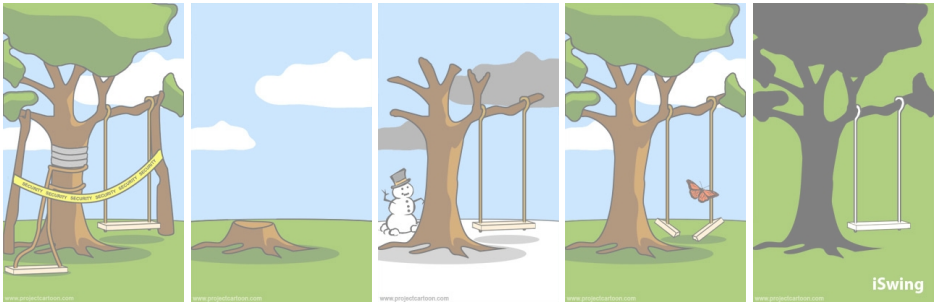
Software Engineering

15. Software Product Lines | Thomas Thüm | May 31, 2022

Software Product Lines (Software-Produktlinien)



how patches were applied	how it was supported	when it was delivered	how it performed under load	what marketing advertised
software evolution	**software maintenance**	**configuration management**	**compilation and static analyses**	**software product lines**

Lessons Learned

- Test-driven development
- Examples for type, runtime, and logical errors
- Examples for JUnit and assertions
- Design by contract: class invariants, preconditions, postconditions
- Blame assignment, behavioral subtyping
- Java Modeling Language

Practice

- Quiz with examples for parse error, type errors, runtime errors, logical errors
- Post questions to Moodle, if any

Lecture Overview

1. Software Product Lines and Feature Modeling
2. Implementation of Software Product Lines
3. Testing of Software Product Lines

Lecture Contents

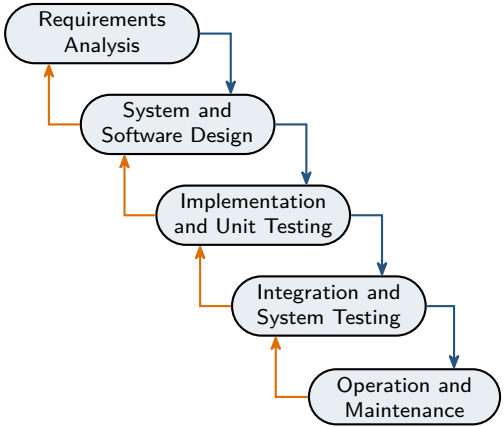
Greenfield Development? (Auf der grünen Wiese?)

Who Produces Only One Product?

Recap: Waterfall Model (Wasserfallmodell)

Waterfall Model [Sommerville]

- first process model, motivated by practice
- by Winston W. Royce 1970
- each development phase ends by the approval of one or more documents (document-driven process model)
- phases do not overlap
- numerous variants with varying number of phases: 5–7
- here: simplified variant by Sommerville



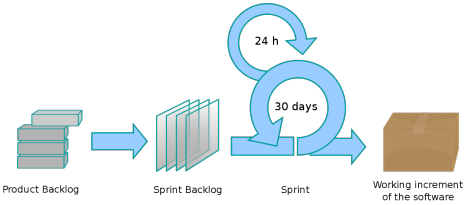
Recap: Scrum

User Story [Sommerville]

- a scenario of use that might be experienced by a system user
- aka. **story card** as user stories are sometimes written on physical cards
- user stories are prioritized by the customer
- subset of all user stories is chosen for the next iteration
- used in many agile methods

Scrum [Sommerville]

- an agile method, most-widely used method
- no special development techniques (like pair programming, test-driven development)
- **product backlog**: list of user stories, collected an priorized by the **product owner**
- **sprint backlog**: user stories selected by the scrum team for the next **sprint**



Software Product Lines [Apel et al. 2013]

