - Use of multiple yet related views is advocated
  - Example: UML multiple views
- Consistency checking
  - Description and verification of semantic relationships among views
- Challenge
  - How to detect inconsistencies in MVM ***with*** variability?
- Our approach
  - Safe composition – programming languages
    - guarantee that ***all*** programs of a product line are type safe
  - ***All models that can be composed in a product line are type safe***

Safe Composition in MVM (2)

- Let

  $\mathtt{PL_f}$ domain constraints from the product line

  $\mathtt{IMP_f}$ an implementation constraint in MV models

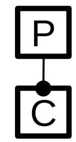
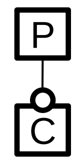
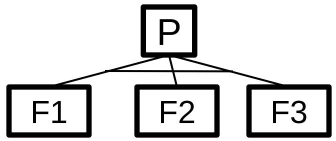
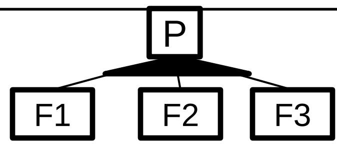
- Using a SAT solver we can check if ***one*** propositional formula is satisfiable or not
- Our interest is verifying that ***all*** the product line members satisfy an implementation constraint

$$\neg (\mathtt{PL_f} \Rightarrow \mathtt{IMP_f})$$

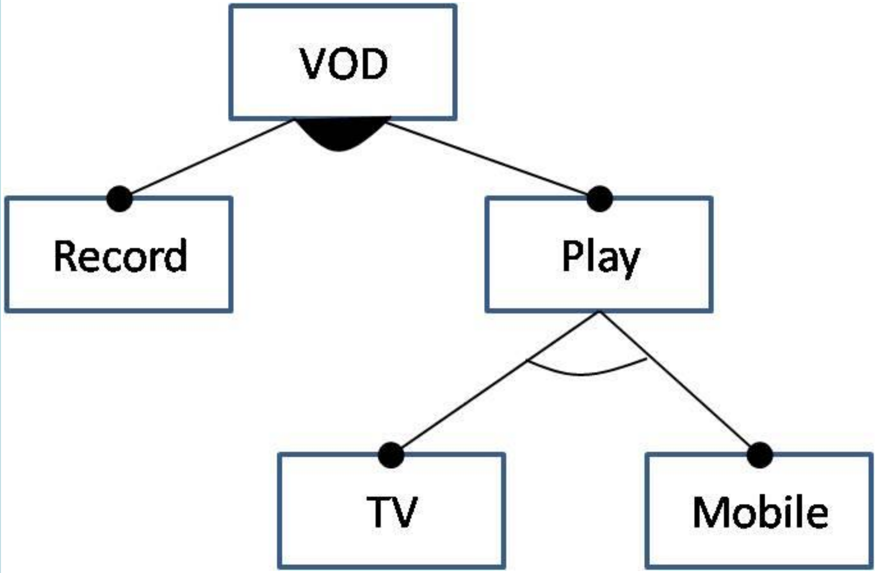
Unsatisfiable = there is <u>no</u> product that violates the constraint

Satisfiable = there is <u>at least one</u> product that violates the constraint

Computing $\mathtt{PL}_{\mathtt{f}}$

Name	**Diagram Notation**	**Propositional Logic**
Mandatory		$P \Leftrightarrow C$
Optional		$C \Rightarrow P$
Alternative		$(F1 \Leftrightarrow (\neg F2 \wedge \neg F3 \wedge P)) \wedge$ $(F2 \Leftrightarrow (\neg F1 \wedge \neg F3 \wedge P)) \wedge$ $(F3 \Leftrightarrow (\neg F1 \wedge \neg F2 \wedge P))$
Or		$P \Leftrightarrow F1 \vee F2 \vee F3$
Requires	Cross feature arrow	$A \Rightarrow B$
Excludes	Cross feature arrow	$A \Rightarrow \neg B \equiv \neg(A \wedge B)$

Example PL_f



$\mathtt{VOD} \Leftrightarrow \mathtt{true} \quad \wedge$

$\mathtt{VOD} \Leftrightarrow \mathtt{Record} \vee \mathtt{Play} \quad \wedge$

$\mathtt{TV} \Leftrightarrow \neg\mathtt{Mobile} \wedge \mathtt{Play} \quad \wedge$

$\mathtt{Mobile} \Leftrightarrow \neg\mathtt{TV} \wedge \mathtt{Play}$

How to compute $\mathtt{IMP}_f$?

