



# Clean Your Variable Code with FeatureIDE

Thomas Thüm, Thomas Leich, Sebastian Krieter

September 20, 2016 — SPLC'16 in Beijing, China

# Part I

## Introduction round



---

## Who we are

---



Dr. Thomas Thüm @ TU Braunschweig

... started 2006 as developer

... coordinates FeatureIDE development + research



M.Sc. Sebastian Krieter @ University of Magdeburg

... started 2012 as developer

... an expert on FeatureIDE's architecture



Prof. Dr. Thomas Leich @ Metop GmbH

... initiated FeatureIDE project in 2004

... coordinates FeatureIDE consulting

---

# Introduce yourself

---

Keep it short (3 sentences):

1. What is your name and affiliation?
2. What is your expertise?
3. What are you expecting to learn in this tutorial?



---

## What is your background?

---

Are you involved in professional software development?

---

## What is your background?

---

Are you involved in professional software development?

Are you performing research?

---

## What is your background?

---

Are you involved in professional software development?

Are you performing research?

Are you teaching?



---

## What is your background?

---

Are you involved in professional software development?

Are you performing research?

Are you teaching?

Do you know feature models?



---

## What is your background?

---

Are you involved in professional software development?

Are you performing research?

Are you teaching?

Do you know feature models?

Do you know preprocessors?



---

# What is your background?

---

Are you involved in professional software development?

Are you performing research?

Are you teaching?

Do you know feature models?

Do you know preprocessors?

Have you experienced problems with preprocessors?

---

# Disclaimer: This is not a lecture!

---



- ▶ Mixture of presentations and hands-on sessions
- ▶ Notebook with Windows, Linux, or MacOS available?
- ▶ Feel free to ask at any time

## Part II

Why is product-line maintenance hard?



# Preprocessing principle is easy

BestLap

```
private void stopSound(int soundIndex) {  
    this.sounds[soundIndex].stop();  
  
    this.sounds[soundIndex].deallocate();  
  
    this.sounds[soundIndex].setMediaTime(0);  
}
```

# Preprocessing principle is easy

BestLap

```
private void stopSound(int soundIndex) {  
    this.sounds[soundIndex].stop();  
    //#ifdef (mmapi)  
    //#ifdef (sound_prefetch)  
    this.sounds[soundIndex].deallocate();  
    //#endif  
    //#ifdef (!media_time)  
    this.sounds[soundIndex].setMediaTime(0);  
    //#endif  
    //#endif  
}
```

# Preprocessing principle is easy

BestLap

```
private void stopSound(int soundIndex) {  
    this.sounds[soundIndex].stop();  
    //#ifdef (mmapi)  
    //#ifdef (sound_prefetch)  
    this.sounds[soundIndex].deallocate();  
    //#endif  
    //#ifdef (!media_time)  
    this.sounds[soundIndex].setMediaTime(0);  
    //#endif  
    //#endif  
}
```

```
private void stopSound(int soundIndex) {  
    this.sounds[soundIndex].stop();  
}
```

# Preprocessing principle is easy

BestLap

```
private void stopSound(int soundIndex) {  
    this.sounds[soundIndex].stop();  
    //#ifdef (mmapi)  
    //#ifdef (sound_prefetch)  
    this.sounds[soundIndex].deallocate();  
    //#endif  
    //#ifdef (!media_time)  
    this.sounds[soundIndex].setMediaTime(0);  
    //#endif  
    //#endif  
}
```

```
private void stopSound(int soundIndex) {  
    this.sounds[soundIndex].stop();  
    this.sounds[soundIndex].deallocate();  
}
```

# Preprocessing principle is easy

BestLap

```
private void stopSound(int soundIndex) {  
    this.sounds[soundIndex].stop();  
    //#ifdef (mmapi)  
    //#ifdef (sound_prefetch)  
    this.sounds[soundIndex].deallocate();  
    //#endif  
    //#ifdef (!media_time)  
    this.sounds[soundIndex].setMediaTime(0);  
    //#endif  
    //#endif  
}
```

```
private void stopSound(int soundIndex) {  
    this.sounds[soundIndex].stop();  
}
```

```
private void stopSound(int soundIndex) {  
    this.sounds[soundIndex].stop();  
}
```

```
private void stopSound(int soundIndex) {  
    this.sounds[soundIndex].stop();  
  
    this.sounds[soundIndex].setMediaTime(0);  
}
```

# Preprocessing principle is easy

BestLap

```
private void stopSound(int soundIndex) {  
    this.sounds[soundIndex].stop();  
    //#ifdef (mmapi)  
    //#ifdef (sound_prefetch)  
    this.sounds[soundIndex].deallocate();  
    //#endif  
    //#ifdef (!media_time)  
    this.sounds[soundIndex].setMediaTime(0);  
    //#endif  
    //#endif  
}
```

```
private void stopSound(int soundIndex) {  
    this.sounds[soundIndex].stop();  
    this.sounds[soundIndex].deallocate();  
    this.sounds[soundIndex].setMediaTime(0);  
}
```

# Preprocessing is widely adopted

# An Analysis of the Variability in Forty Preprocessor-Based Software Product Lines



# Preprocessor code is complex

## An Analysis of the Variability in Forty Preprocessor-Based Software Product Lines

Jörg Liebig, Sven Apel,  
and Christian Lämmel  
(jliebig.apel.laemmel@fim.uni-passau.de)

Christian Kästner and Michael Schulze  
University of Magdeburg  
(kaestner.schulze@ovgu.de)

**ABSTRACT**  
Over 30 years ago, the preprocessor *cpp* was developed by Dennis Ritchie as part of the C programming language for macro expansion. Despite its reusability and low abstraction level, the preprocessor is still widely used in generic software projects to implement variability. However, not much is known about how *cpp* is employed in large-scale systems. To address this gap, we analyzed forty open-source software projects written in C, C++, Java, and C# to study the influence of *cpp* on program size and source variability. How complex are macros used via *cpp*? How variability metrics change as whole files are preprocessed? At which level of preprocessing are performance overheads applied? Which types of code are generated?

pre  
cpp  
c  
c++  
java  
c#  
macro  
language

Liebig et al. @ ICSE'10

### Categories and Subject Descriptors

D.2.3 Software Engineering: Coding Tools and Techniques; D.2.4 Software Engineering: Methodologies; D.2.4.1 Programming Languages: Preprocessor—*Preprocessors*

### General Terms

Empirical Study

### Keywords

Software Product Lines, C Preprocessor

### 1. INTRODUCTION

The C preprocessor (*cpp*) is a powerful tool for implementing variable software. It has been developed to reduce C-type

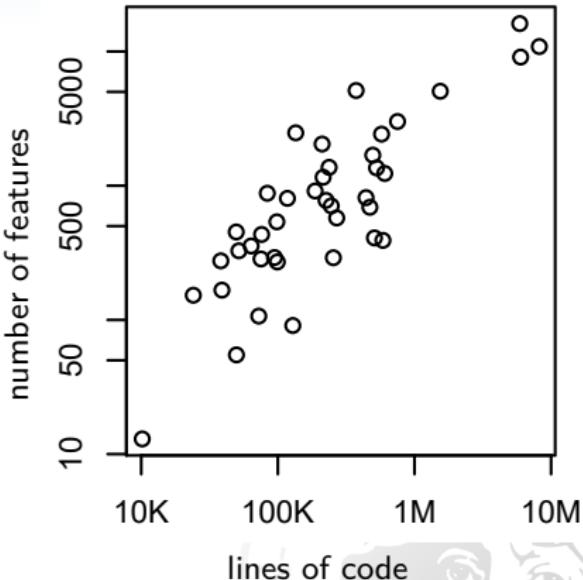
lengthy macroexpansions and code repetitions and is commonly used to merge files, switch statements, and code fragments (e.g., in configuration files) [21]. As the *cpp* is line-based, it can be used with many different programming languages, such as C, C++, C#, or Java or C#. In the past, it has been observed that *cpp* is often used to implement variability [22], (1) to support reuse of syntactic and semantic errors during the generation of software products [22], (2) under specific circumstances, such as in Java or C#, (3) a decrease in maintainability and in ability to review [22].

The preprocessor is also increasingly used in a major field of software product line engineering. A software product line (SPL) is a set of software-intensive systems sharing a

common infrastructure to support variability [23]. It is well-known that the preprocessor is a key component of the SPL and is used quite frequently in the implementation of SPLs [24, 25, 26, 27]. We believe this assumption to be true because the preprocessor is a well-known technique for reuse and SPLs started in prior work with a related technique, namely, the reuse of code fragments [28].

How does program size influence variability of SPLs? How does variability influence the complexity of programs? At which level of abstraction are macros used? At which level of preprocessing are performance overheads applied? Which types of optimization occur? We answer these questions by analyzing the 40 open-source projects with *cpp* help to judge whether *cpp* usage causes performance overheads, influences variability, and reduces maintainability and ability to review. An analysis of the 40 projects shows that *cpp* is used in many ways and benefits of integrating to other, more well-defined implementation techniques for SPL engineering, such as aspects [27, 28].

To answer the questions above, we analyzed forty open-source projects with *cpp* help. The projects have between 10k and 10M lines of code. They cover different domains including mobile devices, embedded systems, and enterprise systems. The projects are a set of artifacts that allow us to infer and classify *cpp* usage patterns and to study the patterns to examine SPL characteristics. The results show that *cpp* is used in a large variety to implement and control variability.



# Preprocessor code is complex

An Analysis of the Variability in Forty Preprocessor-Based Software Product Lines

Jörg Liebig, Sven Apel,  
and Christian Lengauer  
University of Magdeburg  
(jliebig.apel.lengauer@fmi.uni-magdeburg.de)

Christian Kästner and Michael Schulze  
University of Magdeburg  
(ckaechner.mschulze@fmi.uni-magdeburg.de)

**ABSTRACT**  
Over 20 years ago, the preprocessor (`#pre`) was developed to reduce the complexity of programs [7] by separating software generating capabilities. Since its re-invention and low-level extension in C, the preprocessor has become a standard component of day-to-day software projects to implement variable software. However, not much is known about how easy it is to implement variability in C. To help answer this question, we analyzed forty open-source software projects written in C, looking at the relationship between the number of features and the program size (in lines of variable code). How complex are software systems with regard to variability? How many levels of generality are necessary to implement variability? Which types of code reuse are possible, such as source code artifacts [3, 11, 22]? Is it possible to reuse code artifacts from one feature to another? Can the preprocessor be used quite frequently in the implementation of SPLs [3, 12, 14, 22]? We believe this analysis to be true because the preprocessor is a well-known technique for implementing product lines and SPLs started in prior work with a relatively small number of features [12].

How does program size influence variability of SPLs? How does the number of features influence variability of the variability mechanisms? At which level of generality are mechanisms applied? Which types of reuse are possible? What are the reasons for the reuse of the same piece of code involved with one feature to judge whether other programs cause similar variability? What are the benefits of using more abstract and well-defined implementation mechanisms for variability engineering, such as aspects [27, 28] or various forms of templates?

To answer these questions above, we analyzed forty open-source software projects (with a total of 10,000+ lines of code) that have different domains and reuse mechanisms. The results show that the preprocessor is a good tool to implement variability in C. We propose a set of metrics that allow us to infer and classify variability engineering in C. We also analyze the C# implementation concepts. Our analysis reveals that C# is used to a large extent to implement and control variability.