Mass Customization

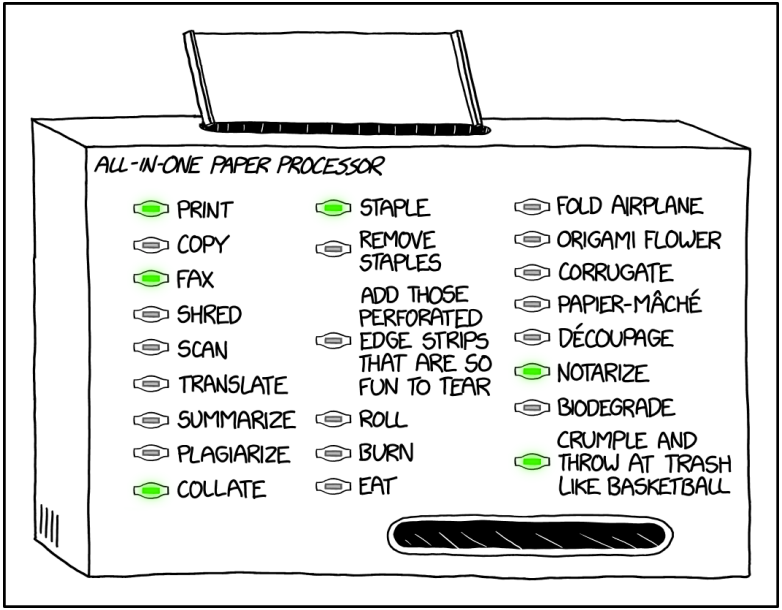
- opposed to mass production
- opposed to individual customization
- idea: produce customized goods as efficient as with mass production

Software Product Line

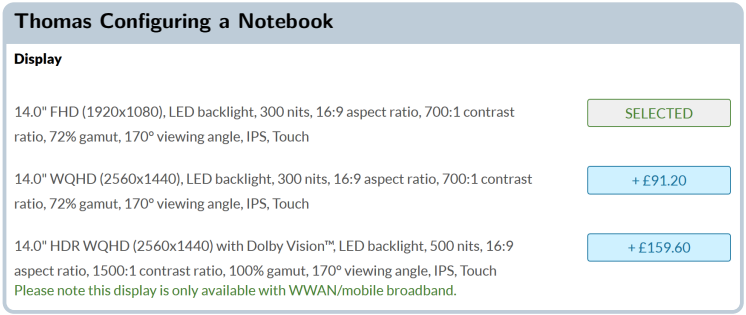
- set of software-intensive systems (aka. products)
- sharing a common, managed set of features
- satisfying the needs of a domain
- developed from a common set of core assets (reuse)

Feature

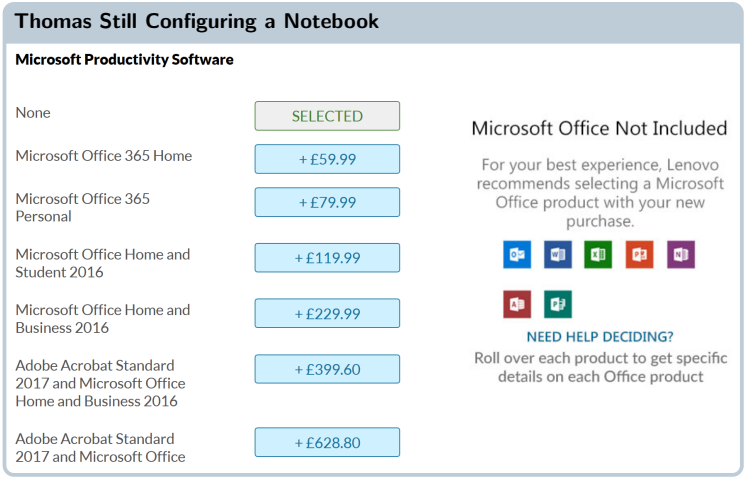
- domain abstraction
- used for communication by stakeholders (e.g., manager, developer, tester, customer, marketing)
- specifies differences between products


ALL-IN-ONE PAPER PROCESSOR
PRINT
COPY
FAX
SHRED
SCAN
TRANSLATE
SUMMARIZE
PLAGIARIZE
COLLATE
STAPLE
REMOVE STAPLES
ADD THOSE PERFORATED EDGE STRIPS THAT ARE SO FUN TO TEAR
ROLL
BURN
EAT
FOLD AIRPLANE
ORIGAMI FLOWER
CORRUGATE
PAPIER-MÂCHÉ
DÉCOUPAGE
NOTARIZE
BIODEGRADE
CRUMPLE AND THROW AT TRASH LIKE BASKETBALL