- In MVM consistency rules
  - Establish semantic relationships among elements
  - We regard them as implementation constraints
- We identified classes of rules depending on their consistency validation
- We create one $\mathtt{IMP}_f$ for each constraint instance that we need to check

Requiring Rules

- Each constraint instance $\mathtt{IMP_f}$ is computed as follows

$$\mathtt{IMP_f} \equiv \mathtt{F} \Rightarrow \bigvee_{\mathtt{i=1..k}} \mathtt{Freq_i}$$

where

$\mathtt{F}$ is the feature that requires another feature(s)

$\mathtt{Freq_i}$ are the features that satisfy requirement of $\mathtt{F}$

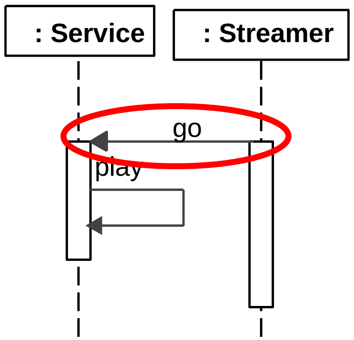
- Considering $\mathtt{PL_f}$

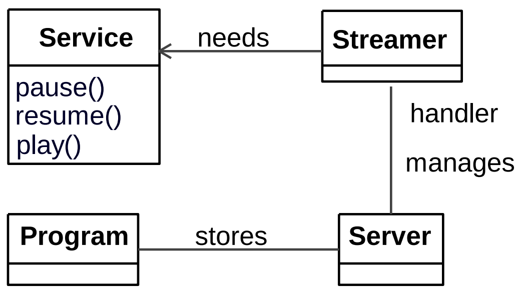
$$\neg(\mathtt{PL_f} \Rightarrow \mathtt{IMP_f}) \equiv \mathtt{PL_f} \wedge \mathtt{F} \bigwedge_{\mathtt{i=1..k}} \neg\mathtt{Freq_i}$$

JOHANNES KEPLER UNIVERSITY LINZ

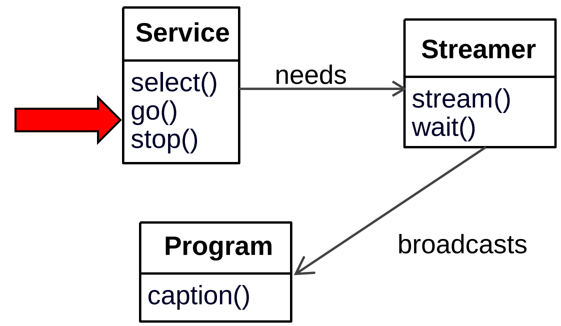
Requiring Rule Example

- R5. Message action must be defined as an operation in receiver's class





Feature Play



Feature VOD

$$IMP_f \equiv \underbrace{Play \Rightarrow VOD}$$

$$\neg(PL_f \Rightarrow IMP_f) \longrightarrow \boxed{\text{SAT}} \longrightarrow \text{false} \checkmark$$

