



Clean Your Variable Code with FeatureIDE

Thomas Thüm, Sebastian Krieter, Thomas Leich

September 11, 2018 — SPLC'18 in Gothenburg, Schweden

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!



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

Part II

Product-line maintenance is difficult



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  
    //#ifndef (!media_time)  
    this.sounds[soundIndex].setMediaTime(0);  
    //#endif  
    //#endif  
}
```

Preprocessing principle is easy

Best Lap

```
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

Best Lap

```
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 vo  
this.sou  
}  
}
```

```
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();  
    this.sounds[soundIndex].deallocate();  
}
```

```
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

Jörg Listig, Sven Apel,
and Michael Kästner
University of Passau
[jlistig.apel.jengauer]@fim.uni-passau.de

Christian Kästner and Michael Schulze
Chair of Software Engineering
(ckae@informatik.vu.edu)

ABSTRACT
Given the popularity of the preprocessing cap was developed to increase the programming language C by lightweight meta-programming capabilities. Despite its few promises and low adoption rate, it has been used in many open source projects and some modern-day software projects to implement variable software. However, there is little research on the use of preprocessing in real-world software projects. In this paper, we analyze the use of preprocessing to implement variability. To address this issue, we have analyzed forty open-source software projects written in C. Specifically, we have analyzed the use of preprocessing to implement program size influence variability? How many are extensible? What is the level of variability? What is the level of generativity or extensibility applied? Which types of macro definitions are used?

Liebig et al. @ ICSE'10

Categories and Subject Descriptions
D.2.1 [Software Engineering]: Coding Tools and Techniques; D.2.2 [Software Engineering]: Metrics; D.3.1 [Program Generation Languages]: Preprocessor - Preprocessors

General Terms

Design Studies

Keywords

Software Product Lines, C Preprocessor

1. INTRODUCTION

The C preprocessor (cpp) is a popular tool for implementing variable software. It has been developed to reduce C by

lightweight meta-programming capabilities and is commonly used to implement variable software. It can be used to generate and define conditional code fragments (a.k.a. conditional include files) [21]. As the cpp tool is based on C, it can be used with most C compilers. It is also supported by many other programming languages such as Java or C#. In the past, it has been observed that the use of preprocessing is increasing [11, 12]. However, the use of preprocessing is often accompanied by a lack of syntactic and semantic errors during the generation of code [11, 12]. This is due to the fact that preprocessing is a form of template and template-like (a.k.a. filled-in) [22, 13]. A downside is maintainability and is evident in reader [23].

The use of preprocessing is not limited to C. It is also a major part of software product line engineering. A software product line (SPL) is a set of software inheritance constructs. Using a