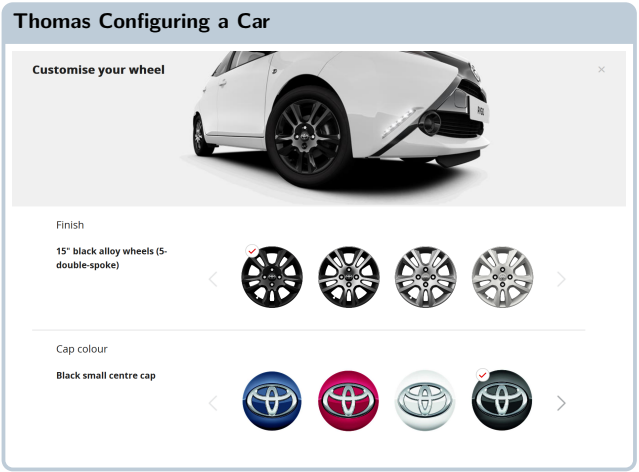
Dependencies Between Features



Dependencies Between Features

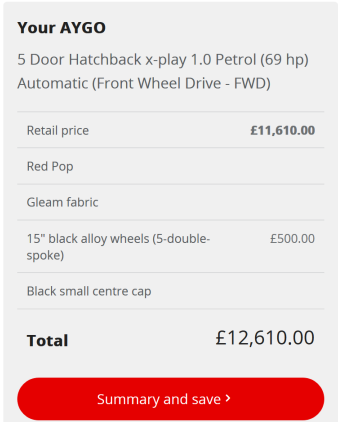


Dependencies Between Features

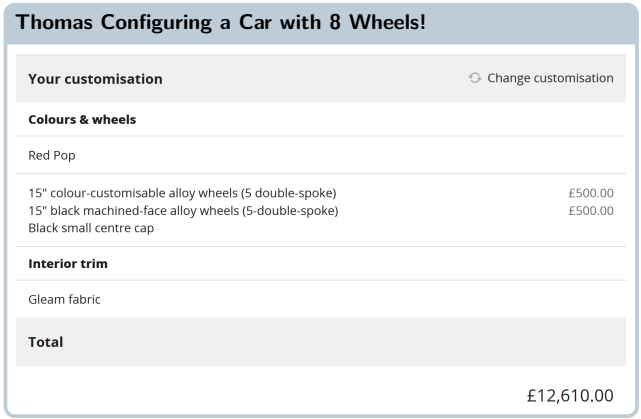


Dependencies Between Features

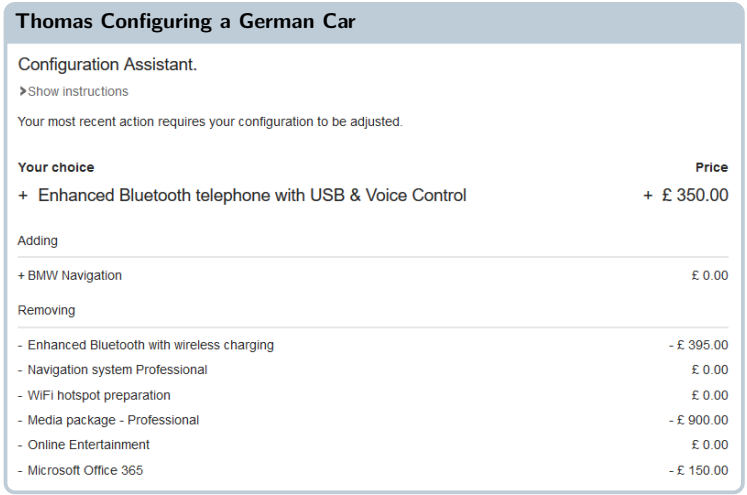
Thomas Configuring a Car with a Weird Price



Dependencies Between Features



Dependencies Between Features



Thomas Configuring a German Car

Configuration Assistant.

Show instructions

Your most recent action requires your configuration to be adjusted.

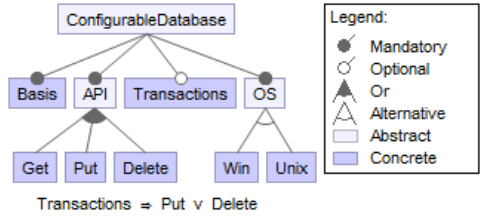
Your choice	Price
+ Enhanced Bluetooth telephone with USB & Voice Control	+ £ 350.00
Adding	
+ BMW Navigation	£ 0.00
Removing	
- Enhanced Bluetooth with wireless charging	- £ 395.00
- Navigation system Professional	£ 0.00
- WiFi hotspot preparation	£ 0.00
- Media package - Professional	- £ 900.00
- Online Entertainment	£ 0.00
- Microsoft Office 365	- £ 150.00

Feature Modeling [Apel et al. 2013]

Feature Model

- hierarchy of features
- dependencies between features modeled by tree and cross-tree constraints
- **tree constraints**: defined by the hierarchy
- **cross-tree constraints**: propositional formulas over features
- **concrete features** have an implementation
- **abstract features** are used to group other features