**Categories and Subject Descriptors**  
D.2.2 [Software Engineering]: Tools and Techniques; D.2.3 [Software Engineering]: Metrics; D.2.4 [Program Processing Languages]: Preprocessor—Preprocessors

**General Terms**  
Empirical Study

**Keywords**  
Software Product Lines, C Preprocessor

**1. INTRODUCTION**  
The C preprocessor (`#pre`) is a popular tool for implementing variable software. It has been developed to reduce C program size [7].

Promises to make digital or hard copies of all or part of this work and permission to do so, subject to specific terms and conditions, may be granted by the copyright holders for people to read, copy, print, use, download and store digital copies of the material in whole or in part for personal research and educational, professional and not-for-profit purposes provided that:

• The full name of the copyright holders is given,

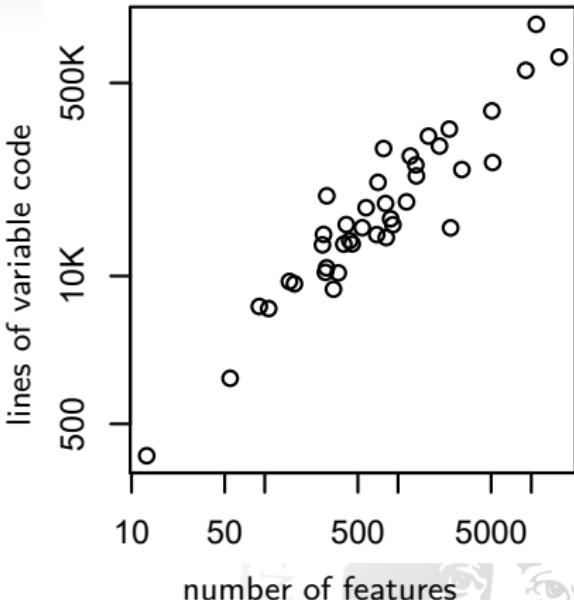
• The full name of the journal is given,

• A copy of the article is not made available for sale in any format or medium without the formal permission of the copyright holders,

• The full bibliographic reference is made to this article in any republication,

• A link is made to the journal's home page (or a link is made to the article using a digital object identifier).

ISSN: 0898-1562 © May 2010, IEEE Trans. Softw. Eng., Vol. 36, No. 5, pp. 593–606, DOI 10.1109/TSE.2010.2010030



---

# The #ifdef hell

---

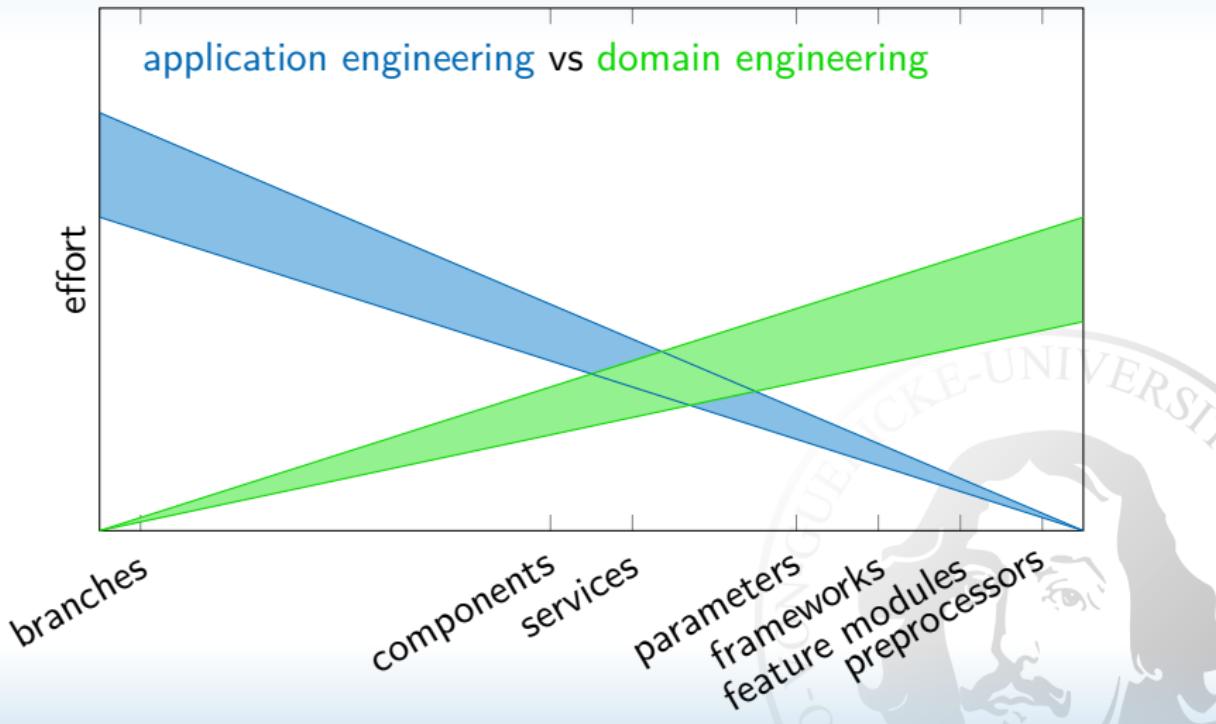


## Part III

# Feature-oriented software development



# Application engineering vs domain engineering



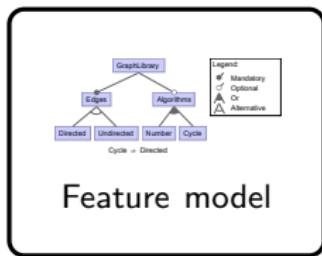
---

# Heterogeneous software artifacts

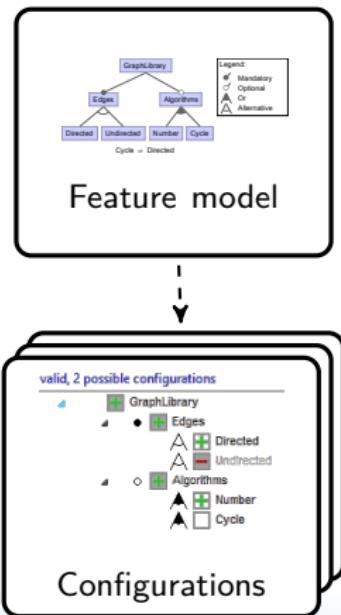
---

- ▶ source code (here Java)
- ▶ binaries
- ▶ tests (here JUnit)
- ▶ specification
- ▶ documentation