macro definition, the SPL developer can reuse parts of the app tool to avoid code repetition [9, 11, 22].

In this paper, we analyze the use of preprocessing in the app tool to avoid code repetition in the implementation of SPLs [9, 11, 12, 22]. We believe this combination to be interesting because the use of preprocessing in open source projects and SPLs started to grow with a substantial increase of interest in SPLs in the last years [11, 12].

How does program size influence variability of SPLs? How many are extensible? What is the level of variability? What is the level of generativity or extensibility applied? Which types of macro definitions are used? We hope that insights into the typical application problems of lines solved with cpp help to judge whether cpp usage causes maintainability problems and how to solve them. We also hope that insights into the typical application problems of lines solved with cpp help to judge whether cpp usage causes maintainability problems and how to solve them. We also hope that insights into the typical application problems of lines solved with cpp help to judge whether cpp usage causes maintainability problems and how to solve them.

To answer the questions above, we analyzed forty open-source software projects written in C. The analyzed projects have less than 100k lines of code taken from different domains including web servers, databases, and network protocols.

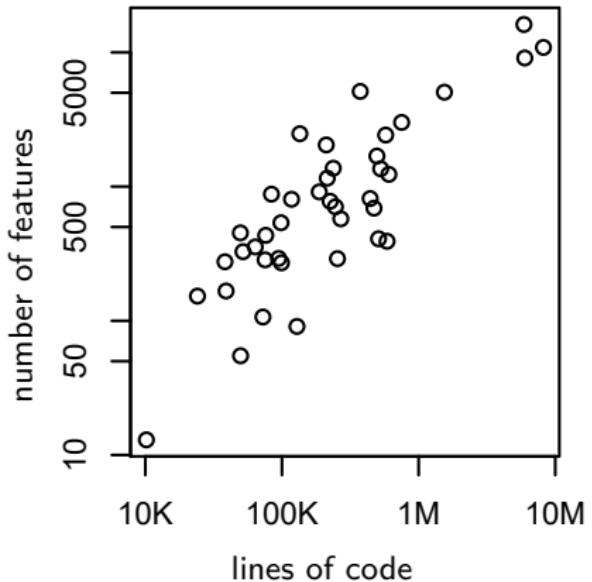
We propose a set of metrics that allow us to index and classify app usage patterns and to merge the patterns to common SPLs. We also propose a set of metrics that allow us to index and classify app usage patterns and to merge the patterns to common SPLs. We also propose a set of metrics that allow us to index and classify app usage patterns and to merge the patterns to common SPLs.

Permissions to make digital or hard copies of all or part of this work for personal use or internal distribution to persons not affiliated with the author are granted without prior permission or royalty fees. For those interested in making electronic distributions, or in giving prior permission to make electronic copies of part or all of this work for distribution outside their institution, please apply to ACM. © 2010, May 2-8, 2010, Cape Town, South Africa. Conference on Software Engineering, 300-310.



Preprocessor code is complex

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 Listig, Sven Apel,
and Michael Schulze
University of Passau
[jlistig.apel.lengauer]@fim.uni-passau.de

Christian Kästner and Michael Schulze
Chair of Software Engineering
(ckae@informatik.uni-goettingen.de)

ABSTRACT
Given the complexity of the preprocessor, the preprocessor gap was identified by the research community as one of the major challenges in software product line engineering. Despite its severe problems and low adoption rate, the preprocessor is still used in many software product line engineering projects to implement variable software. However, there is little research on how to effectively manage the preprocessor to implement variability. To address this issue, we have analyzed forty open-source software projects written in C, C++, Java, and C#, and investigated how they implement variability. How does program size influence variability? How complex are existing code patterns? What types of variability are supported? Which types of granularity are reusability applied? Which types of encapsulation are used?