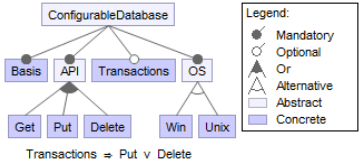
A Database Management Product Line



Tree Constraints

- each feature requires its parent
- an **optional feature** has no further constraints
- a **mandatory feature** is required by its parent
- **or group**: at least one feature must be selected when the parent is selected
- **alternative group**: exactly one feature must be selected when the parent is selected

Enumerating All Configurations

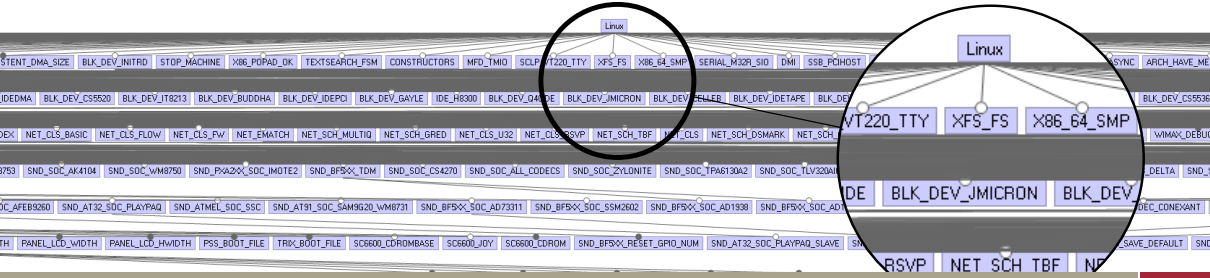


26 Valid Configurations

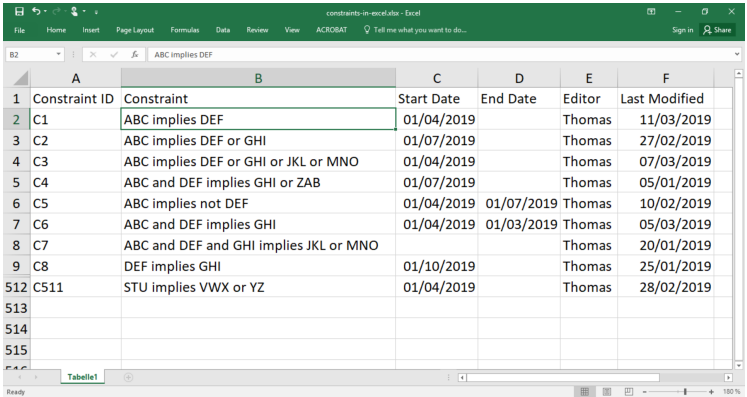
$\{B, G, W\}$
$\{B, P, W\}$
$\{B, G, P, W\}$
$\{B, D, W\}$
$\{B, G, D, W\}$
$\{B, P, D, W\}$
$\{B, G, P, D, W\}$
$\{B, P, T, W\}$
$\{B, G, P, T, W\}$
$\{B, D, T, W\}$
$\{B, G, D, T, W\}$
$\{B, P, D, T, W\}$
$\{B, G, P, D, T, W\}$

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

Linux Feature Model



Dependencies Modeled in Excel



	A	B	C	D	E	F
1	Constraint ID	Constraint	Start Date	End Date	Editor	Last Modified
2	C1	ABC implies DEF	01/04/2019		Thomas	11/03/2019
3	C2	ABC implies DEF or GHI	01/07/2019		Thomas	27/02/2019
4	C3	ABC implies DEF or GHI or JKL or MNO	01/04/2019		Thomas	07/03/2019
5	C4	ABC and DEF implies GHI or ZAB	01/07/2019		Thomas	05/01/2019
6	C5	ABC implies not DEF	01/04/2019	01/07/2019	Thomas	10/02/2019
7	C6	ABC and DEF implies GHI	01/04/2019	01/03/2019	Thomas	05/03/2019
8	C7	ABC and DEF and GHI implies JKL or MNO			Thomas	20/01/2019
9	C8	DEF implies GHI	01/10/2019		Thomas	25/01/2019
512	C511	STU implies VWX or YZ	01/04/2019		Thomas	28/02/2019
513						
514						
515						

Software Product Lines and Feature Modeling

Lessons Learned

- Motivation for mass customization
- Software product lines, features, dependencies between features, configurators
- Feature models: abstract and concrete features, tree and cross-tree constraints
- Tree constraints: optional, mandatory, or group, alternative group
- Feature modeling in practice
- Further Reading: Apel et al. 2013, Chapter 1 Software Product Lines and Section 2.3 Feature Modeling