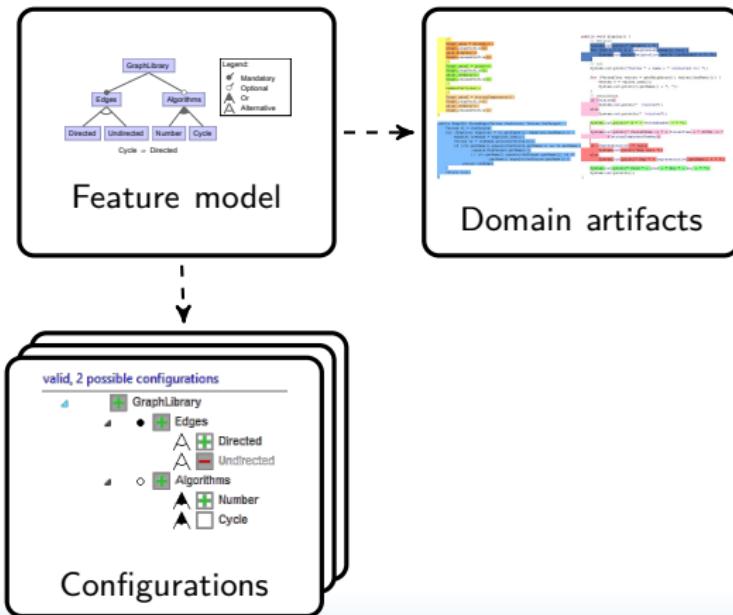
# Feature-oriented software development



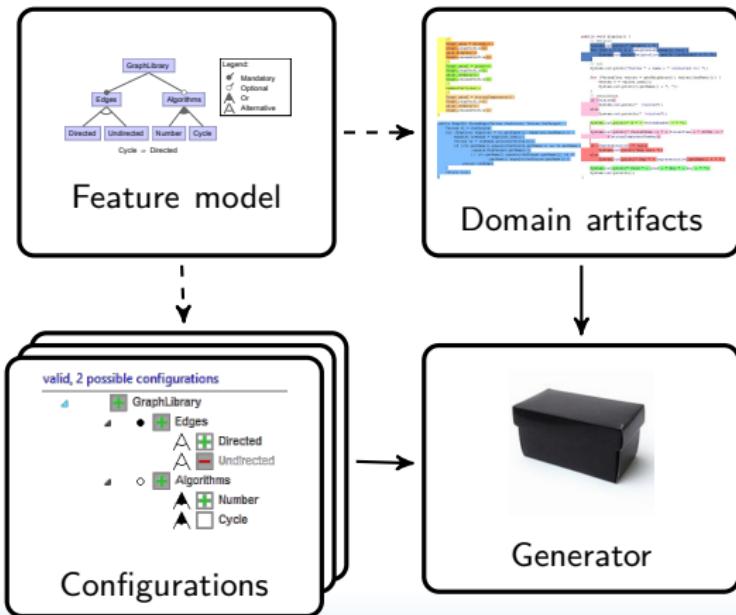
# Feature-oriented software development



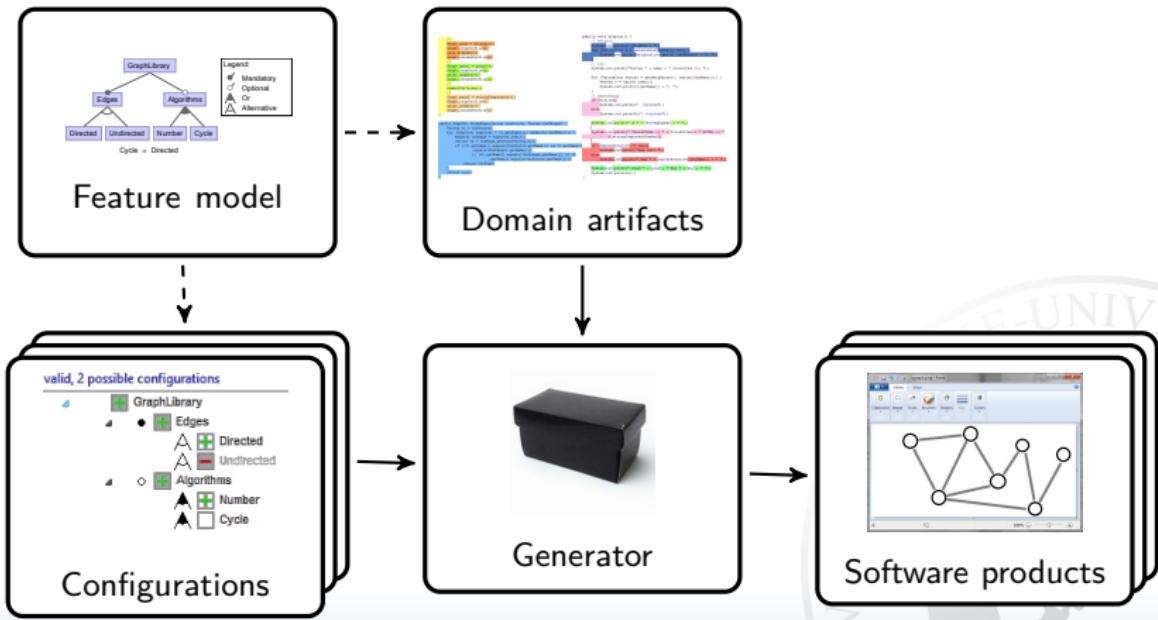
# Feature-oriented software development



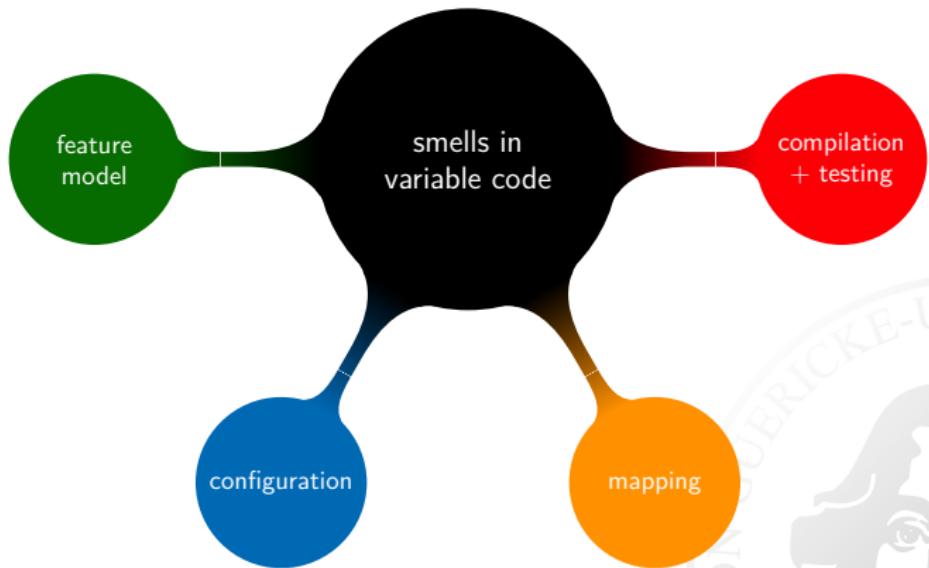
# Feature-oriented software development



# Feature-oriented software development



# Tutorial overview

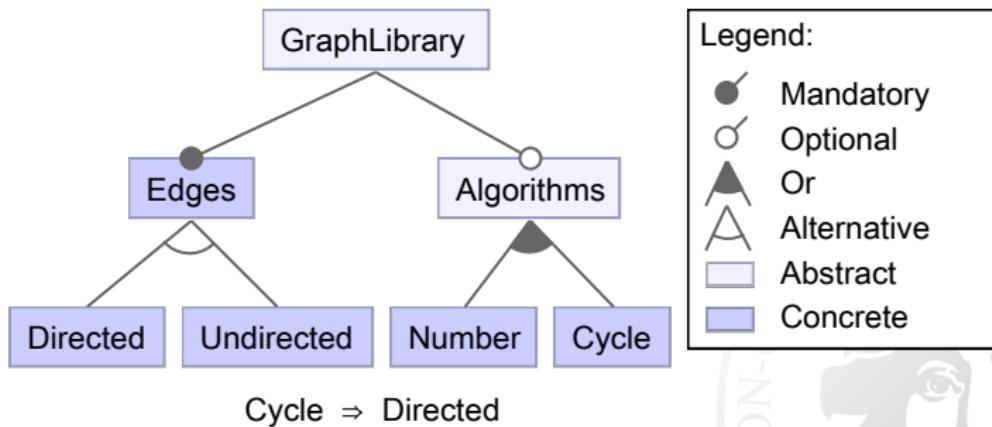


## Hands-on I: run different elevator configurations

1. Install Java 1.7 or higher (if not yet installed)
2. Unzip FeatureIDE for your OS
3. Start FeatureIDE by executing Eclipse
4. Open proposed workspace (i.e., `../workspace`)
5. Run and try the elevator: right click on folder `src` > Run As > Java Application
6. Change configuration: right click on `*.config` in folder `configs` > FeatureIDE > Set as current configuration
7. Repeat 5. and 6. several times

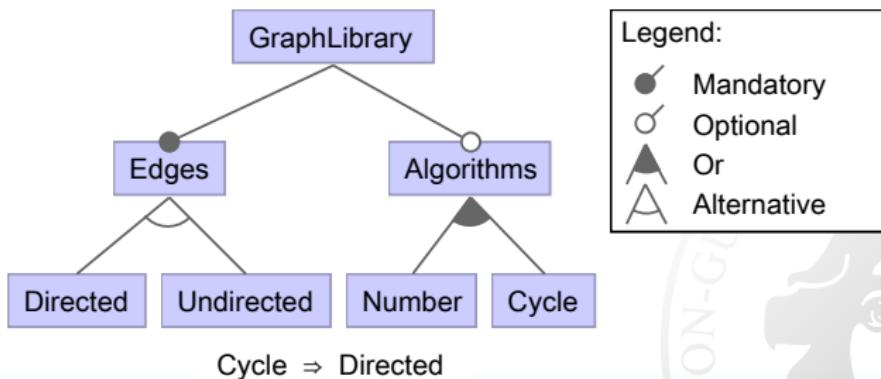
## Part IV

# Smells in feature modeling



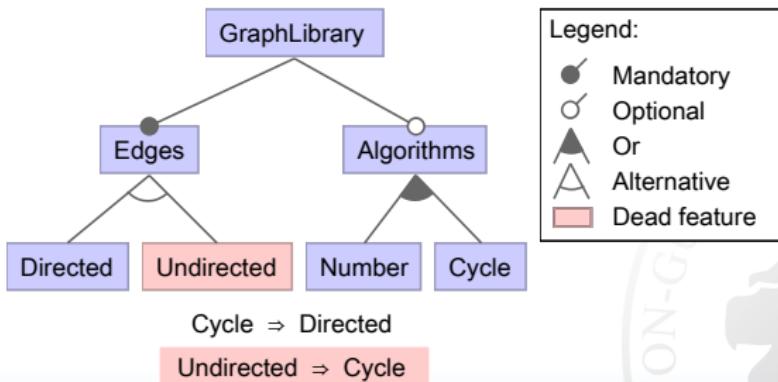
# Feature model

- ▶ Specifies valid combinations of features
- ▶ Graphically represented by a feature diagram
- ▶ Defines the scope/domain of a software product line



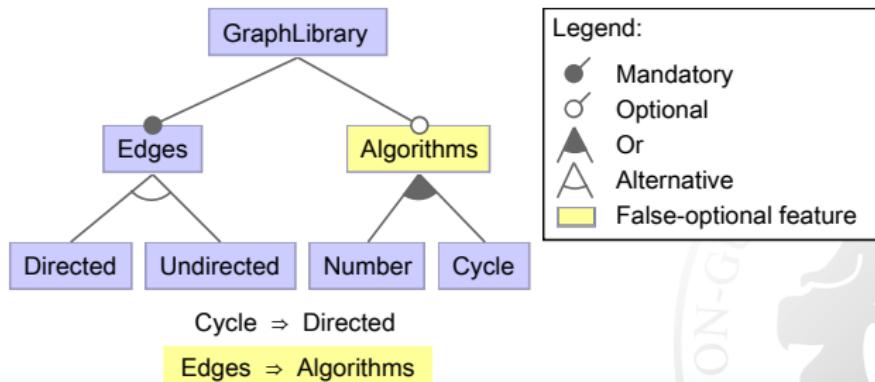
# Dead features

- ▶ There is no valid configuration containing that feature
- ▶ Frustrating during configuration, may result in dead code
- ▶ Fix: remove feature (refactoring) or remove/edit constraints



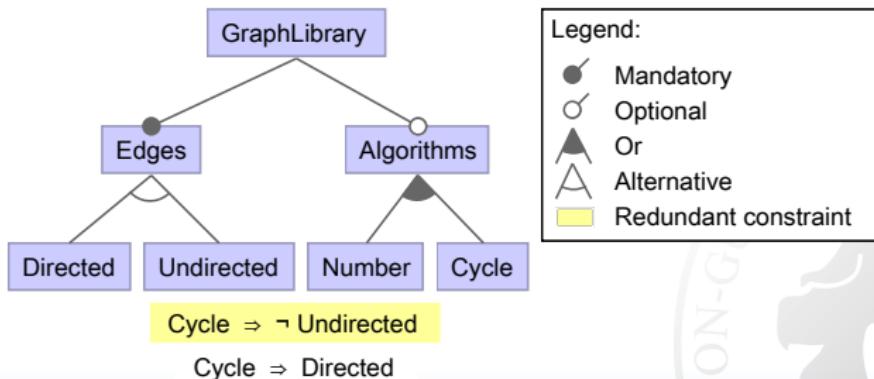
# False-optional features

- ▶ Feature is modeled as optional, but is effectively mandatory
- ▶ Suggest more variability than actually available
- ▶ Fix: make the feature mandatory to avoid confusion



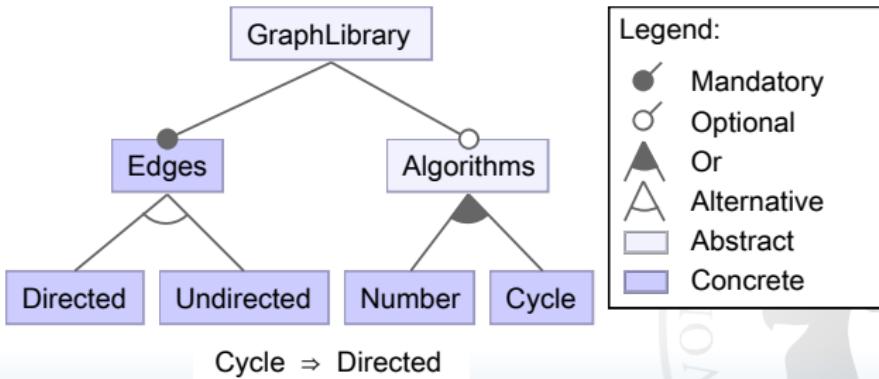
# Redundant constraints

- ▶ Constraint does not influence valid configurations
- ▶ Model is more verbose than necessary
- ▶ Fix: remove it (refactoring) or edit/remove other constraints

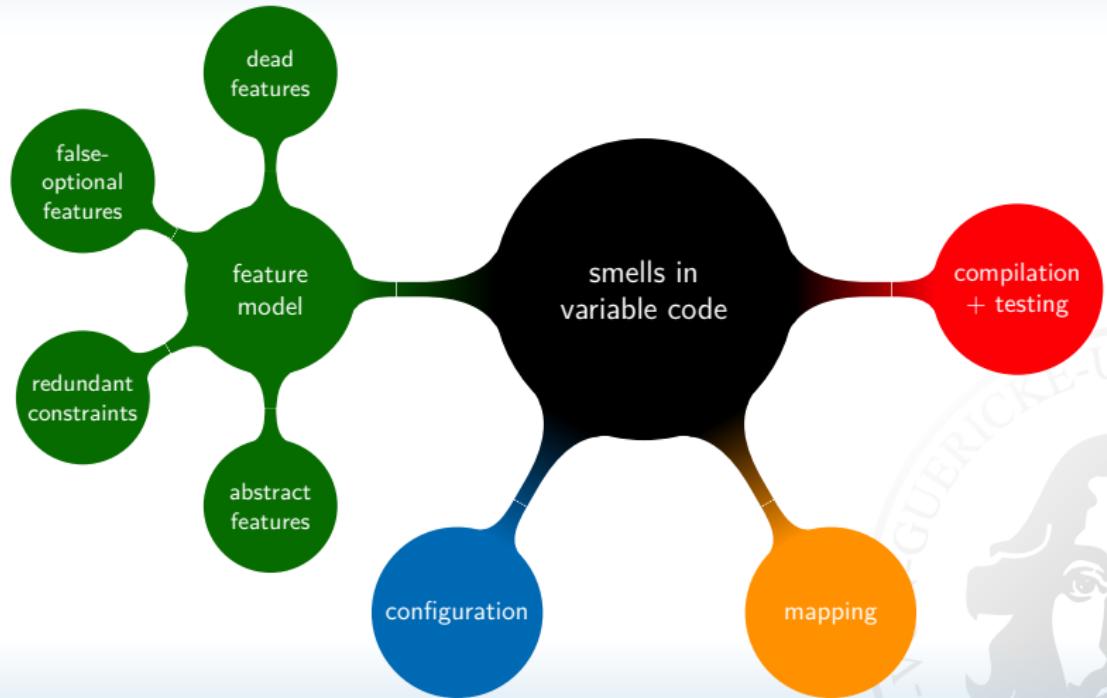


# Abstract and concrete features

- ▶ Feature is called concrete if it is mapped to domain artifacts and abstract otherwise
- ▶ Helps to identify mismatch between modeled and actually implemented variability, used for some analyses (sampling)