Liebig et al. @ ICSE'10

Categories and Subject Descriptions
D.2.4 [Software Engineering]: Code Tools and Techniques; D.2.5 [Software Engineering Metrics]; D.3.1 [Program Generation Languages]: Preprocessors

General Terms
Design Studies

Keywords
Software Product Lines, C Preprocessor

1. INTRODUCTION

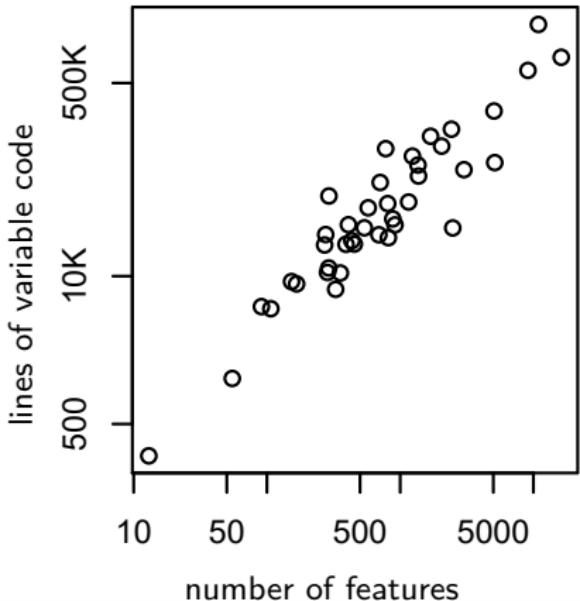
The C preprocessor (`#pre`) is a powerful tool for implementing variable software. It has been developed to reduce the

amount of code in header files of C or C++ code. The code is pre-processed or otherwise to generate software in provided target systems. This has been done to support reuse in commercial and scientific software, but the preprocessor is also used to implement variability in open-source software or to implement or to distribute this, requires prior specific permission.

Permit us to make digital or hard copies of all or part of this work for personal use or internal distribution to persons known to the author, we are made available to distribute to print or electronic commerce and the original author(s) and copyright holders retain all rights. You may not make copies available to others, more well-defined implementation of the preprocessor, such as the C preprocessor, see aspects [37, 24] or various flavors of feature modeling, see aspects [37].

To answer the questions above, we analyzed forty open-source software projects written in C, C++, Java, and C# with respect to their variability and the number of lines of code taken from different domains including scientific, industrial, and commercial software. The purpose is a set of metrics that we use to index and classify preprocessor patterns and to map the patterns to common preprocessor usage patterns. We also show how the preprocessor is used to a large extent to implement and control variability.

ATM '10, May 2-8 2010, Cape Town, South Africa
Copyright © 2010, ACM, New York, NY, USA, 1-4503-0080-0/10/05 \$15.00



The #ifdef hell

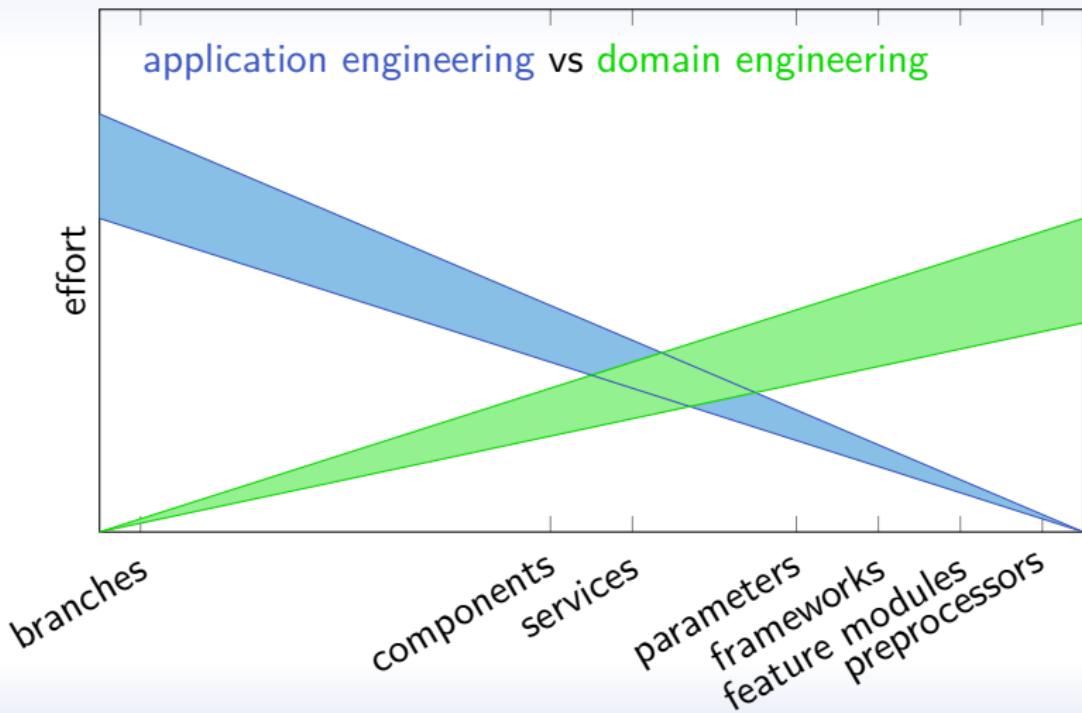


Part III

Feature-oriented software development



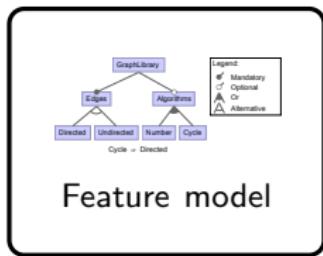
Application engineering vs domain engineering



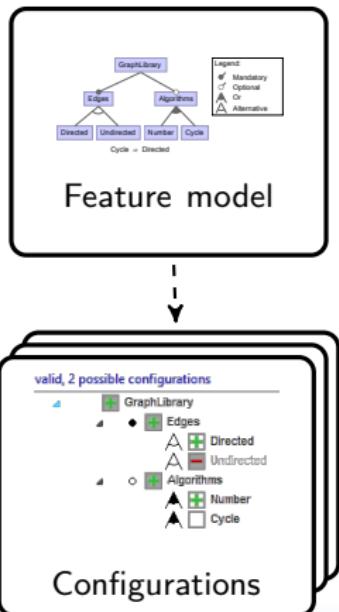
Heterogeneous software artifacts

- ▶ models (in this tutorial: *feature models*)
- ▶ source code (in this tutorial: *Java*)
- ▶ binaries (in this tutorial: *Java Bytecode*)
- ▶ tests (in this tutorial: *JUnit*)
- ▶ specification
- ▶ documentation

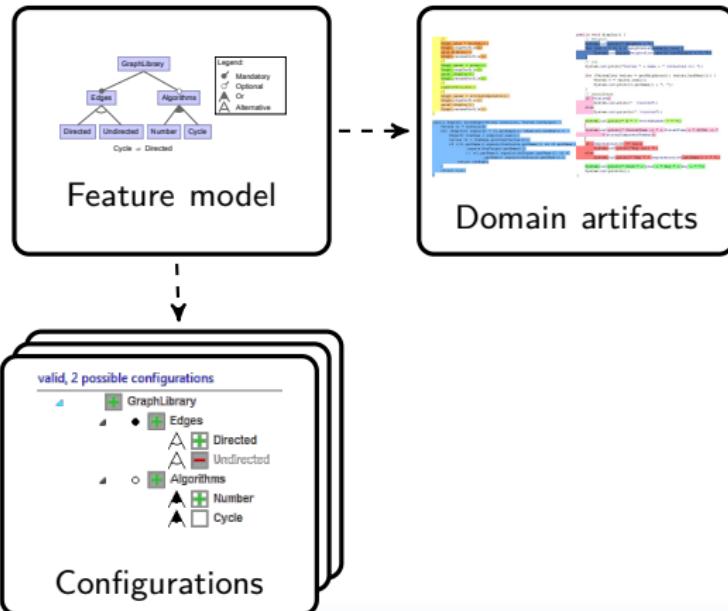
Feature-oriented software development



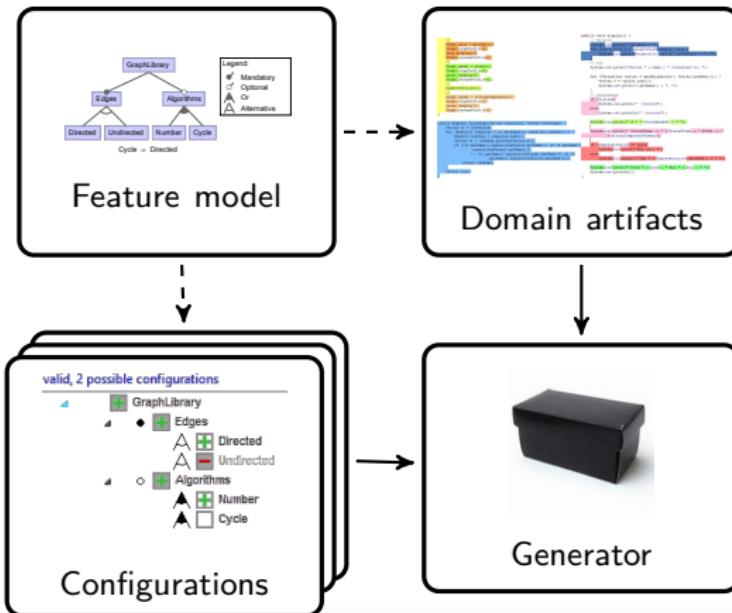
Feature-oriented software development



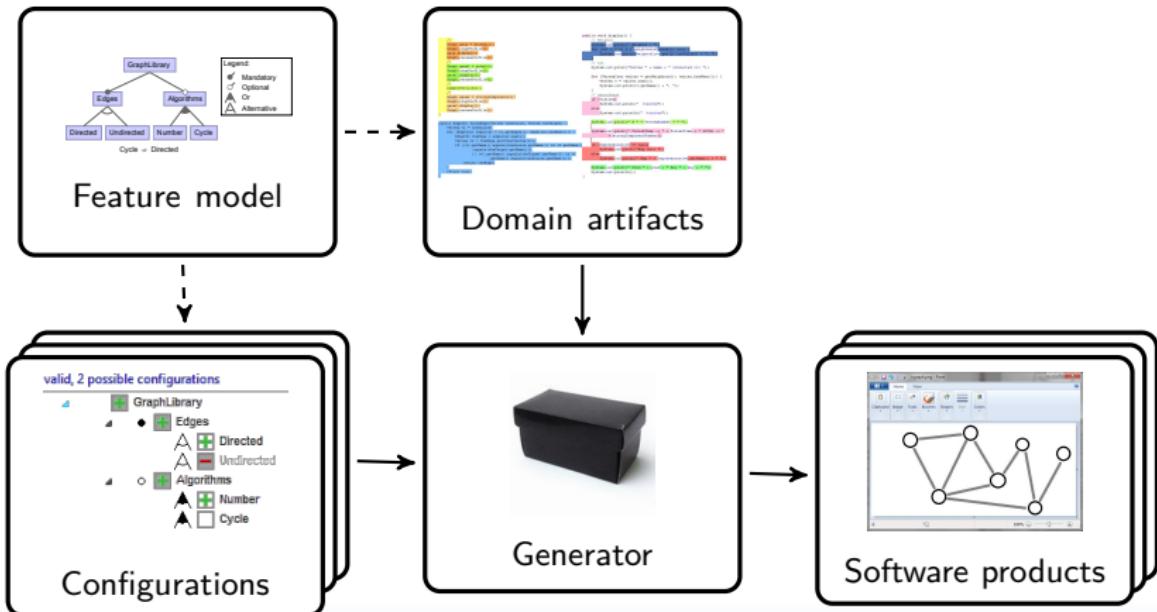
Feature-oriented software development



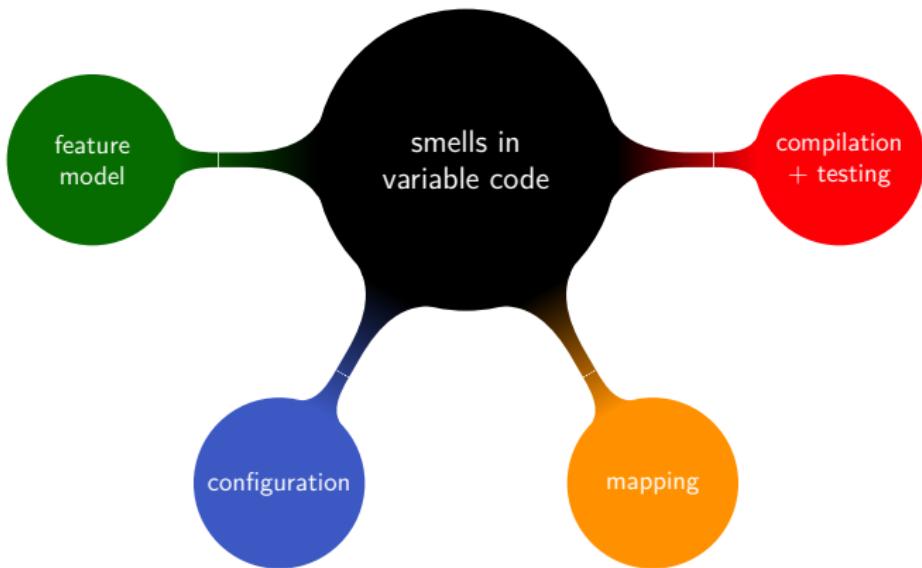
Feature-oriented software development



Feature-oriented software development



Tutorial overview

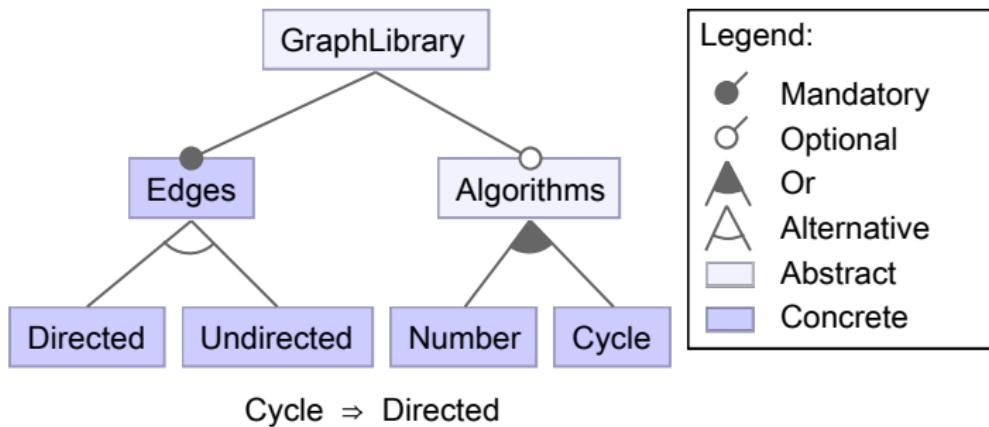


Hands-on I: run different elevator configurations