Practice

- Sketch a feature model that has 9 valid configurations (pen and paper preferred)
- Upload your feature model to Moodle
- Inspect whether other submitted solutions are syntactically correct and have the right number of configurations

Lecture Contents

1. Software Product Lines and Feature Modeling

2. Implementation of Software Product Lines
   - The Value of Feature Modeling
   - Recap: Branching & Merging
   - Runtime Parameters
   - Runtime Parameters and Runtime Properties
   - The C Preprocessor
   - Munge – A Preprocessor for Java
   - Antenna – An In-Place Preprocessor
   - Discussion of Preprocessors
   - Frameworks with Plug-Ins
   - Lessons Learned

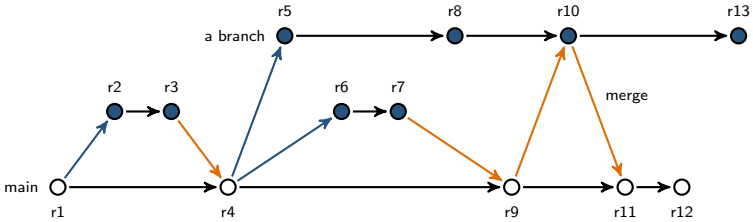
3. Testing of Software Product Lines

The Value of Feature Modeling

Interview with Practitioners [Berger et al. 2014]

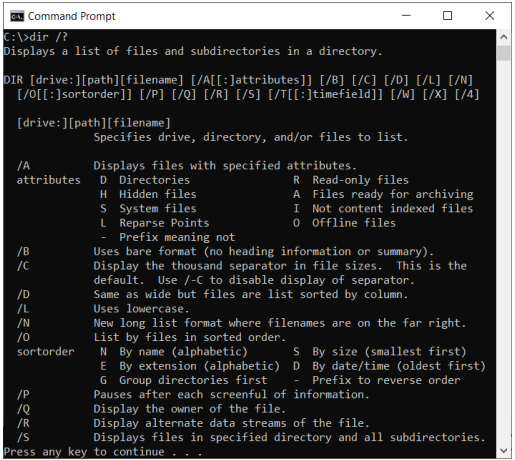
"I think the best [about feature modeling] is you can see relationships, to actually know what configurations are allowed and what are not allowed. That was also not so easy to express in the past [...] This is from the developer's point of view. But it's also, we can see that from the, say project development, it's also important, because before we noticed that **the same functionality was implemented twice** within the same project, basically they haven't realized that. They implemented the same features."

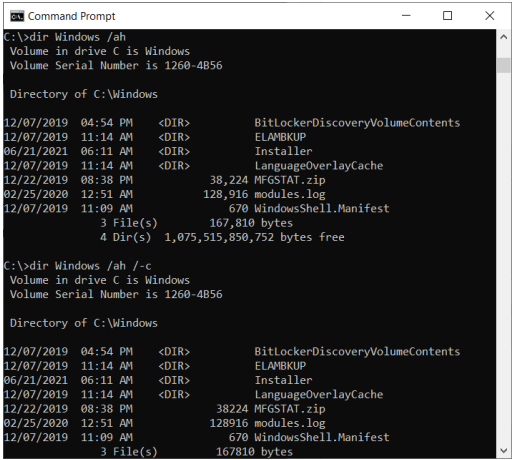
Recap: Branching & Merging



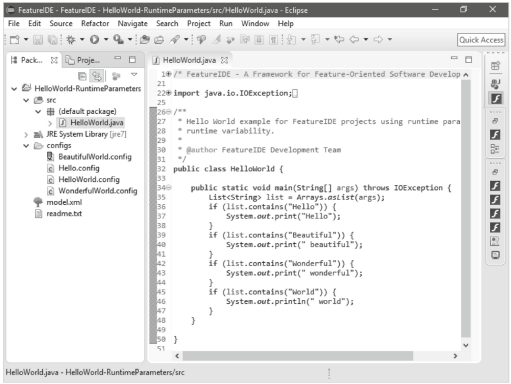
- simultaneous, independent development
- option to merge in the future
- `master/` – main development
- `branch/*/` – parallel developments

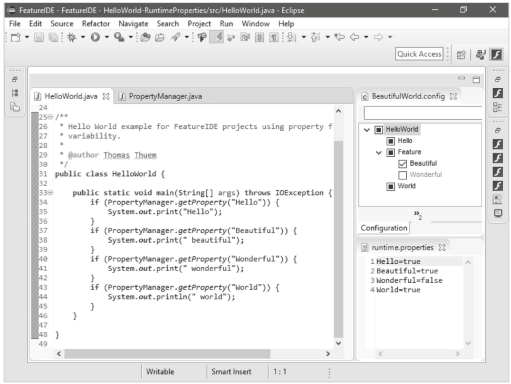
Runtime Parameters





Runtime Parameters and Runtime Properties [Meinicke et al. 2013]






IT TOOK A LOT OF WORK, BUT THIS LATEST LINUX PATCH ENABLES SUPPORT FOR MACHINES WITH 4,096 CPUs, UP FROM THE OLD LIMIT OF 1,024.
DO YOU HAVE SUPPORT FOR SMOOTH FULL-SCREEN FLASH VIDEO YET?
NO, BUT WHO USES THAT?