# Tutorial overview



---

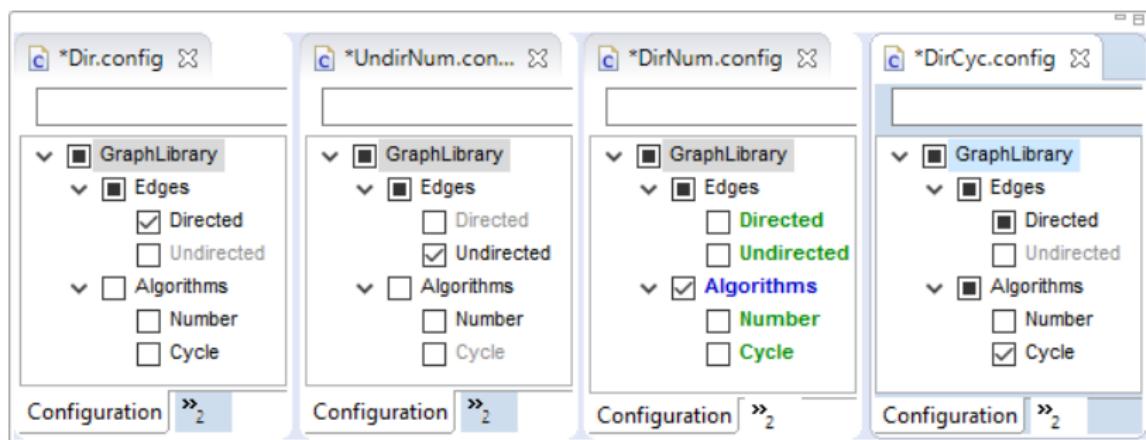
## Hands-on II: fix smells in the feature model

---

1. (Continue with previous project Elevator01)
2. Open file `model.xml`
3. Turn on automated analyses: right click on a feature > Set Calculations > Automated Calculations
4. Fix dead feature *Overloaded* by removing the constraint mentioned in the feature's tool tip
5. Fix false-optional feature *Permission* by making it mandatory
6. Remove redundant constraint (caused by last edit)
7. Problems view tells you which features are not referenced in the implementation, mark them as abstract using right click on each feature and save the feature model

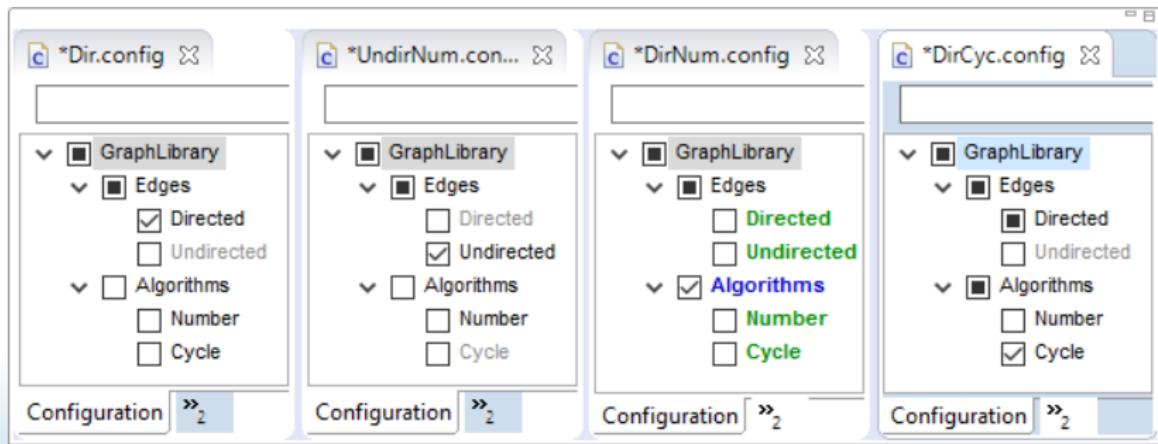
## Part V

# Smells in configurations



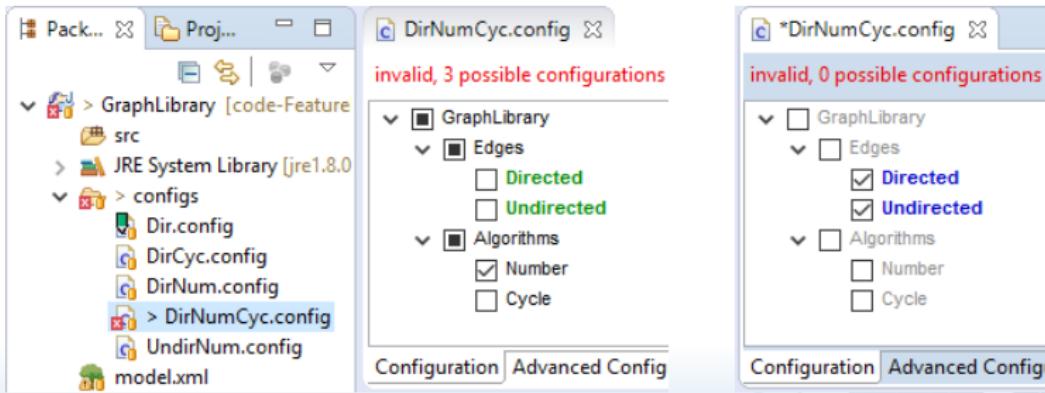
# Creating configurations

- ▶ Possible to manually select features
- ▶ Automatic selections and deselections are computed
- ▶ Validity with respect to feature model is computed
- ▶ Green/blue indicate that selection/deselection will lead to valid configuration



# Invalid configurations

- ▶ Selection of features not allowed in feature model
- ▶ Saved invalid or invalid by change in feature model
- ▶ Computed whenever feature model or configurations change
- ▶ Missing selections (green), conflicting selections (blue)



# Unused features

- Unused feature is not selected in any configuration
- Could indicate unnecessary or untested code

The screenshot shows the FeatureIDE interface with four configuration windows side-by-side:

- Dir.config**: Contains **GraphLibrary** (selected), **Edges** (selected), **Directed** (selected), **Undirected** (unchecked), **Algorithms** (unchecked), **Number** (unchecked), and **Cycle** (unchecked).
- DirCyc.config**: Contains **GraphLibrary** (selected), **Edges** (selected), **Directed** (unchecked), **Undirected** (unchecked), **Algorithms** (unchecked), **Number** (unchecked), and **Cycle** (selected).
- DirNum.config**: Contains **GraphLibrary** (selected), **Edges** (selected), **Directed** (unchecked), **Undirected** (unchecked), **Algorithms** (unchecked), **Number** (selected), and **Cycle** (unchecked).
- DirNumCyc.config**: Contains **GraphLibrary** (selected), **Edges** (selected), **Directed** (unchecked), **Undirected** (unchecked), **Algorithms** (selected), **Number** (selected), and **Cycle** (selected).

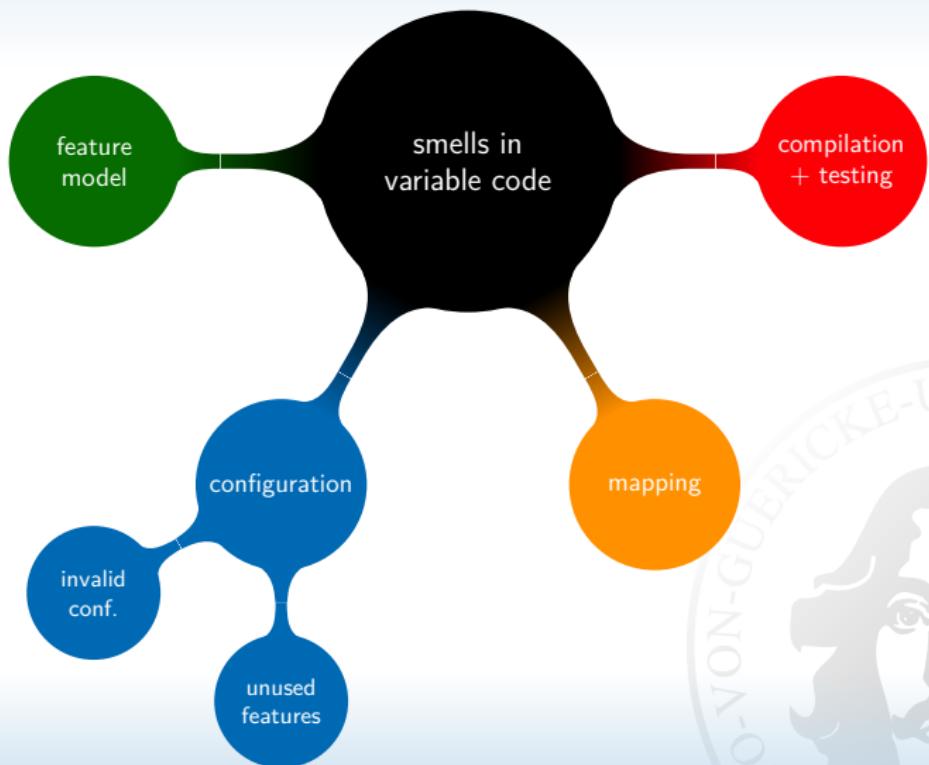
Below the configurations is a toolbar with tabs: **Collaboration Diagram**, **Feature Model Edits**, **FeatureIDE Statistics**, **Problems**, and **Console**.

A message bar at the bottom left says "2 items".