1. Install Java 1.7 or higher (if not yet installed)
2. Copy/download and unzip FeatureIDE for your OS
3. Start FeatureIDE by running Eclipse
4. Open default workspace (i.e., `../workspace`)
5. Import example Elevator-Antenna-v1.0 using the menu
`File > New > Example > FeatureIDE Examples`
6. Run and try the elevator: right click on folder
`src > Run As > Java Application`
7. Change configuration: right click on `*.xml` in folder
`configs > FeatureIDE > Set as current configuration`
8. Repeat Steps 6 and 7 to experience differences of elevator configurations

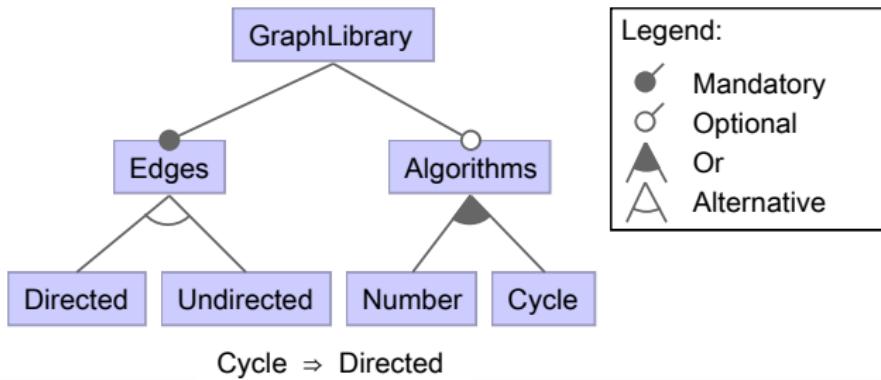
Part IV

Smells in feature modeling



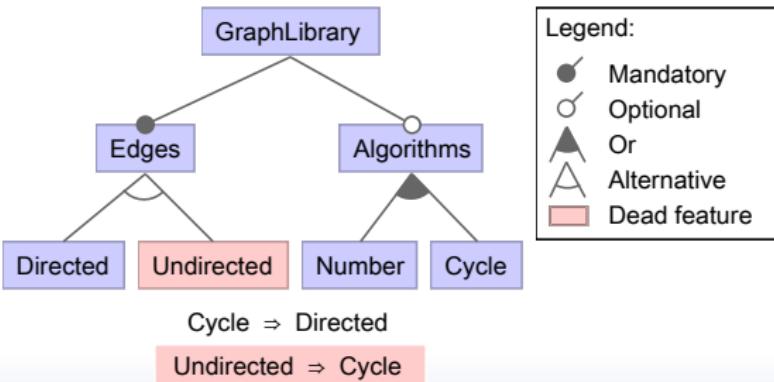
Feature model

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



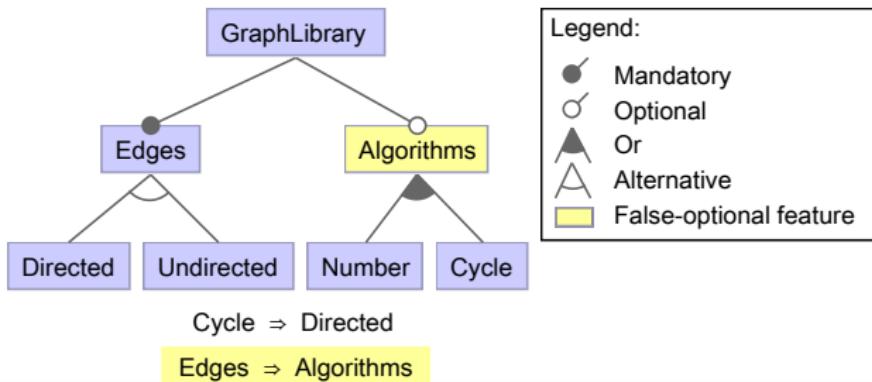
Dead features

- ▶ There is no valid configuration containing that feature
- ▶ Frustrating during configuration, may result in dead code
- ▶ **Fix:** remove feature (refactoring) or remove/edit constraints
- ▶ **Hint:** a feature model is void if all features are dead



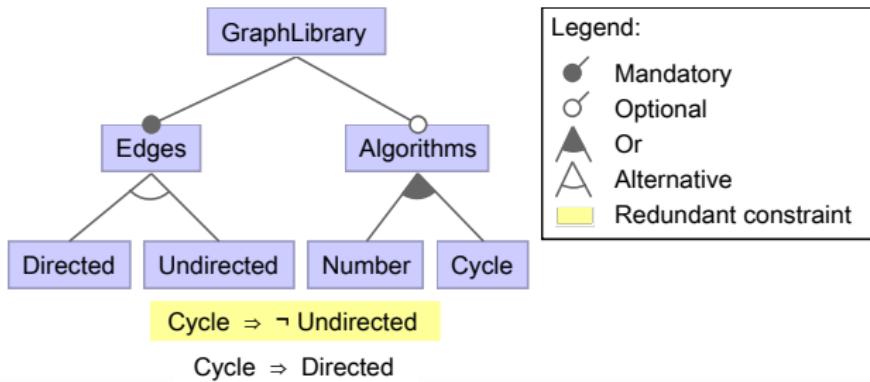
False-optimal features

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



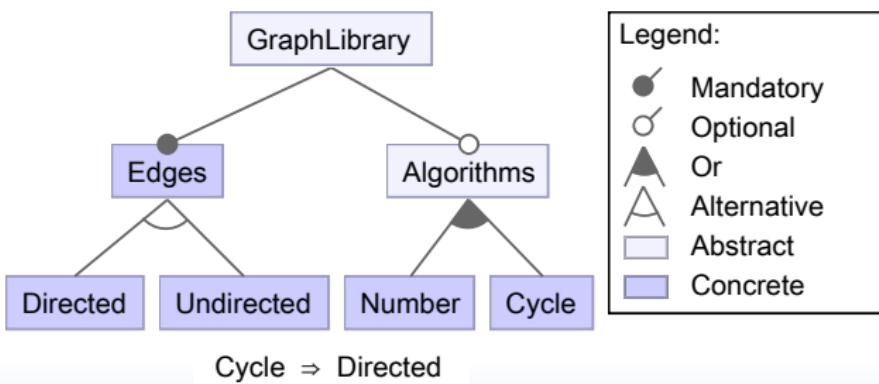
Redundant constraints

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

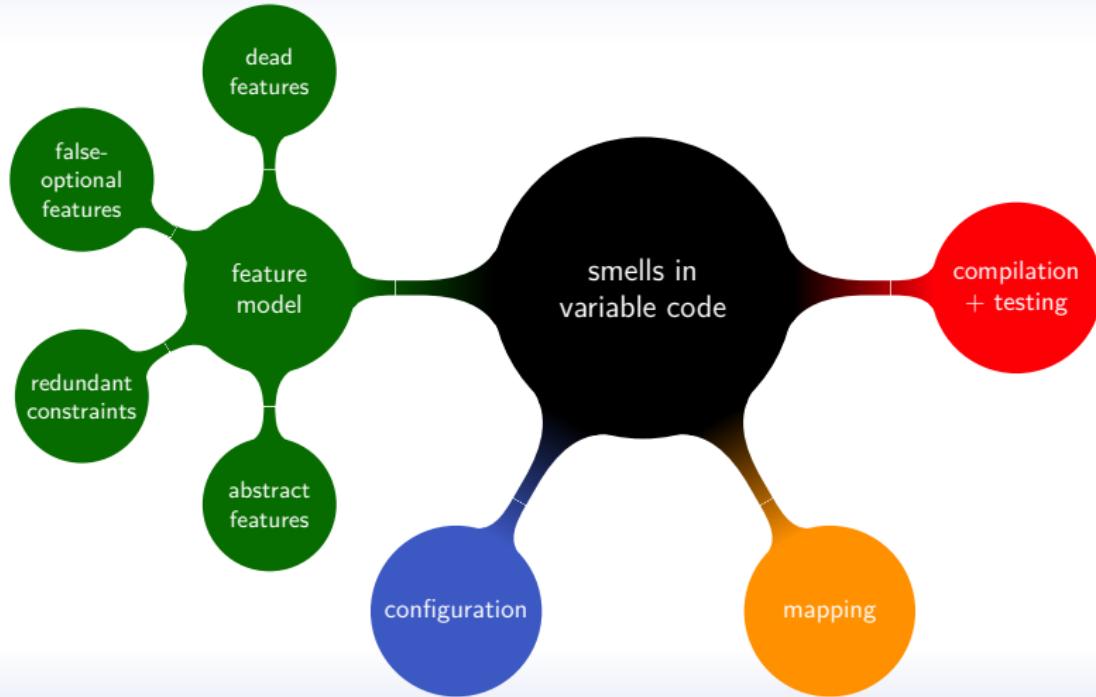


Abstract and concrete features

- ▶ 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

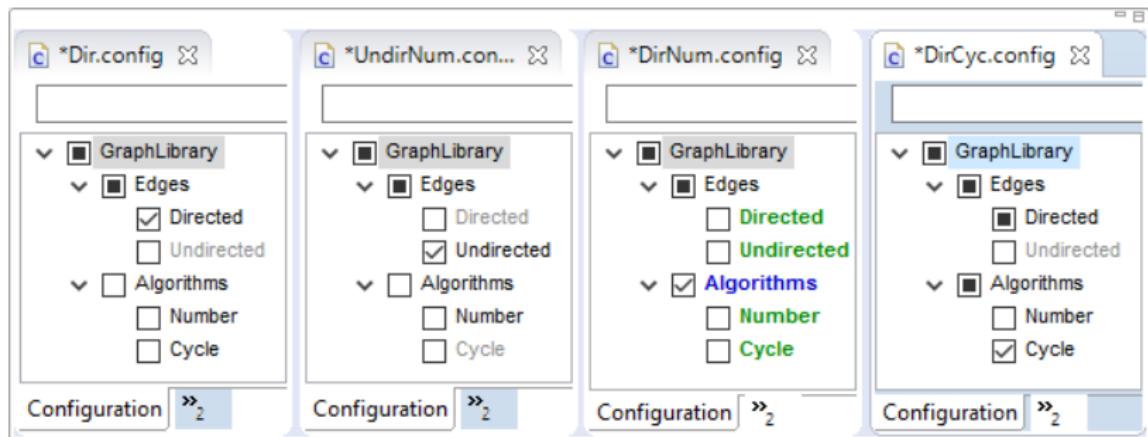


Hands-on II: fix smells in the feature model

1. Continue with previous project Elevator-Antenna-v1.0
2. Open the feature model (i.e., file `model.xml`)
3. Select a feature to see constraints making the feature model void and remove the last constraint in the list
4. Select dead feature *Overloaded* and remove the highlighted constraint
5. Select false-optional feature *Permission* and change it to mandatory
6. Remove redundant constraint (caused by Step 5) and save the feature model
7. Fix warning in problems view regarding feature model (`model.xml`) by changing the mentioned features to abstract

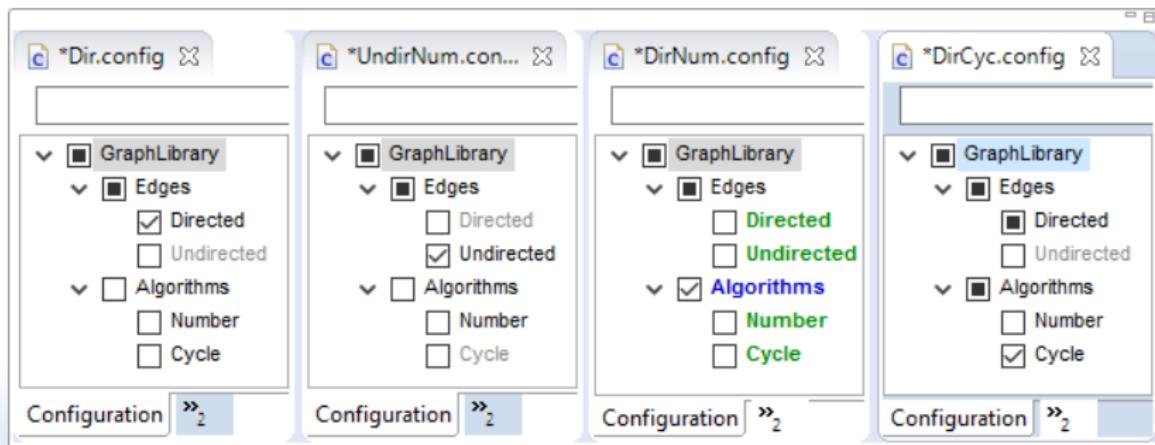
Part V

Smells in configurations



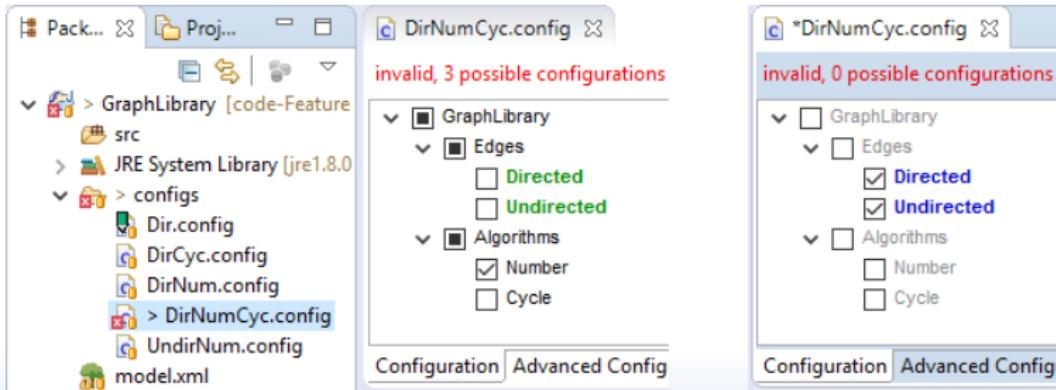
Creating configurations

- 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 contradicts feature model
- ▶ Saved invalid or invalid by change in feature model
- ▶ Computed whenever feature model or configurations change
- ▶ Missing selections (green), conflicting selections (blue)



Never-selected features

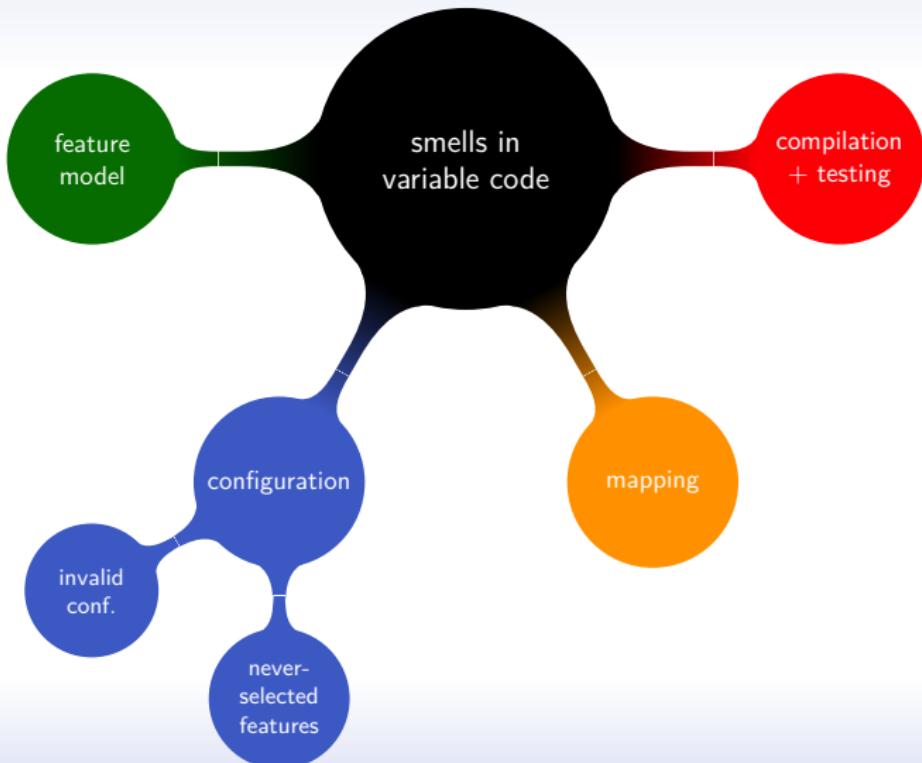
- ▶ Feature is not selected in any configuration (aka. unused)
- ▶ Could indicate unnecessary or untested code

The screenshot shows the FeatureIDE interface with four configuration tabs: Dir.config, DirCyc.config, DirNum.config, and DirNumCyc.config. Each tab displays a feature tree under the GraphLibrary category. The 'Directed' checkbox is checked in all four configurations, while 'Undirected' is unchecked. In Dir.config, 'Number' and 'Cycle' are unchecked. In DirCyc.config, 'Number' is unchecked and 'Cycle' is checked. In DirNum.config, 'Number' is checked and 'Cycle' is unchecked. In DirNumCyc.config, both 'Number' and 'Cycle' are checked.