$VOD \Leftrightarrow true \wedge$

$VOD \Leftrightarrow Record \vee Play \wedge$

$TV \Leftrightarrow \neg Mobile \wedge Play \wedge$

$Mobile \Leftrightarrow \neg TV \wedge Play$

Variable assignments that yield true signal the faulty feature combinations

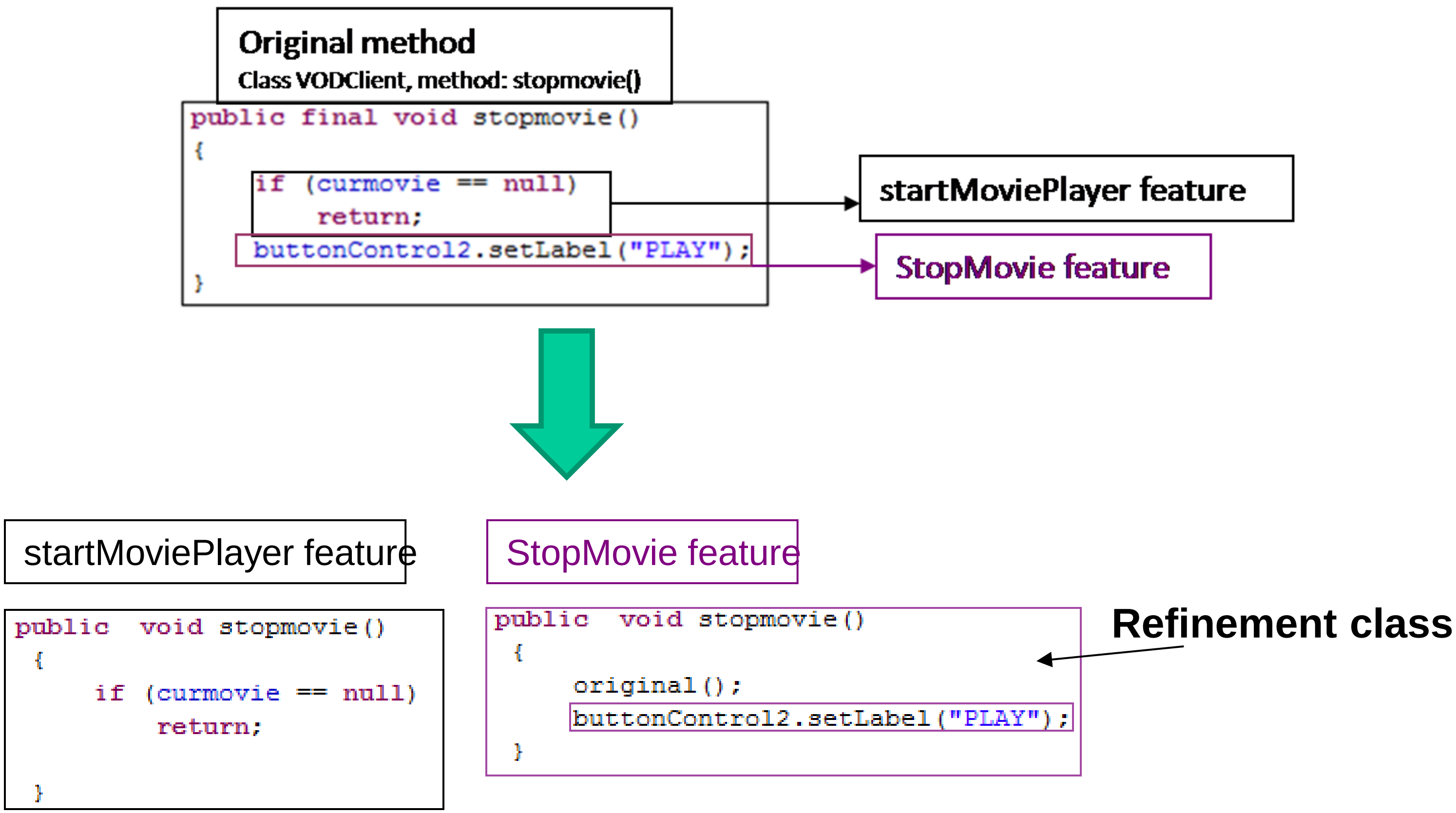
Feature Oriented Refactoring Patterns

- We created two SPLs from two existing single programs by taking them appart into features
  - Starting with list of requirements, each describing a feature
  - Target features modularized with FOSD