A detailed view table follows:

Description	Resource	Path
Infos (2 items)		
False optional: 1 feature is optional but used in all configurations: Directed	configs	/GraphLibrary
Unused: 1 feature is not used: Undirected	configs	/GraphLibrary

# Tutorial overview



---

## Hands-on III: fix smells of configurations

---

1. (Continue with previous project or import Elevator02)
2. Create a new configuration called Professional, select the features FIFO, DirectedCall, FloorPermission, save and close the editor
3. Open the feature model, change the group type below feature Mode to an alternative, and save the model
4. Open the invalid configuration Professional, deselect feature FIFO, and save it
5. Problem view indicates unused features, use right click > Quick Fix to create a configuration with feature Sabbath, refresh folder configs, and rename configuration to Starter

## Part VI

# Smells in feature-to-code mapping

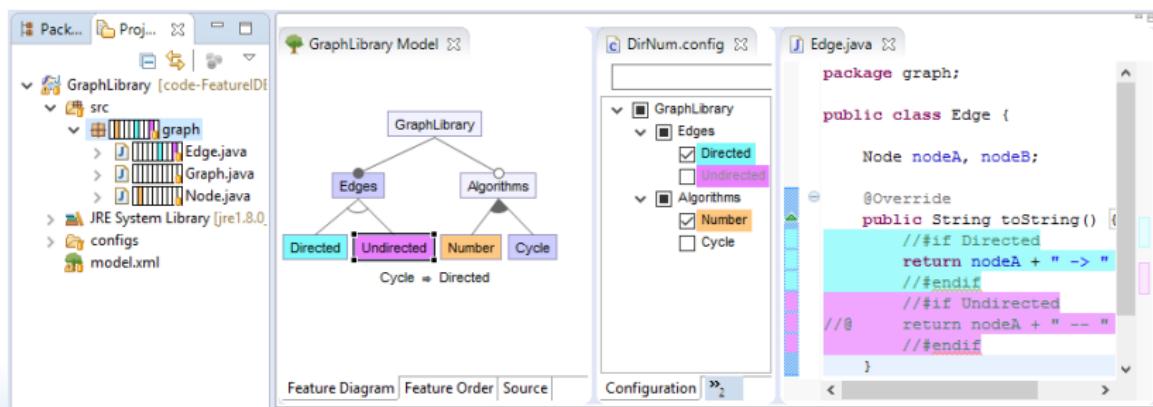
The screenshot displays a software interface for managing a 'GraphLibrary Model'. It consists of three main panels:

- GraphLibrary Model:** A Feature Diagram showing the structure of the library. The root node is 'GraphLibrary', which branches into 'Edges' and 'Algorithms'. 'Edges' further branches into 'DirectedEdges' (selected) and 'Undirected'. 'Algorithms' branches into 'Number' (selected) and 'Cycle'. A note at the bottom states "Cycle => DirectedEdges".
- DirNum.config:** A configuration view showing the selected features: 'DirectedEdges' and 'Number'.
- Edge.java:** A code editor displaying the implementation of the 'Edge' class. The code uses conditionals to handle different edge types based on the selected features.

```
public class Edge {  
    Node nodeA, nodeB;  
  
    @Override  
    public String toString() {  
        //#if DirectedEdges  
        return nodeA + " -> " + nodeB;  
        //#endif  
        //#if Undirected  
        return nodeA + " -- " + nodeB;  
        //#endif  
    }  
}
```

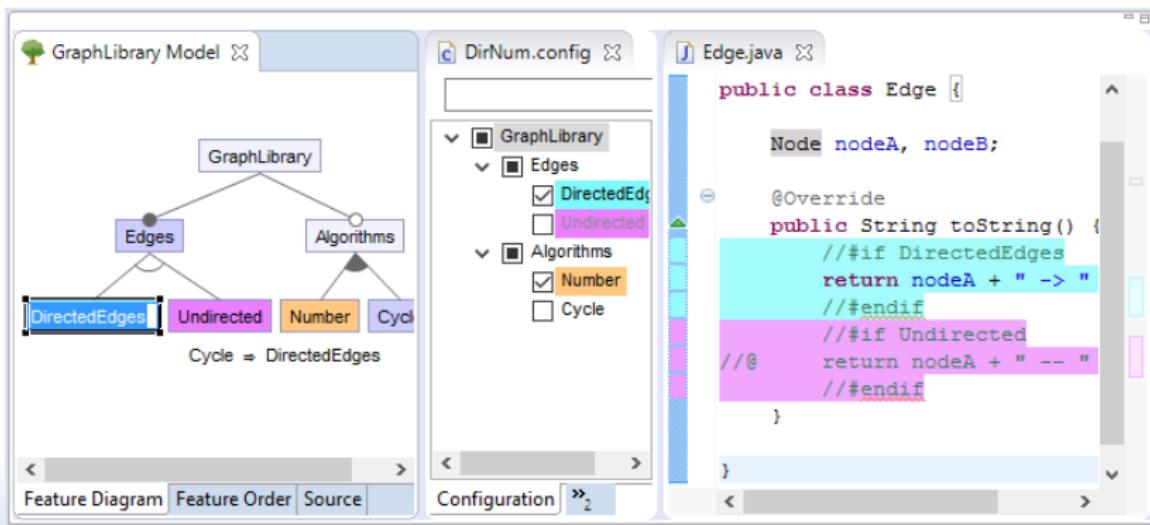
# Feature traceability problem

- ▶ Feature traceability is hard to establish for large projects
- ▶ Product-line maintenance typically requires to trace features
- ▶ Manually assigned colors help to locate features in feature model, configurations, packages, and source code



# Misleading feature names

- ▶ During evolution, features change what they used to be
- ▶ Misleading names create confusion and obfuscate artifacts
- ▶ FeatureIDE renames features automatically in all artifacts



# Undefined feature references

- ▶ Features referenced in source code that do not exist in the feature model are always assumed to be false
- ▶ May lead to code being maintained but never used
- ▶ Content assist and warnings support developers to avoid them

The screenshot shows the FeatureIDE interface. On the left, the 'GraphLibrary Model' feature diagram is displayed, showing a tree structure with 'GraphLibrary' at the root, which branches into 'Edges' (solid circle) and 'Algorithms' (open circle). 'Edges' further branches into 'Directed' and 'Undirected'. 'Algorithms' has a dependency arrow pointing to 'Directed'. Below the tree, there are tabs for 'Feature Diagram', 'Feature Order', and 'Source'. The 'Source' tab is selected, showing the Java code for 'Edge.java':

```
@Override  
public String toString() {  
    //if Directet  
    return nodeA + " -> "  
    //endif  
    //if  
    return  
    //endif  
}
```

A red squiggly underline is under the word 'Directet', indicating it is undefined. On the right, a content assist dropdown shows suggestions for undefined features: 'Cycle', 'Directed', 'Edges', 'Number', and 'Undirected'. At the bottom, the 'Problems' view shows one warning: 'Antenna: Directet is not defined in the feature model and, thus, always assumed to be false'.

# Dead code blocks

- ▶ Even if all features are defined, some blocks are never selected
- ▶ Contradictions typically arise due to nesting or feature model
- ▶ Similar to unreachable code in a programming language

The screenshot shows the FeatureIDE interface with two main panes. On the left is the 'GraphLibrary Model' feature diagram, which is a tree structure starting from 'GraphLibrary'. It branches into 'Edges' (with 'Directed' and 'Undirected' children) and 'Algorithms' (with 'Number' and 'Cycle' children). A note below the tree states 'Cycle => Directed'. Below the tree are tabs for 'Feature Diagram', 'Feature Order', and 'Source'. On the right is the 'Edge.java' code editor window. The code is:

```
@Override
public String toString() {
    //##if Directed && !Directed
    return nodeA + " -> " + nodeB;
    //##endif
    //##if Undirected
    return nodeA + " -- " + nodeB;
    //##endif
}
```

Below the code editor, the 'Problems' view shows '0 errors, 1 warning, 0 others'. The 'Warnings' section contains one item: 'Antenna: This expression is a contradiction and causes a dead code block.' The status bar at the bottom right shows 'Edge.java /G'.

# Superfluous preprocessor annotations

- ▶ Depending on nesting and feature model, some blocks may always be enabled making the preprocessor annotations useless
- ▶ Make source code unnecessarily complex and can be removed

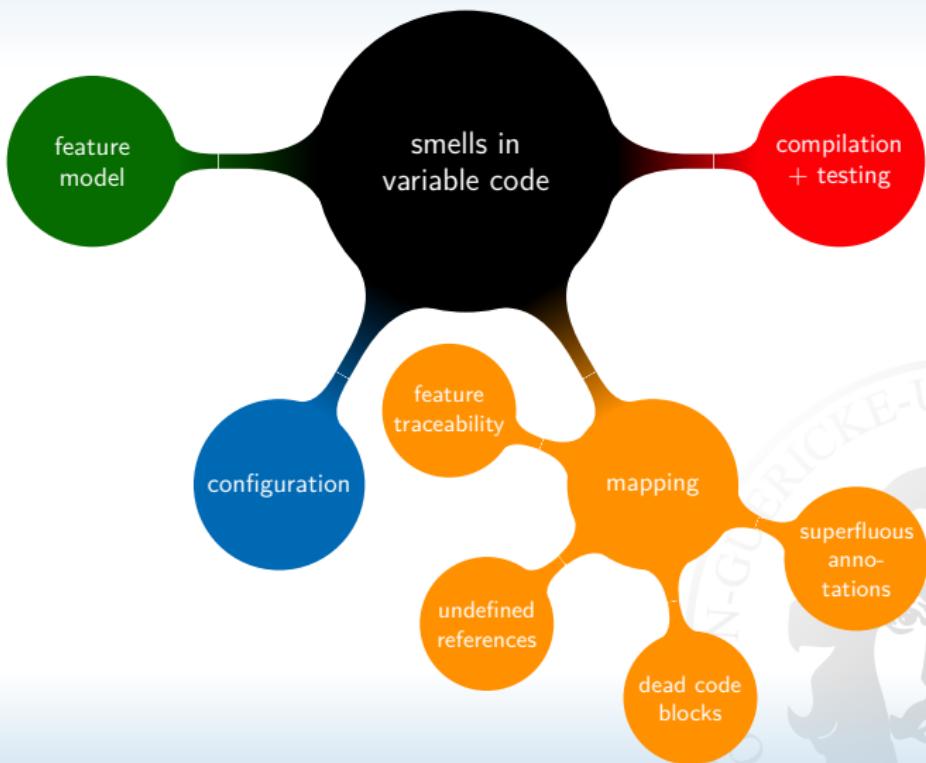
The screenshot shows the FeatureIDE interface. In the center is a code editor window titled "Edge.java" containing the following Java code:

```
//#if Directed
return nodeA
//#if Directed
+ " -> "
//#endif
+ nodeB;
//#endif
```

A yellow warning icon is visible on the left margin next to the first "#if" directive. Below the code editor, the status bar displays "0 errors, 1 warning, 0 others". In the bottom right corner of the status bar, there is a "Warnings" section with the following message:

⚠ Warnings (1 item)  
⚠ Antenna: This expression is a tautology and causes a superfluous code block.