Below the configurations, a results table lists two items:

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 Elevator-Antenna-v1.1 and close other projects)
2. Create a new configuration called Professional via menu File > New > Configuration File
3. Select the features FIFO, DirectedCall, and FloorPermission; save and close the editor
4. Open feature model; use context menu to change group type below feature Modes to an alternative; save the model
5. Open the invalid configuration Professional; deselect feature FIFO; save the configuration
6. Problem view indicates never-selected features; use Quick Fix in context menu to create a configuration with feature Sabbath; rename configuration to Starter and open it

Part VI

Smells in feature-to-code mapping

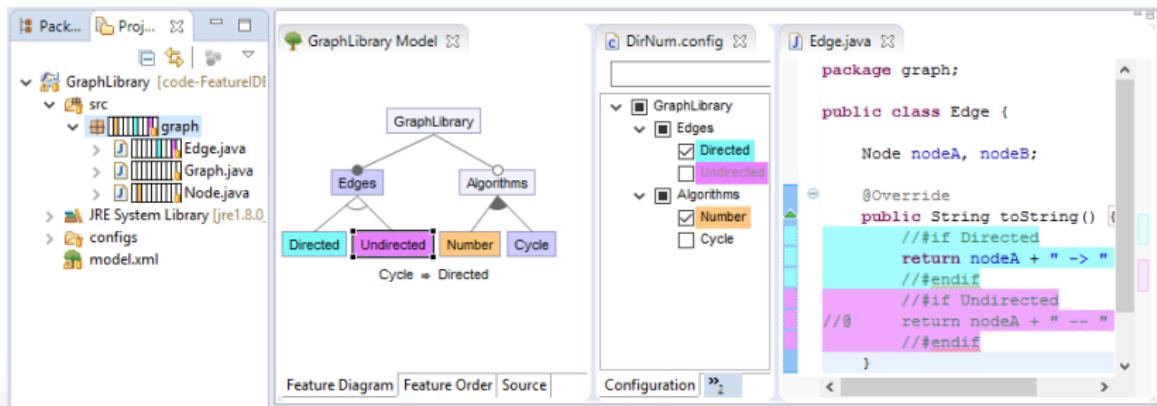
The screenshot displays a software interface with three main panels:

- GraphLibrary Model:** A Feature Diagram showing the structure of the GraphLibrary. It includes a root node "GraphLibrary" which branches into "Edges" and "Algorithms". "Edges" further branches into "DirectedEdges" and "Undirected". "Algorithms" branches into "Number" and "Cycle". A note at the bottom states "Cycle => DirectedEdges". The bottom navigation bar includes tabs for "Feature Diagram", "Feature Order", and "Source".
- DirNum.config:** A configuration editor showing a tree structure under "GraphLibrary". The "Edges" node has two children: "DirectedEdges" (checked) and "Undirected" (unchecked). The "Algorithms" node has two children: "Number" (checked) and "Cycle" (unchecked).
- Edge.java:** A code editor showing the implementation of the Edge class. The code uses conditional logic based on the configuration settings from the DirNum.config file.

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

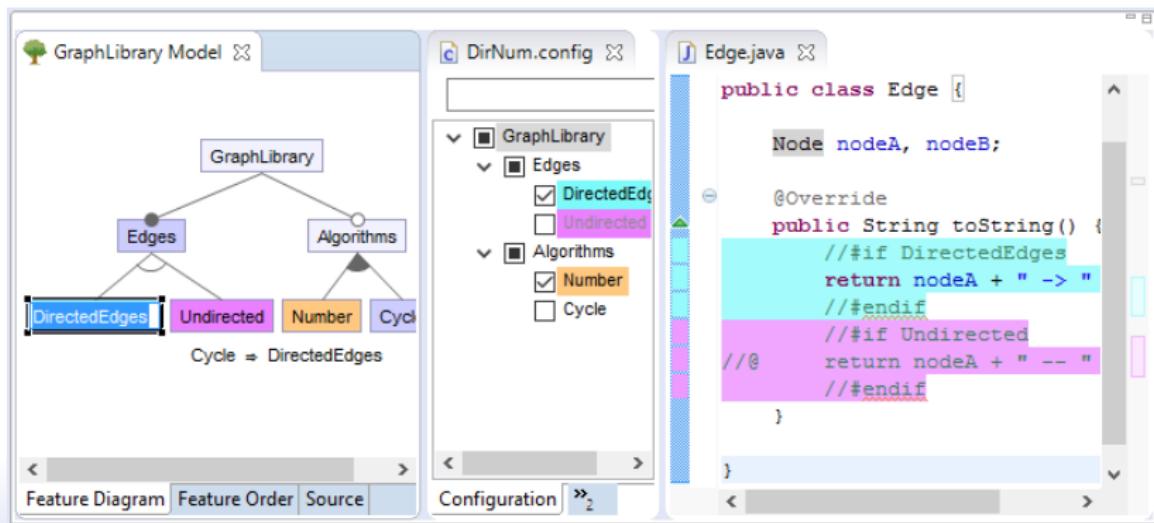
Feature traceability problem

- ▶ 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, meaning of a feature may change
- ▶ 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 hierarchy from 'GraphLibrary' to 'Edges' and 'Algorithms', which further branches into 'Directed', 'Undirected', 'Number', and 'Cycle'. A note at the bottom states 'Cycle => Directed'. Below the diagram are tabs for 'Feature Diagram', 'Feature Order', and 'Source'. The 'Source' tab is selected, showing a portion of the 'Edge.java' code:

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

A warning icon is shown next to the code. On the right, the 'Problems' view shows one error: 'Antenna: Directet is not defined in the feature model and, thus, always assumed to be false'. A tooltip for 'Directet' points to the 'Cycle' feature in the feature model diagram.

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. On the left, a 'GraphLibrary Model' feature diagram is displayed. It includes a 'GraphLibrary' node at the top, which branches into 'Edges' and 'Algorithms'. 'Edges' further branches into 'Directed' and 'Undirected'. 'Algorithms' branches into 'Number' and 'Cycle'. A note below states 'Cycle => Directed'. Below the diagram are tabs for 'Feature Diagram', 'Feature Order', and 'Source'. On the right, an 'Edge.java' code editor window is open, showing the following Java code:

```
@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 one warning: 'Antenna: This expression is a contradiction and causes a dead code block.' The status bar at the bottom indicates '0 errors, 1 warning, 0 others'.

Superfluous preprocessor annotations

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

The screenshot shows a Java code editor window titled "Edge.java". The code contains the following snippet:

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

A warning icon is visible next to the second "#if Directed" line. Below the editor, the FeatureIDE interface is shown with tabs for "Collaboration Diagram", "Feature Model Edits", "FeatureIDE Statistics", and "Problems". The "Problems" tab is active, displaying "0 errors, 1 warning, 0 others". Under the "Warnings" section, there is one 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 `Elevator-Antenna-v1.2` and close other projects)
2. Trace the misspelled feature `ShortestPath` in source code
 - 2.1 Open feature model and use context menu to assign a color to feature `ShortestPath`
 - 2.2 Identify source files containing this feature using the project explorer (*not package*) and open one of them
 - 2.3 Rename the feature in feature model and save it; check whether renaming is applied to configurations and source code
3. Locate and fix issues identified as warnings in problems view
 - 3.1 Fix misspelled feature name `DirectedCall` in the source code
 - 3.2 Remove dead code blocks (remove enclosed Java code)
 - 3.3 Remove superfluous preprocessor statements (keep Java code)

Part VII

Smells in compilation and testing



Invisible compiler errors

- ▶ Compiler errors may exist only in some configurations
- ▶ Problematic if found late in development process
- ▶ Large effort to create and generate many configurations

The image shows two side-by-side Java code editors. The left editor is for the file `Edge.java` and the right for `Graph.java`. Both files contain code related to graph structures, including classes `Edge` and `Graph`, and their respective methods and fields. In the `Edge.java` editor, there are several syntax errors highlighted with red squiggly lines under words like `package`, `public`, and `class`. The `Graph.java` editor also has similar red squiggly lines under the same set of reserved words. These errors are likely due to the code being incomplete or不合乎某些编译配置要求，从而导致在某些环境下无法通过编译。

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

```
package graph;

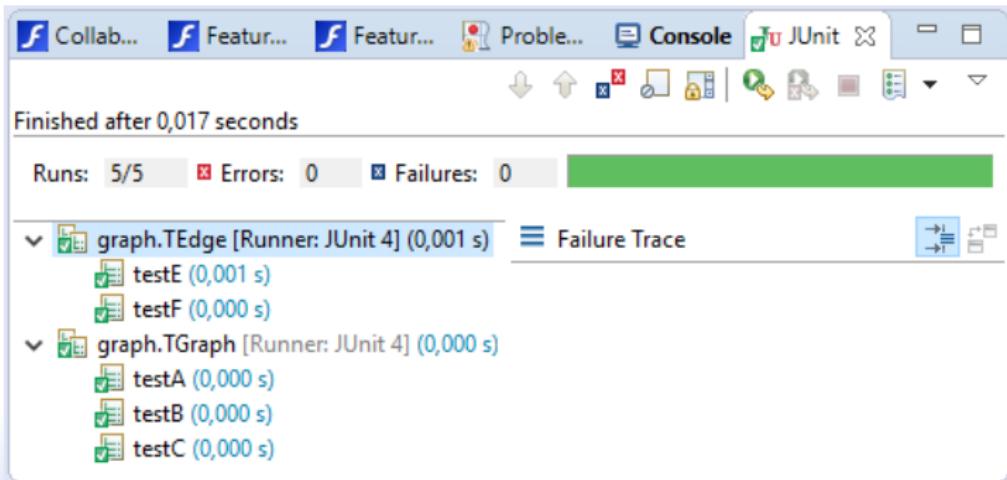
import java.util.List;

public class Graph {

    List<Node> nodes;
    List<Edge> edges;
}
```

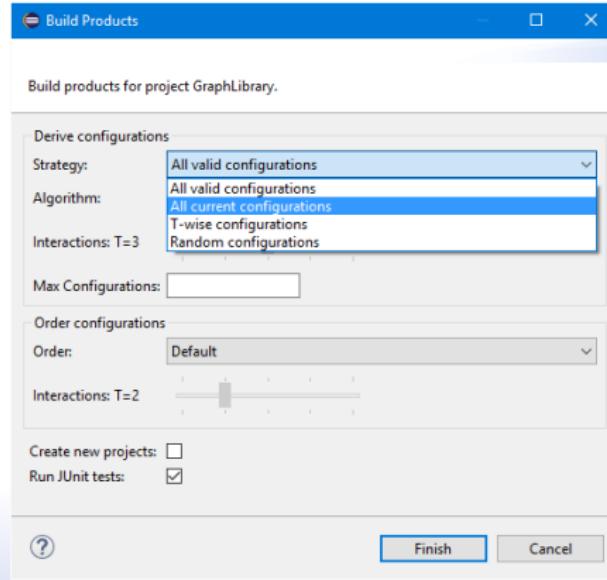
Hidden test failures

- ▶ Each unit test may or may not fail for some configurations
- ▶ Testing only one configuration is insufficient
- ▶ Running unit tests manually for each valid configuration and interpreting the results is time-consuming



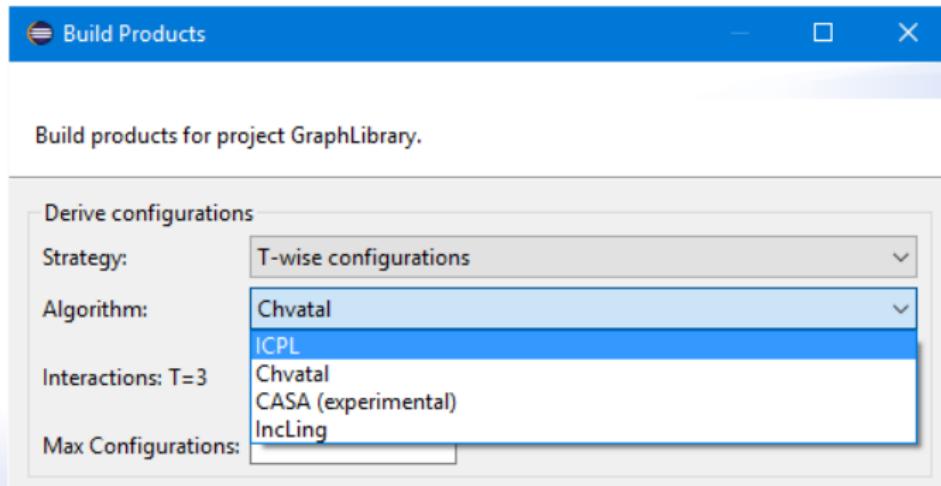
Solution: product generation in batch mode