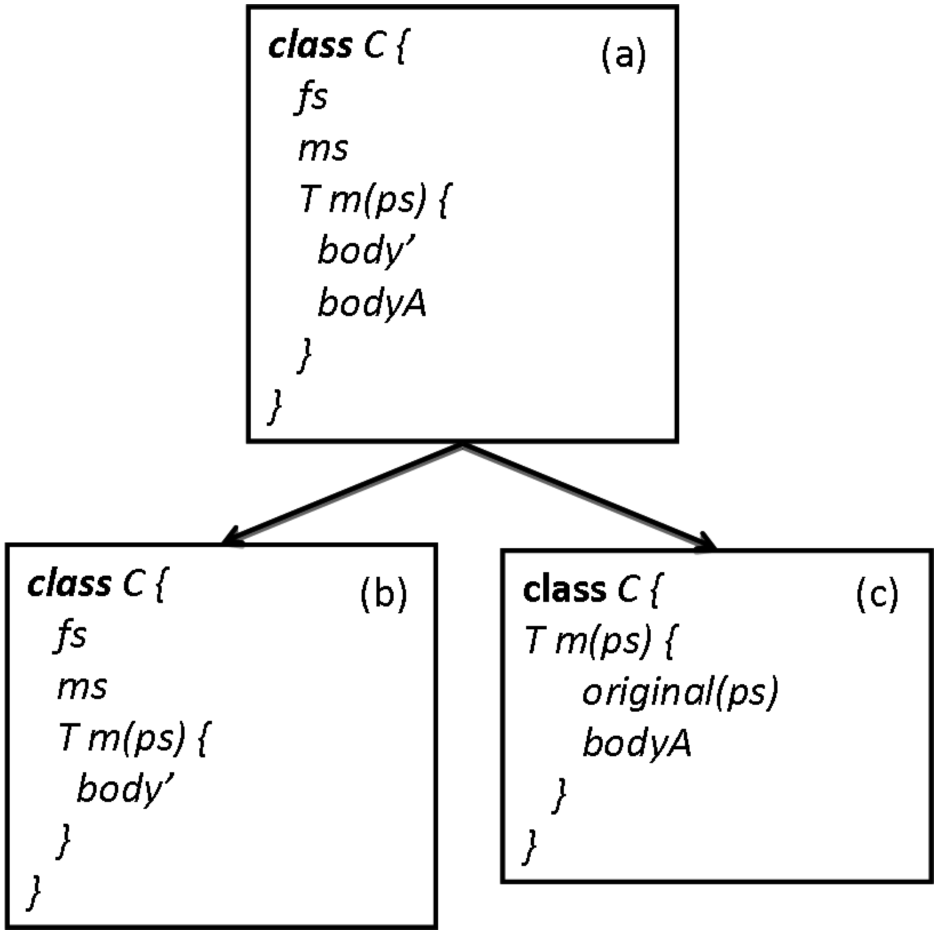
Num.	Level	Name of refactoring
1	**Method**	**Addition at the end of the method**
2	Method	Addition anywhere with a hook method
3	Method	Addition at the beginning of the method
4	Method	Overwrite method
5	Method	Move entire method
6	Attribute	Move field
7	Attribute	Remove access declaration
8	Class	Move entire class

Pattern Example

- Refactoring at end of the method



Pattern Example Overview



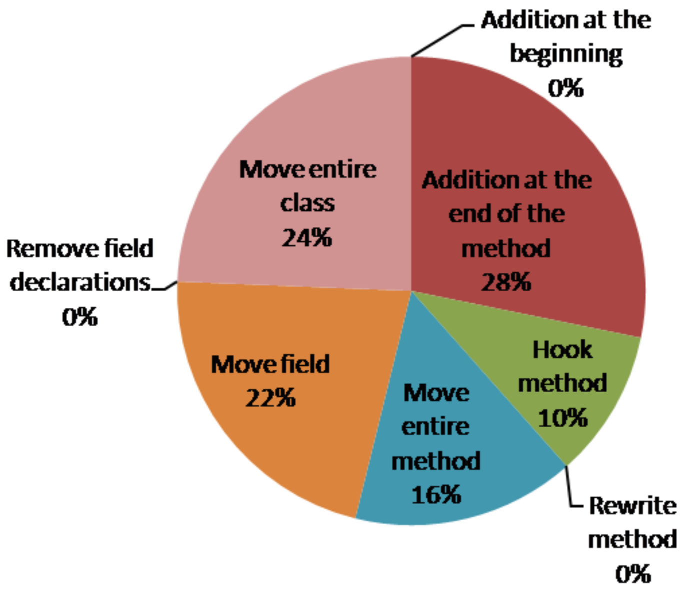
Constraints

-bodyA cannot use variables defined by body'
-bodyA uses variable in ps if not modified by body'