# Tutorial overview



---

## Hands-on IV: fix smells in mapping

---

1. (Continue with previous project or import Elevator03)
2. Trace the misspelled feature ShortestPath in source code
  - 2.1 Activate color scheme by right click on project in package explorer > FeatureIDE > Color Scheme > default
  - 2.2 Open the feature model and assign a color to feature ShortestPath
  - 2.3 Identify source files containing that feature in project (not package) explorer and open one of them
3. Rename the feature in feature model and save it, check whether renaming is applied to configurations and source code
4. Use the problems view to locate and remove dead code blocks
5. Remove superfluous preprocessor statements (keep Java code)
6. Fix misspelled feature name DirectedCall in the source code

## Part VII

# Smells in compilation and testing



# Invisible compiler errors

- ▶ Compiler errors may exist only in some configurations
- ▶ Problem if found late in development process (i.e., in application engineering)
- ▶ Large effort to create and generate many configurations

The image shows two side-by-side Java code editors. On the left is the code for `Edge.java`, which contains a syntax error: the identifier `nodeB` is used before it is declared. On the right is the code for `Graph.java`, which contains a syntax error: the identifier `Graph` is used before it is declared. Both errors are underlined with red, indicating they are compiler errors.

```
//#if Edges
package graph;
public class Edge {
    Node nodeA, nodeB;
    @Override
    public String toString() {
        //#if Directed
        return nodeA + " -> " + nodeB;
        //#endif
    }
}
//#endif
```

```
package graph;

import java.util.List;

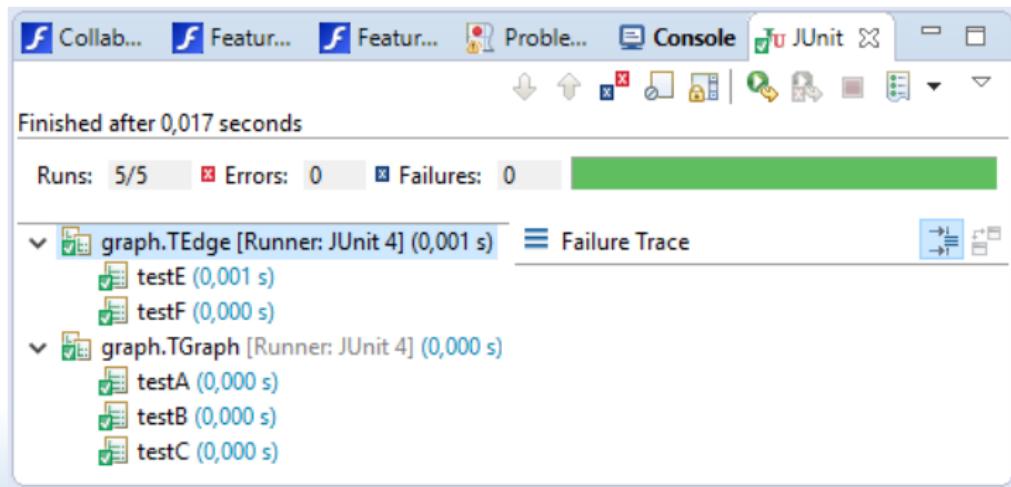
public class Graph {

    List<Node> nodes;

    List<Edge> edges;
}
```

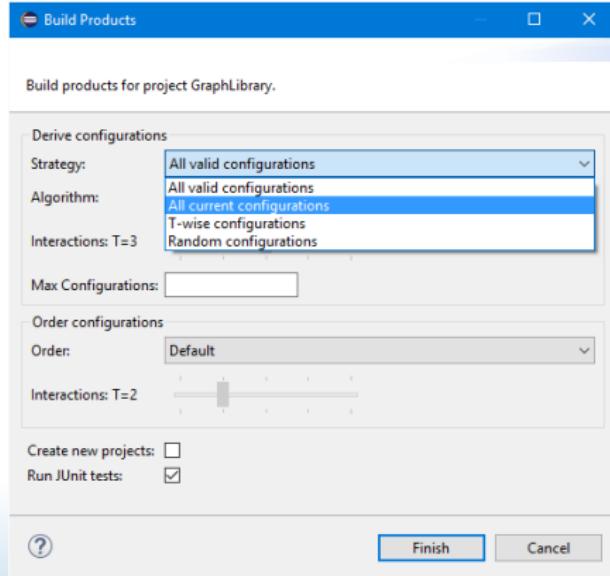
# Hidden test failures

- ▶ Each unit test may or may not fail for some configurations
- ▶ It is not enough if all tests pass in one configuration
- ▶ Running unit tests manually for each valid configuration and interpreting the results is laborious



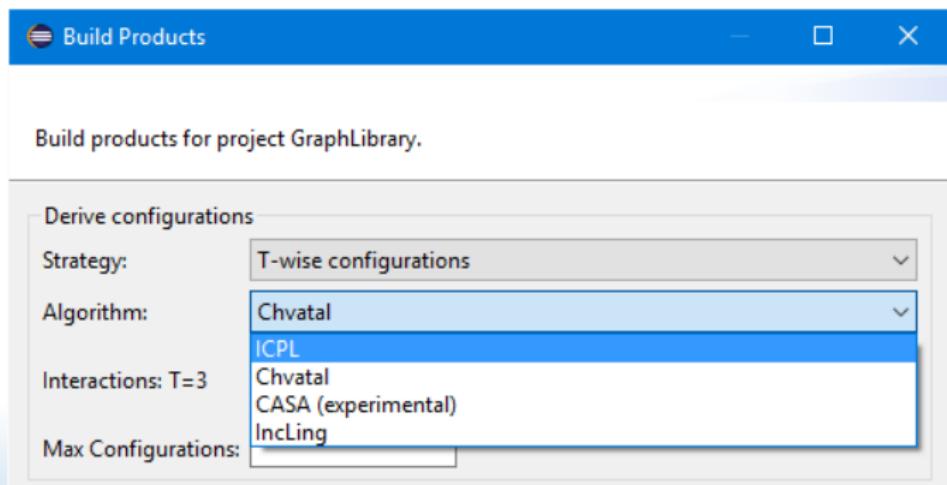
# Solution: product generation in batch mode

- ▶ FeatureIDE can compile and test configurations on demand
- ▶ Compiler errors are mapped back to original source code
- ▶ All unit tests are visualized in the JUnit view



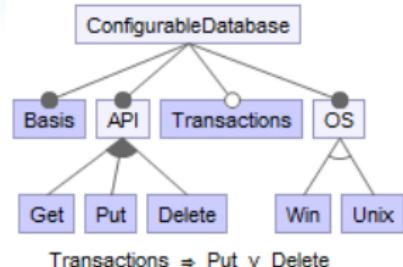
# T-wise sampling

- ▶ Compiling each product is typically not feasible
- ▶ Executing all tests for all configurations does not scale
- ▶ Most of those defects are due to an interaction of few features
- ▶ Configuration set covers interactions between up-to T features



## Example of pairwise sampling ( $T=2$ )

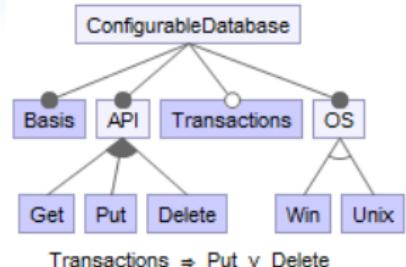
$G \wedge P$	$G \wedge \neg P$	$\neg G \wedge P$	$\neg G \wedge \neg P$
$G \wedge D$	$G \wedge \neg D$	$\neg G \wedge D$	$\neg G \wedge \neg D$
$G \wedge T$	$G \wedge \neg T$	$\neg G \wedge T$	$\neg G \wedge \neg T$
$G \wedge W$	$G \wedge \neg W$	$\neg G \wedge W$	$\neg G \wedge \neg W$
$G \wedge U$	$G \wedge \neg U$	$\neg G \wedge U$	$\neg G \wedge \neg U$
$P \wedge D$	$P \wedge \neg D$	$\neg P \wedge D$	$\neg P \wedge \neg D$
$P \wedge T$	$P \wedge \neg T$	$\neg P \wedge T$	$\neg P \wedge \neg T$
$P \wedge W$	$P \wedge \neg W$	$\neg P \wedge W$	$\neg P \wedge \neg W$
$P \wedge U$	$P \wedge \neg U$	$\neg P \wedge U$	$\neg P \wedge \neg U$
$D \wedge T$	$D \wedge \neg T$	$\neg D \wedge T$	$\neg D \wedge \neg T$
$D \wedge W$	$D \wedge \neg W$	$\neg D \wedge W$	$\neg D \wedge \neg W$
$D \wedge U$	$D \wedge \neg U$	$\neg D \wedge U$	$\neg D \wedge \neg U$
$T \wedge W$	$T \wedge \neg W$	$\neg T \wedge W$	$\neg T \wedge \neg W$
$T \wedge U$	$T \wedge \neg U$	$\neg T \wedge U$	$\neg T \wedge \neg U$
	$W \wedge \neg U$	$\neg W \wedge U$	



$\{B, P, D, T, W\}$   
 $\{B, G, D, U\}$   
 $\{B, G, P, T, U\}$   
 $\{B, G, W\}$   
 $\{B, P, W\}$   
 $\{B, D, T, U\}$

## Example of pairwise sampling ( $T=2$ )

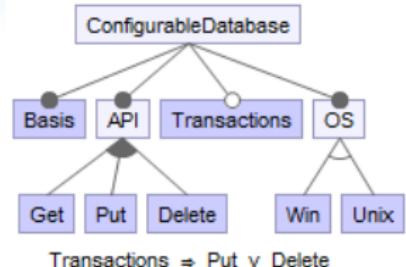
$G \wedge P$	$G \wedge \neg P$	$\neg G \wedge P$	$\neg G \wedge \neg P$
$G \wedge D$	$G \wedge \neg D$	$\neg G \wedge D$	$\neg G \wedge \neg D$
$G \wedge T$	$G \wedge \neg T$	$\neg G \wedge T$	$\neg G \wedge \neg T$
$G \wedge W$	$G \wedge \neg W$	$\neg G \wedge W$	$\neg G \wedge \neg W$
$G \wedge U$	$G \wedge \neg U$	$\neg G \wedge U$	$\neg G \wedge \neg U$
$P \wedge D$	$P \wedge \neg D$	$\neg P \wedge D$	$\neg P \wedge \neg D$
$P \wedge T$	$P \wedge \neg T$	$\neg P \wedge T$	$\neg P \wedge \neg T$
$P \wedge W$	$P \wedge \neg W$	$\neg P \wedge W$	$\neg P \wedge \neg W$
$P \wedge U$	$P \wedge \neg U$	$\neg P \wedge U$	$\neg P \wedge \neg U$
$D \wedge T$	$D \wedge \neg T$	$\neg D \wedge T$	$\neg D \wedge \neg T$
$D \wedge W$	$D \wedge \neg W$	$\neg D \wedge W$	$\neg D \wedge \neg W$
$D \wedge U$	$D \wedge \neg U$	$\neg D \wedge U$	$\neg D \wedge \neg U$
$T \wedge W$	$T \wedge \neg W$	$\neg T \wedge W$	$\neg T \wedge \neg W$
$T \wedge U$	$T \wedge \neg U$	$\neg T \wedge U$	$\neg T \wedge \neg U$
		$\neg W \wedge U$	



- $\{B, P, D, T, W\}$
- $\{B, G, D, U\}$
- $\{B, G, P, T, U\}$
- $\{B, G, W\}$
- $\{B, P, W\}$
- $\{B, D, T, U\}$

## Example of pairwise sampling ( $T=2$ )

$G \wedge P$	$G \wedge \neg P$	$\neg G \wedge P$	$\neg G \wedge \neg P$
$G \wedge D$	$G \wedge \neg D$	$\neg G \wedge D$	$\neg G \wedge \neg D$
$G \wedge T$	$G \wedge \neg T$	$\neg G \wedge T$	$\neg G \wedge \neg T$
$G \wedge W$	$G \wedge \neg W$	$\neg G \wedge W$	$\neg G \wedge \neg W$
$G \wedge U$	$G \wedge \neg U$	$\neg G \wedge U$	$\neg G \wedge \neg U$
$P \wedge D$	$P \wedge \neg D$	$\neg P \wedge D$	$\neg P \wedge \neg D$
$P \wedge T$	$P \wedge \neg T$	$\neg P \wedge T$	$\neg P \wedge \neg T$
$P \wedge W$	$P \wedge \neg W$	$\neg P \wedge W$	$\neg P \wedge \neg W$
$P \wedge U$	$P \wedge \neg U$	$\neg P \wedge U$	$\neg P \wedge \neg U$
$D \wedge T$	$D \wedge \neg T$	$\neg D \wedge T$	$\neg D \wedge \neg T$
$D \wedge W$	$D \wedge \neg W$	$\neg D \wedge W$	$\neg D \wedge \neg W$
$D \wedge U$	$D \wedge \neg U$	$\neg D \wedge U$	$\neg D \wedge \neg U$
$T \wedge W$	$T \wedge \neg W$	$\neg T \wedge W$	$\neg T \wedge \neg W$
$T \wedge U$	$T \wedge \neg U$	$\neg T \wedge U$	$\neg T \wedge \neg U$
		$W \wedge \neg U$	$\neg W \wedge U$