The C Preprocessor [Meinicke et al. 2013]

Example Input to the Preprocessor

```
1  #include <iostream>
2
3  #define Hello true
4  #define Beautiful true
5  #define Wonderful false
6  #define World true
7
8  int main() {
9    ::std::cout
10   #if Hello
11     << "Hello "
12   #endif
13   #if Beautiful
14     << "beautiful "
15   #endif
16   #if Wonderful
17     << "wonderful ";
18   #endif
19   #if World
20     << "world!"
21   #endif
22     << std::endl;
23 }
```

Example Output (Simplified)

```
1 int main() {
2   ::std::cout
3     << "Hello "
4     << "Beautiful "
5     << "World!"
6     << std::endl;
7 }
```

Munge – A Preprocessor for Java [Meinicke et al. 2013]

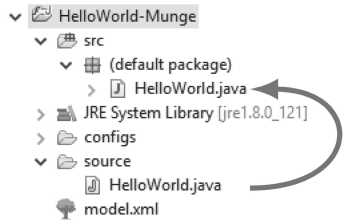
Example Input and Output

```
1  public class Main {
2    public static void main(String[]
         args) {
3      /*if[Hello]*/
4      System.out.print("Hello");
5      /*end[Hello]*/
6      /*if[Beautiful]*/
7      System.out.print(" beautiful");
8      /*end[Beautiful]*/
9      /*if[Wonderful]*/
10     System.out.print(" wonderful");
11     /*end[Wonderful]*/
12     /*if[World]*/
13     System.out.print(" world!");
14     /*end[World]*/
15   }
16 }
```

```
public class Main {
  public static void main(String[]
      args) {

    System.out.print("Hello");

    System.out.print(" beautiful");

    System.out.print(" world!");

  }
}
```

Calling the Preprocessor

```
Munge -DHello -DBeautiful -DWorld Main.java targetDir
```



Antenna – An In-Place Preprocessor [Meinicke et al. 2013]

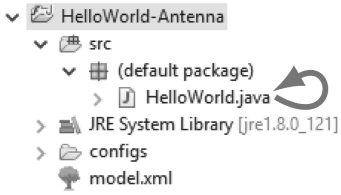
Example Input and Output

```
1  public class Main {
2    public static void main(String[]
         args) {
3      //#if Hello
4      System.out.print("Hello");
5      //#endif
6      //#if Beautiful
7      System.out.print(" beautiful");
8      //#endif
9      //#if Wonderful
10 //@ System.out.print(" wonderful");
11     //#endif
12     //#if World
13     System.out.print(" world!");
14     //#endif
15   }
16 }
```

```
public class Main {
  public static void main(String[]
       args) {
    //#if Hello
    System.out.print("Hello");
    //#endif
    //#if Beautiful
//@  System.out.print(" beautiful");
    //#endif
    //#if Wonderful
    System.out.print(" wonderful");
    //#endif
    //#if World
    System.out.print(" world!");
    //#endif
  }
}
```

Calling the Preprocessor

```
java Antenna Main.java Hello,World,Beautiful
```

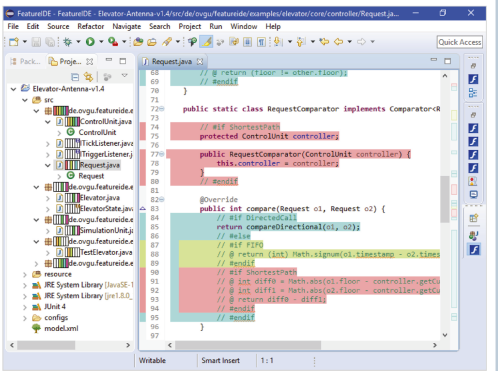


Discussion of Preprocessors [Meinicke et al. 2013]