Summary of Case Studies

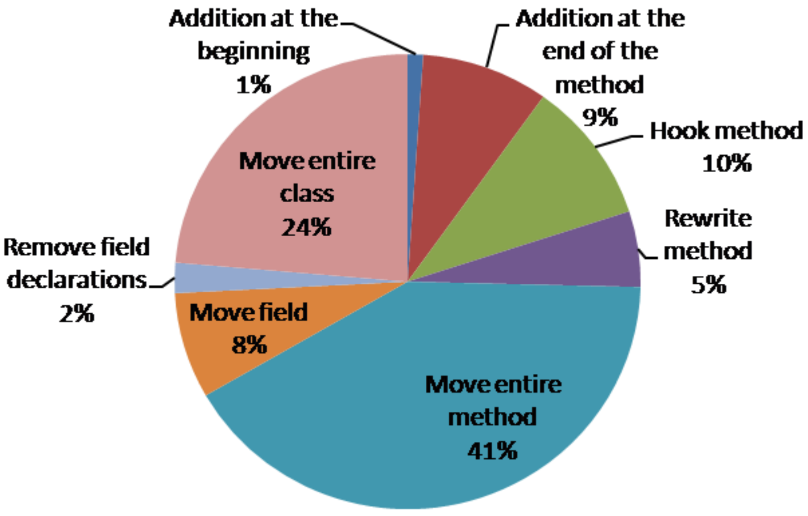
VOD Player

- 3.6 KLOC, 15 features, 42 classes



Gantt Chart

- 41 KLOC, 16 features, 43 packages



Making the connection with SBSE …

Safe Composition → Detect Inconsistencies

Feature Oriented Refactoring → Fix Inconsistencies

Open Question:

Can we leverage work from refactoring + SBSE?

CHALLENGE 4.
COPE WITH GROWING PAINS

Big Picture – Two Imporant Issues

- Evolution scenarios
  - Features are added, deleted, or their relationships changed in a feature model
  - Realizing artifacts change – feature renovation
  - Challenge:
    - How to cope with the co-evolution of variability and its realization?
- Maintenance activities
  - Software Testing

Relevant References (1)

- E. N. Haslinger, R. E. Lopez-Herrejon, and A. Egyed, "On extracting feature models from sets of valid feature combinations", FASE 2013.
- R. E. Lopez-Herrejon, J. F. Chicano, J. Ferrer, A. Egyed, and E. Alba, "Multi-objective optimal test suite computation for software product line pairwise testing", ICSM 2013
- L. Linsbauer, R. E. Lopez-Herrejon, and A. Egyed, "Recovering traceability between features and code in product variants", SPLC 2013.
- R. E. Lopez-Herrejon and A. Egyed, "Towards interactive visualization support for pair-wise testing software product lines", VISSOFT 2013
- R. E. Lopez-Herrejon, J. A. Galindo, D. Benavides, S. Segura, and A. Egyed, "Reverse engineering feature models with evolutionary algorithms: An exploratory study", SSBSE 2012.
- E. N. Haslinger, R. E. Lopez-Herrejon, and A. Egyed, "Reverse engineering feature models from programs' feature sets", WCRE 2011.
- R. E. Lopez-Herrejon, L. Montalvillo-Mendizabal, and A. Egyed, "From requirements to features: An exploratory study of feature-oriented refactoring", SPLC 2011.
- M. Alferez, R. E. Lopez-Herrejon, A. Moreira, V. Amaral, and A. Egyed, "Supporting consistency checking between features and software product line use scenarios", in ICSR 2011.
- R. E. Lopez-Herrejon, A. Egyed, "Detecting inconsistencies in multi-view models with variability", ECMFA 2010.

Relevant References (2)

- A. Eiben and J. Smith, Introduction to Evolutionary Computing. Springer Verlag, 2003
- M. Harman, P. McMinn, J. T. de Souza, and S. Yoo, Search Based Software Engineering: Techniques, Taxonomy, Tutorial. LASER Summer School 2010.
- M. Harman, S. Afshin Mansouri, Y. Zhang. Search-Based Software Engineering: Trends, Techniques and Applications. ACM Computing Surveys 2012.
- Roberto E. Lopez-Herrejon, Javier Ferrer, Francisco Chicano, Alexander Egyed, Enrique Alba. Comparative Analysis of Classical Multi-Objective Evolutionary Algorithms and Seeding Strategies for Pairwise Testing of Software Product Lines. Accepted at IEEE Congress on Evolutionary Computation (IEEE CEC), Beijing, China, July 2014.
- Roberto E. Lopez-Herrejon, Javier Ferrer, Francisco Chicano, Evelyn Nicole Haslinger, Alexander Egyed, Enrique Alba. A Parallel Evolutionary Algorithm for Prioritized Pairwse Testing of Software Product Lines. GECCO 2014.