Transactions  $\Rightarrow$  Put v Delete

$\{B, P, D, T, W\}$

$\{B, G, D, U\}$

$\{B, G, P, T, U\}$

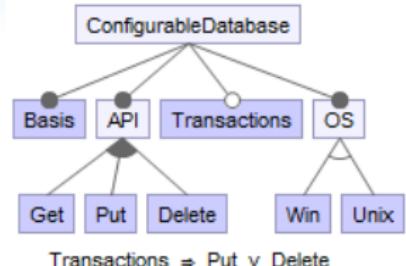
$\{B, G, W\}$

$\{B, P, W\}$

$\{B, D, T, U\}$

## Example of pairwise sampling ( $T=2$ )

$G \wedge P$	$G \wedge \neg P$	$\neg G \wedge P$	$\neg G \wedge \neg P$
$G \wedge D$	$G \wedge \neg D$	$\neg G \wedge D$	$\neg G \wedge \neg D$
$G \wedge T$	$G \wedge \neg T$	$\neg G \wedge T$	$\neg G \wedge \neg T$
$G \wedge W$	$G \wedge \neg W$	$\neg G \wedge W$	$\neg G \wedge \neg W$
$G \wedge U$	$G \wedge \neg U$	$\neg G \wedge U$	$\neg G \wedge \neg U$
$P \wedge D$	$P \wedge \neg D$	$\neg P \wedge D$	$\neg P \wedge \neg D$
$P \wedge T$	$P \wedge \neg T$	$\neg P \wedge T$	$\neg P \wedge \neg T$
$P \wedge W$	$P \wedge \neg W$	$\neg P \wedge W$	$\neg P \wedge \neg W$
$P \wedge U$	$P \wedge \neg U$	$\neg P \wedge U$	$\neg P \wedge \neg U$
$D \wedge T$	$D \wedge \neg T$	$\neg D \wedge T$	$\neg D \wedge \neg T$
$D \wedge W$	$D \wedge \neg W$	$\neg D \wedge W$	$\neg D \wedge \neg W$
$D \wedge U$	$D \wedge \neg U$	$\neg D \wedge U$	$\neg D \wedge \neg U$
$T \wedge W$	$T \wedge \neg W$	$\neg T \wedge W$	$\neg T \wedge \neg W$
$T \wedge U$	$T \wedge \neg U$	$\neg T \wedge U$	$\neg T \wedge \neg U$
		$W \wedge \neg U$	$\neg W \wedge U$



Transactions  $\Rightarrow$  Put v Delete

$\{B, P, D, T, W\}$

$\{B, G, D, U\}$

$\{B, G, P, T, U\}$

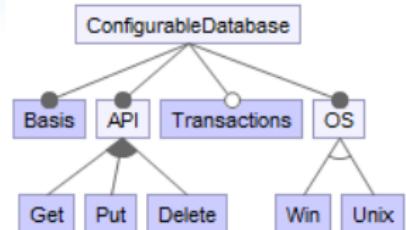
$\{B, G, W\}$

$\{B, P, W\}$

$\{B, D, T, U\}$

## Example of pairwise sampling ( $T=2$ )

$G \wedge P$	$G \wedge \neg P$	$\neg G \wedge P$	$\neg G \wedge \neg P$
$G \wedge D$	$G \wedge \neg D$	$\neg G \wedge D$	$\neg G \wedge \neg D$
$G \wedge T$	$G \wedge \neg T$	$\neg G \wedge T$	$\neg G \wedge \neg T$
$G \wedge W$	$G \wedge \neg W$	$\neg G \wedge W$	$\neg G \wedge \neg W$
$G \wedge U$	$G \wedge \neg U$	$\neg G \wedge U$	$\neg G \wedge \neg U$
$P \wedge D$	$P \wedge \neg D$	$\neg P \wedge D$	$\neg P \wedge \neg D$
$P \wedge T$	$P \wedge \neg T$	$\neg P \wedge T$	$\neg P \wedge \neg T$
$P \wedge W$	$P \wedge \neg W$	$\neg P \wedge W$	$\neg P \wedge \neg W$
$P \wedge U$	$P \wedge \neg U$	$\neg P \wedge U$	$\neg P \wedge \neg U$
$D \wedge T$	$D \wedge \neg T$	$\neg D \wedge T$	$\neg D \wedge \neg T$
$D \wedge W$	$D \wedge \neg W$	$\neg D \wedge W$	$\neg D \wedge \neg W$
$D \wedge U$	$D \wedge \neg U$	$\neg D \wedge U$	$\neg D \wedge \neg U$
$T \wedge W$	$T \wedge \neg W$	$\neg T \wedge W$	$\neg T \wedge \neg W$
$T \wedge U$	$T \wedge \neg U$	$\neg T \wedge U$	$\neg T \wedge \neg U$
		$W \wedge \neg U$	$\neg W \wedge U$

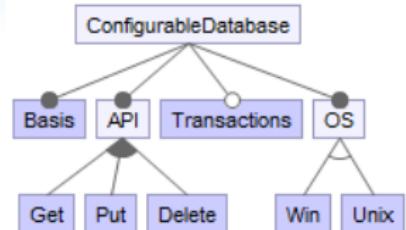


Transactions  $\Rightarrow$  Put v Delete

- $\{B, P, D, T, W\}$
- $\{B, G, D, U\}$
- $\{B, G, P, T, U\}$
- $\{B, G, W\}$
- $\{B, P, W\}$
- $\{B, D, T, U\}$

## Example of pairwise sampling ( $T=2$ )

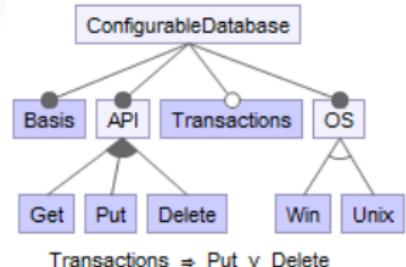
$G \wedge P$	$G \wedge \neg P$	$\neg G \wedge P$	$\neg G \wedge \neg P$
$G \wedge D$	$G \wedge \neg D$	$\neg G \wedge D$	$\neg G \wedge \neg D$
$G \wedge T$	$G \wedge \neg T$	$\neg G \wedge T$	$\neg G \wedge \neg T$
$G \wedge W$	$G \wedge \neg W$	$\neg G \wedge W$	$\neg G \wedge \neg W$
$G \wedge U$	$G \wedge \neg U$	$\neg G \wedge U$	$\neg G \wedge \neg U$
$P \wedge D$	$P \wedge \neg D$	$\neg P \wedge D$	$\neg P \wedge \neg D$
$P \wedge T$	$P \wedge \neg T$	$\neg P \wedge T$	$\neg P \wedge \neg T$
$P \wedge W$	$P \wedge \neg W$	$\neg P \wedge W$	$\neg P \wedge \neg W$
$P \wedge U$	$P \wedge \neg U$	$\neg P \wedge U$	$\neg P \wedge \neg U$
$D \wedge T$	$D \wedge \neg T$	$\neg D \wedge T$	$\neg D \wedge \neg T$
$D \wedge W$	$D \wedge \neg W$	$\neg D \wedge W$	$\neg D \wedge \neg W$
$D \wedge U$	$D \wedge \neg U$	$\neg D \wedge U$	$\neg D \wedge \neg U$
$T \wedge W$	$T \wedge \neg W$	$\neg T \wedge W$	$\neg T \wedge \neg W$
$T \wedge U$	$T \wedge \neg U$	$\neg T \wedge U$	$\neg T \wedge \neg U$
		$W \wedge \neg U$	$\neg W \wedge U$



$\{B, P, D, T, W\}$   
 $\{B, G, D, U\}$   
 $\{B, G, P, T, U\}$   
 $\{B, G, W\}$   
 $\{B, P, W\}$   
 $\{B, D, T, U\}$

## Example of pairwise sampling ( $T=2$ )

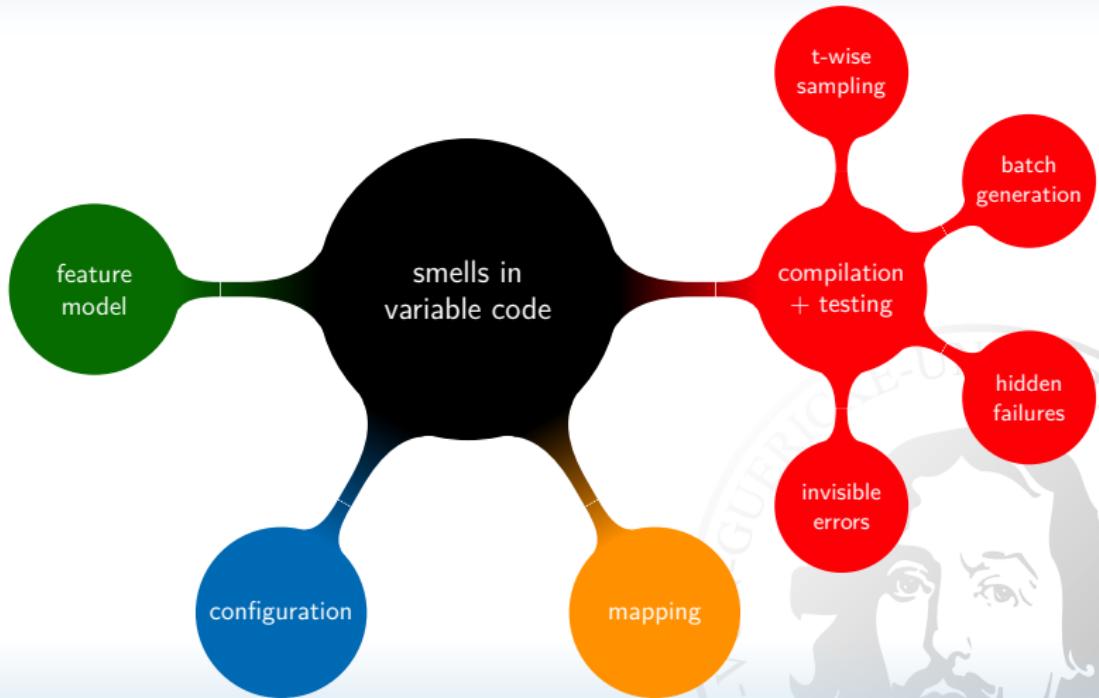
$G \wedge P$	$G \wedge \neg P$	$\neg G \wedge P$	$\neg G \wedge \neg P$
$G \wedge D$	$G \wedge \neg D$	$\neg G \wedge D$	$\neg G \wedge \neg D$
$G \wedge T$	$G \wedge \neg T$	$\neg G \wedge T$	$\neg G \wedge \neg T$
$G \wedge W$	$G \wedge \neg W$	$\neg G \wedge W$	$\neg G \wedge \neg W$
$G \wedge U$	$G \wedge \neg U$	$\neg G \wedge U$	$\neg G \wedge \neg U$
$P \wedge D$	$P \wedge \neg D$	$\neg P \wedge D$	$\neg P \wedge \neg D$
$P \wedge T$	$P \wedge \neg T$	$\neg P \wedge T$	$\neg P \wedge \neg T$
$P \wedge W$	$P \wedge \neg W$	$\neg P \wedge W$	$\neg P \wedge \neg W$
$P \wedge U$	$P \wedge \neg U$	$\neg P \wedge U$	$\neg P \wedge \neg U$
$D \wedge T$	$D \wedge \neg T$	$\neg D \wedge T$	$\neg D \wedge \neg T$
$D \wedge W$	$D \wedge \neg W$	$\neg D \wedge W$	$\neg D \wedge \neg W$
$D \wedge U$	$D \wedge \neg U$	$\neg D \wedge U$	$\neg D \wedge \neg U$
$T \wedge W$	$T \wedge \neg W$	$\neg T \wedge W$	$\neg T \wedge \neg W$
$T \wedge U$	$T \wedge \neg U$	$\neg T \wedge U$	$\neg T \wedge \neg U$
	$W \wedge \neg U$	$\neg W \wedge U$	