- ▶ 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



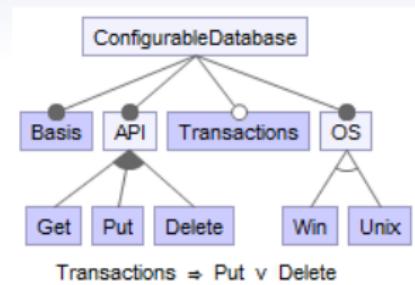
T-wise sampling

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



Example of pairwise sampling ($T=2$)

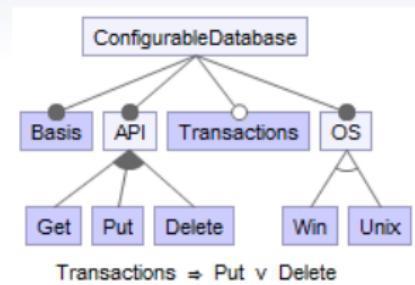
$G \wedge P$	$G \wedge \neg P$	$\neg G \wedge P$	$\neg G \wedge \neg P$
$G \wedge D$	$G \wedge \neg D$	$\neg G \wedge D$	$\neg G \wedge \neg D$
$G \wedge T$	$G \wedge \neg T$	$\neg G \wedge T$	$\neg G \wedge \neg T$
$G \wedge W$	$G \wedge \neg W$	$\neg G \wedge W$	$\neg G \wedge \neg W$
$G \wedge U$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



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

Example of pairwise sampling ($T=2$)

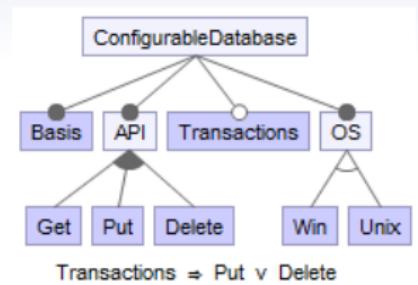
$G \wedge P$	$G \wedge \neg P$	$\neg G \wedge P$	$\neg G \wedge \neg P$
$G \wedge D$	$G \wedge \neg D$	$\neg G \wedge D$	$\neg G \wedge \neg D$
$G \wedge T$	$G \wedge \neg T$	$\neg G \wedge T$	$\neg G \wedge \neg T$
$G \wedge W$	$G \wedge \neg W$	$\neg G \wedge W$	$\neg G \wedge \neg W$
$G \wedge U$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



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

Example of pairwise sampling ($T=2$)

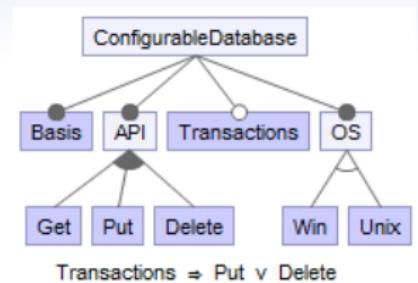
$G \wedge P$	$G \wedge \neg P$	$\neg G \wedge P$	$\neg G \wedge \neg P$
$G \wedge D$	$G \wedge \neg D$	$\neg G \wedge D$	$\neg G \wedge \neg D$
$G \wedge T$	$G \wedge \neg T$	$\neg G \wedge T$	$\neg G \wedge \neg T$
$G \wedge W$	$G \wedge \neg W$	$\neg G \wedge W$	$\neg G \wedge \neg W$
$G \wedge U$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



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

Example of pairwise sampling ($T=2$)

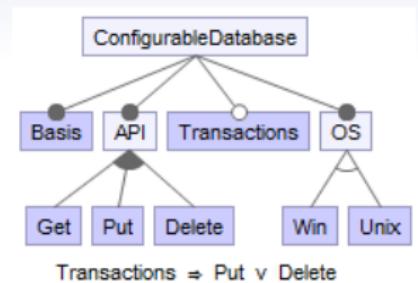
$G \wedge P$	$G \wedge \neg P$	$\neg G \wedge P$	$\neg G \wedge \neg P$
$G \wedge D$	$G \wedge \neg D$	$\neg G \wedge D$	$\neg G \wedge \neg D$
$G \wedge T$	$G \wedge \neg T$	$\neg G \wedge T$	$\neg G \wedge \neg T$
$G \wedge W$	$G \wedge \neg W$	$\neg G \wedge W$	$\neg G \wedge \neg W$
$G \wedge U$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



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

Example of pairwise sampling ($T=2$)

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

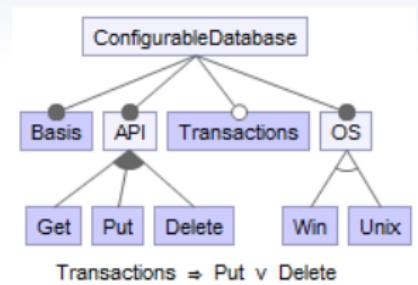


Transactions \Rightarrow Put v Delete

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

Example of pairwise sampling ($T=2$)

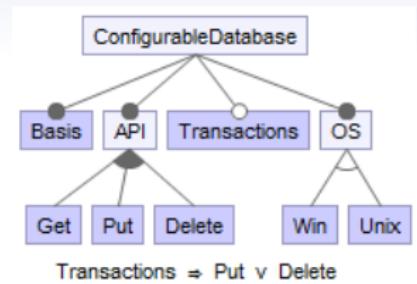
$G \wedge P$	$G \wedge \neg P$	$\neg G \wedge P$	$\neg G \wedge \neg P$
$G \wedge D$	$G \wedge \neg D$	$\neg G \wedge D$	$\neg G \wedge \neg D$
$G \wedge T$	$G \wedge \neg T$	$\neg G \wedge T$	$\neg G \wedge \neg T$
$G \wedge W$	$G \wedge \neg W$	$\neg G \wedge W$	$\neg G \wedge \neg W$
$G \wedge U$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



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

Example of pairwise sampling ($T=2$)

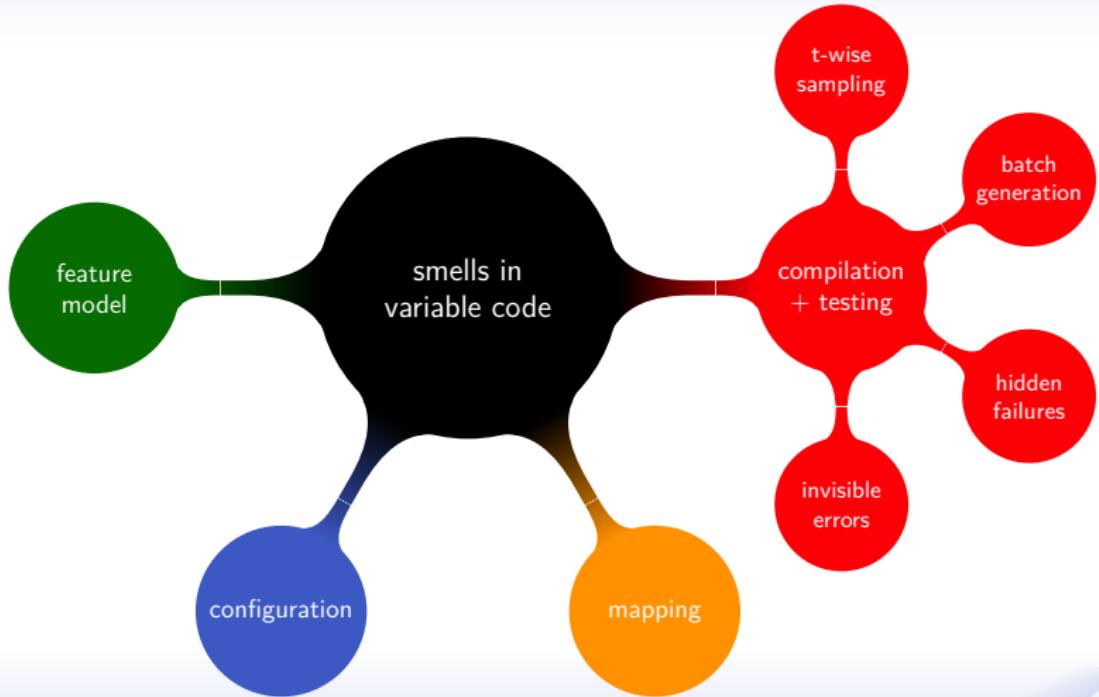
$G \wedge P$	$G \wedge \neg P$	$\neg G \wedge P$	$\neg G \wedge \neg P$
$G \wedge D$	$G \wedge \neg D$	$\neg G \wedge D$	$\neg G \wedge \neg D$
$G \wedge T$	$G \wedge \neg T$	$\neg G \wedge T$	$\neg G \wedge \neg T$
$G \wedge W$	$G \wedge \neg W$	$\neg G \wedge W$	$\neg G \wedge \neg W$
$G \wedge U$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