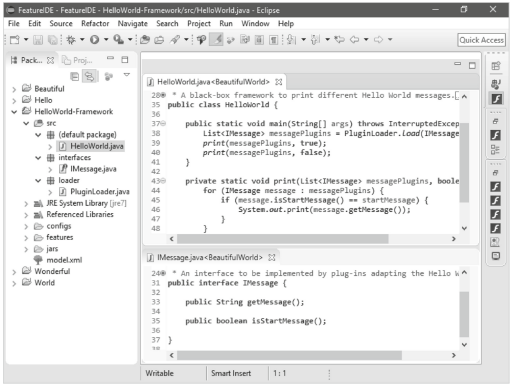
A Slightly More Complex Example

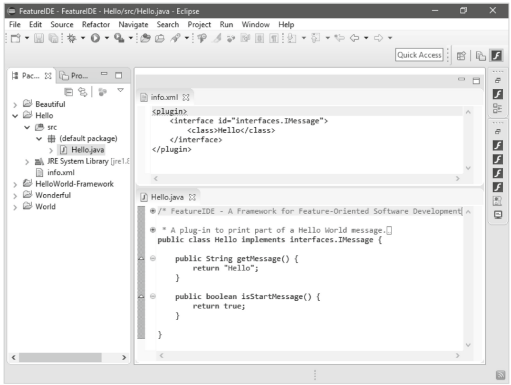
```
1  public class SimulationUnit {
2    public static void main(String[] args) {
3      SimulationUnit sim = new SimulationUnit();
4      //#if FIFO | Service
5      simulationWindow = new MainWindow(sim);
6      //#else
7      //@ simulationWindow = new MainWindow();
8      //#endif
9      sim.start(5);
10   }
11
12   //#if FIFO
13   public void floorRequest(Request floorRequest) {
14     controller.trigger(floorRequest);
15   }
16
17   public int getCurrentFloor() {
18     return controller.getCurrentFloor();
19   }
20   //#endif
21
22   // further source code
23 }
```

Tool Support for Feature Traceability



Frameworks with Plug-Ins [Meinicke et al. 2013]





Implementation of Software Product Lines

Lessons Learned

- Implementation of software product lines
- Branches, runtime parameters
- Preprocessors: in-place or not, feature traceability
- Frameworks with plug-ins
- Further Reading: Meinicke et al. 2013 Chapter 9 Conditional Compilation with FeatureIDE and Chapter 17 Tool Support Beyond Preprocessors and Feature Modules

Practice

- Quiz'n'Disquiz
- Quiz: fill out the Moodle quiz on your own
- Disquiz: compare and discuss the results with your colleagues

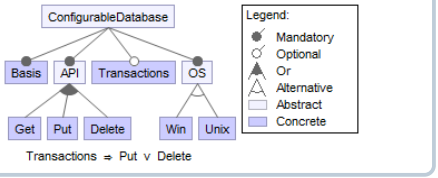
Lecture Contents

1. Software Product Lines and Feature Modeling

2. Implementation of Software Product Lines

3. Testing of Software Product Lines
   - Testing All Configurations
   - Testing One Configuration
   - Pairwise Interaction Testing
   - Pairwise Coverage
   - T-Wise Interaction Testing
   - Combinatorial Interaction Testing with ICPL
   - Recap: Linux Feature Model
   - Pairwise Interaction Testing in Practice
   - Choose Features Wisely
   - Number of Features in Linux
   - Lessons Learned

Testing All Configurations



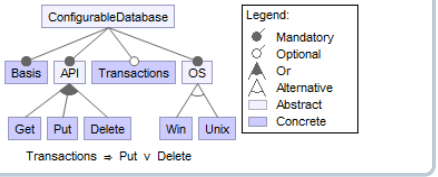
26 Valid Configurations

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

Discussion

- only feasible for small product lines (few valid configurtions)
- redundant test effort
- large product lines: not even feasible to generate and compile all configurations
- (some) large product lines: not even the number of valid configurations is known

Testing One Configuration



Which Valid Configuration to Test?

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

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

Discussion

- applicable to large product lines
- no redundant test effort (from configurations)
- often not feasible to test all features in one configuration (e.g., Win and Unix)
- unnoticed feature interactions

Feature Interactions

- features work well in isolation, but not in combination
- example: fire control and flood control in a building
- **pairwise interactions**: interaction of two features
- **t-wise interaction**: interaction of t features

Pairwise Interaction Testing

Pairwise Interaction Testing

- test a sample set $S \subseteq C$ of all valid configurations C with pairwise coverage
- every pairwise interaction is covered by at least one configuration in the sample S

Configurations with the Interaction Get $\wedge$ Put

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

Discussion

- applicable to large product lines
- reduced redundant effort compared to testing all configurations
- coverage guarantee (opposed to random configurations)
- still requires good test cases (program inputs)
- hard to compute small sample sets