$\{B, P, D, T, W\}$   
 $\{B, G, D, U\}$   
 $\{B, G, P, T, U\}$   
 $\{B, G, W\}$   
 $\{B, P, W\}$   
 $\{B, D, T, U\}$

Already 6/26 configurations cover all pairwise interactions

# Tutorial overview



## Hands-on V: fix smells with compilation and testing

---

1. (Continue with previous project or import Elevator04)
2. Generate and compile all six configurations by right click on project > FeatureIDE > Product Generator > All current configurations, deselect JUnit tests
3. Fix compiler errors for configuration Starter by adding annotation `//#if CallButtons` (try autocompletion) around `onRequestFinished` and repeat 2. for validation
4. Repeat 2. but with selection of JUnit tests to uncover problem with processing order of feature FIFO
5. Assign a color to feature FIFO to identify and fix the problem in class Request and repeat 4. for validation
6. Repeat 4. but select ICPL and Chvatal with different values of T to identify three lines that should be removed

## Part VIII

Closing remarks on FeatureIDE



# Available under GPLv3 on Github

FeatureIDE / FeatureIDE

Unwatch 34 Star 24 Fork 29

Code Issues 83 Pull requests 0 Wiki Pulse Graphs

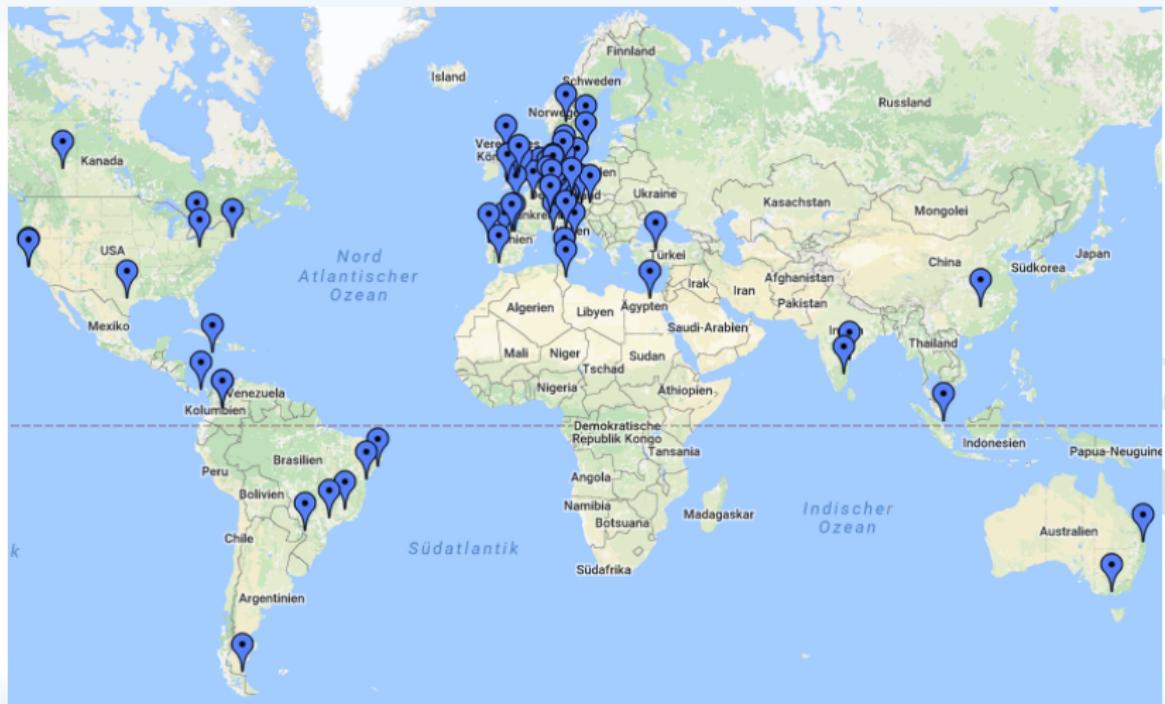
An extensible framework for feature-oriented software development <http://featureide.cs.ovgu.de/>

3,434 commits 24 branches 24 releases 33 contributors

Branch: develop New pull request Create new file Upload files Find file Clone or download ▾

meinicke	automatic organize imports	Latest commit c982d12 10 hours ago
.github	Update ISSUE_TEMPLATE	7 months ago
deploy	Created new version v3.1.0 and uploaded it to the update site	2 months ago
eclipse_settings	line endings	5 months ago
experimental	Updated license text's year to 2016.	6 months ago
featuremodels	Introduce end-of-line normalization	2 years ago
lib	Introduce end-of-line normalization	2 years ago
misc/FeatureModelBuilder	Updated file header in path misc/* and tests/*	2 years ago
plugins	automatic organize imports	10 hours ago
tests	Revert "Removed unnecessary libraries and dependencies."	2 months ago

# Support requests (since 2007)



# Follow FeatureIDE on Twitter



**FeatureIDE**  
@FeatureIDE

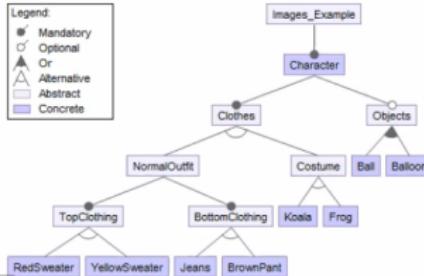
Eclipse plug-ins for feature-oriented software development  
[www1.cs.uni-magdeburg.de/iti\\_db/research...](http://www1.cs.uni-magdeburg.de/iti_db/research...)

Beigetreten März 2015

Fotos und Videos



Legend:  
• Mandatory  
○ Optional  
▲ Or  
△ Alternative  
□ Abstract  
■ Concrete



TWEETS 20 FOLLOWER 33 GEFÄLLT MIR 4

[Folgen](#)

**Tweets** **Tweets & Antworten** **Medien**

FeatureIDE hat retweetet

**Jabier Martinez** @jabier\_lab · 1. Aug.  
Images composer enhancement for next @FeatureIDE version, now t-wise generation is working.

Neu bei Twitter?  
Melde Dich jetzt an, um Deine eigene, personalisierte Timeline zu erhalten!

[Registrieren](#)

Vieelleicht gefällt Dir auch -  
Aktualisieren

**DBSE working group** @DBSEgroup

**Leonardo Passos** @lnrdps

**Norbert Siegmund** @Norbsen

**Thorsten Berger** @thorsten\_berger

**Eric Walkingshaw** @EricWalkingshaw

GIF

1 6 9 \*\*\*

# Enjoy FeatureIDE on Youtube

## FeatureIDE

 Abonnieren 21



This is the YouTube channel of FeatureIDE an Eclipse plug-in for Feature-Oriented Software Development.  
Mehr anzeigen

## Uploads

ANGESEHEN



Support for runtime-variability using a Run Configuration

2:44

ANGESEHEN



Support for runtime-variability using property files

3:54

ANGESEHEN



Framework support in FeatureIDE

5:10

ANGESEHEN



Scalable Product Configuration  
of Variable Systems

4:58

Support for runtime-variability  
with run configurations

119 Aufrufe • vor 5 Monaten

Support for runtime-variability  
using property files

69 Aufrufe • vor 5 Monaten

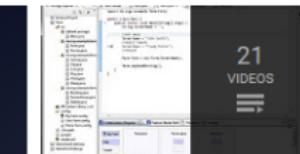
Framework support in FeatureIDE

89 Aufrufe • vor 5 Monaten

Scalable Product Configuration  
of Variable Systems

169 Aufrufe • vor 7 Monaten

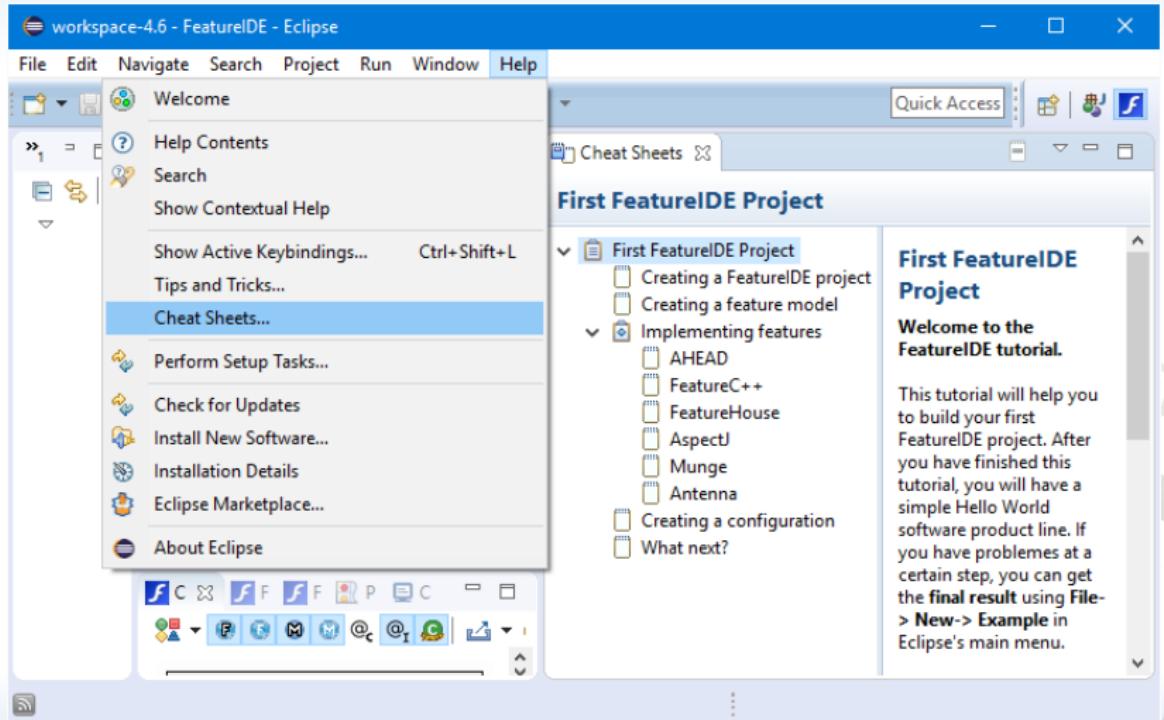
## Eigene Playlists



Software Projects

Demonstrations

# Learn more with FeatureIDE cheat sheet



---

# Feedback and discussion

---

Keep it short (1-2 sentences):

1. What did you learn?
2. What should be improved?



# The end