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

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 Elevator-Antenna-v1.3 and close other projects)
2. Generate and compile all six configurations by right clicking on project > FeatureIDE > Product Generator > All current configurations, deselect JUnit tests
3. Fix compiler errors for configuration Starter by adding annotations `//#if CallButtons` (try autocompletion) and `//#endif` around `ITickListener.onRequestFinished()` and repeat Step 2 for validation
4. Repeat Step 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 Step 4 for validation
6. Repeat Step 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 LGPLv3 on Github

FeatureIDE / FeatureIDE

Unwatch 34 Star 24 Fork 29

Code Issues 83 Pull requests 0 Wiki Pulse Graphs

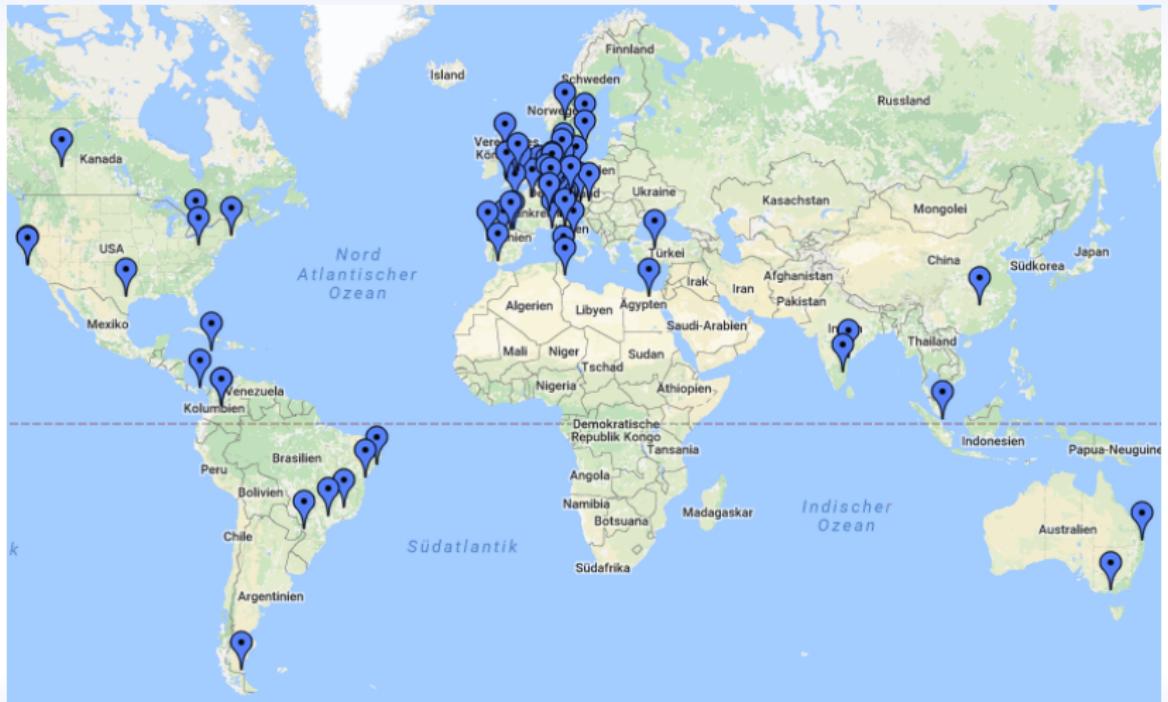
An extensible framework for feature-oriented software development <http://featureide.cs.ovgu.de/>

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

More than 100 support requests since 2007



Follow FeatureIDE on Twitter

The screenshot shows the Twitter profile of @FeatureIDE. The header features a large blue circular logo with the 'f' icon and the word 'IDE feature'. Below the header, the profile information includes:

- Tweets: 48
- Follower: 64
- Gefällt mir: 18
- Listen: 0
- Moments: 0

On the right, there's a "Profil bearbeiten" button. The main feed section has three tabs: "Tweets", "Tweets & Antworten", and "Medien". The first tweet is from @FeatureIDE (@FeatureIDE · 19. Jan.):

You want to create feature models and derive valid configurations? You are tired of the time-consuming set-up of Eclipse to do that? Since 2017, we release every @FeatureIDE version as pre-packaged Eclipse for Windows, Mac, and Linux. featureide.github.io/#download

Below the tweet, there's a link to "Original (Englisch) übersetzen". The "Medien" tab shows two screenshots of the FeatureIDE interface.

On the right sidebar, there's a "Wem folgen?" section with two users: Jörg Liebig (@joliebiq) and Norbert Siegmund (@Nor...), each with a "Folgen" button. Below that is a "Finde Leute, die du kennst" section. At the bottom, there's a "Trends für dich" section with hashtags like #SO4WOB, #sgem05, #GroKo, #bachelor, and #Heimatministerium.

Enjoy FeatureIDE on Youtube

YouTube DE Suchen

FeatureIDE 44 Abonnenten KANAL ANPASSEN CREATOR STUDIO

ÜBERSICHT

Uploads Öffentlich ALLE WIEDERGEBEN

VariantSync: Change Synchronization from 14 Aufrufe • vor 1 Monat

VariantSync: Change Synchronization into Multiple 5 Aufrufe • vor 1 Monat

VariantSync: Recording Feature Mappings During 17 Aufrufe • vor 1 Monat

VariantSync: Setting-Up Legacy Clones for 13 Aufrufe • vor 1 Monat

VariantSync: Automating the Synchronization of Software 40 Aufrufe • vor 1 Monat

Eigene Playlists Öffentlich

VariantSync 5

Team Project @ TU Braunschweig (October) 4

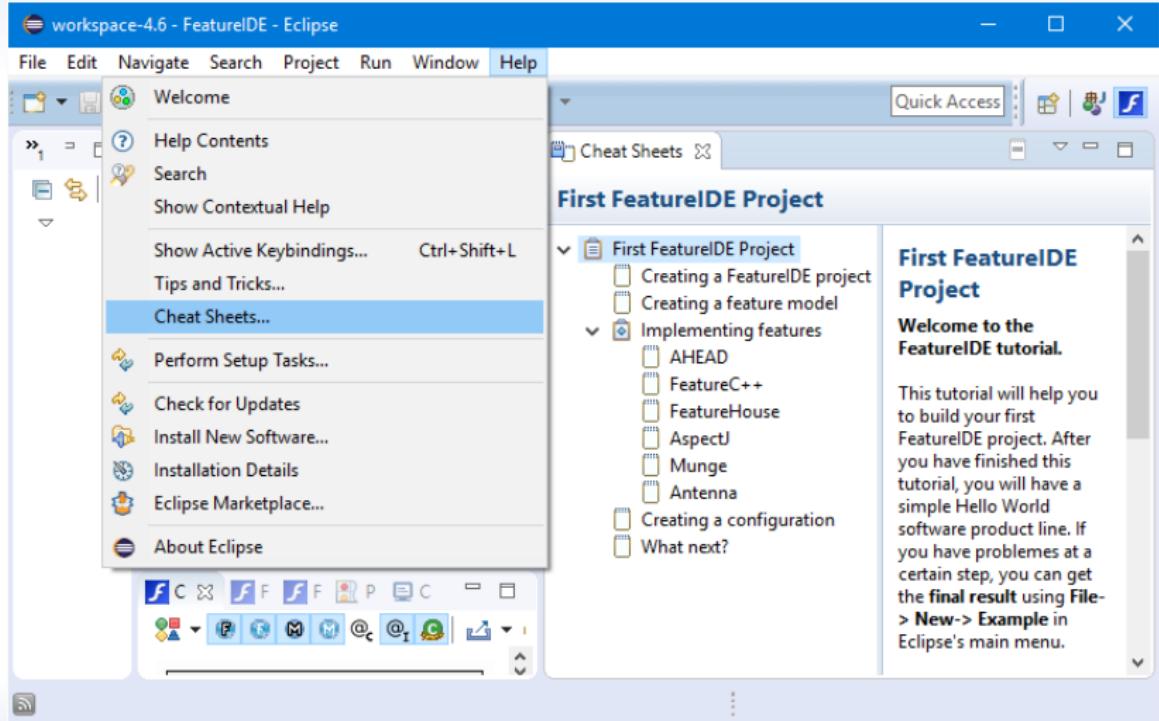
Software Projects @ University of Magdeburg 23

Awards 2

Master Projects @ TU Braunschweig 1

youtube.com/watch?v=ztpdk-mkpzM

Learn more with FeatureIDE cheat sheet



Feedback and discussion

1. What did you learn?
2. What did you miss?

Keep it short (2-3 sentences)



FeatureIDE Tutorial