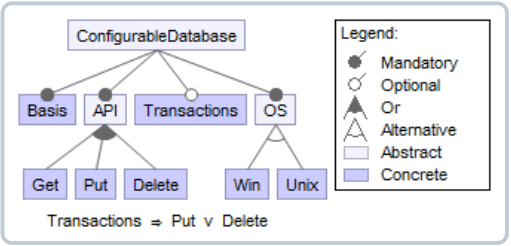
Pairwise Interactions

- **up-to** four interactions between A and B
- both selected: $A \wedge B$
- one selected: $\neg A \wedge B$, $A \wedge \neg B$
- none selected: $\neg A \wedge \neg B$

Pairwise Coverage

Interactions to Cover

- exclude invalid combinations (e.g., $W \wedge U$)
- exclude abstract features (e.g., API)
- exclude features contained in every configuration (e.g., B)



Pairwise Interactions

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

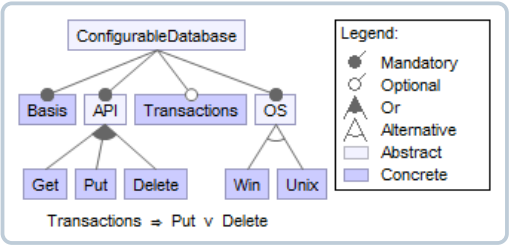
Pairwise Coverage with Six Configurations

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

Pairwise Coverage

Interactions to Cover

- exclude invalid combinations (e.g., $W \wedge U$)
- exclude abstract features (e.g., API)
- exclude features contained in every configuration (e.g., B)



Pairwise Interactions

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

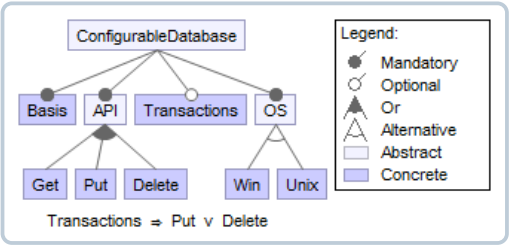
Pairwise Coverage with Six Configurations

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

Pairwise Coverage

Interactions to Cover

- exclude invalid combinations (e.g., $W \wedge U$)
- exclude abstract features (e.g., API)
- exclude features contained in every configuration (e.g., B)



Pairwise Interactions

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

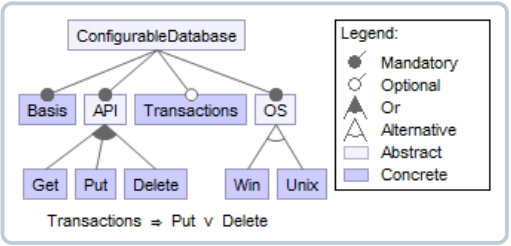
Pairwise Coverage with Six Configurations

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

Pairwise Coverage

Interactions to Cover

- exclude invalid combinations (e.g., $W \wedge U$)
- exclude abstract features (e.g., *API*)
- exclude features contained in every configuration (e.g., B)



Pairwise Interactions

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

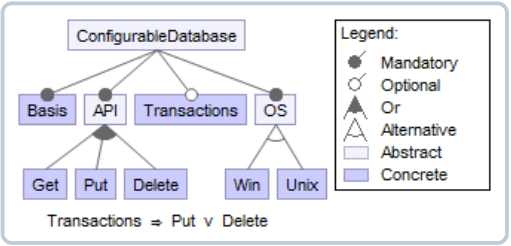
Pairwise Coverage with Six Configurations

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

Pairwise Coverage

Interactions to Cover

- exclude invalid combinations (e.g., $W \wedge U$)
- exclude abstract features (e.g., *API*)
- exclude features contained in every configuration (e.g., B)



Pairwise Interactions

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

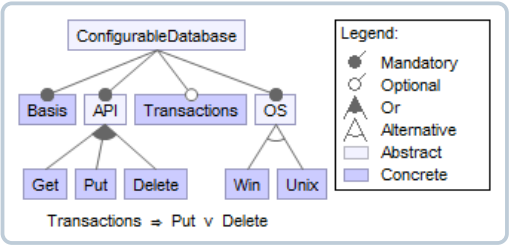
Pairwise Coverage with Six Configurations

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

Pairwise Coverage

Interactions to Cover

- exclude invalid combinations (e.g., $W \wedge U$)
- exclude abstract features (e.g., API)
- exclude features contained in every configuration (e.g., B)



Pairwise Interactions

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

Pairwise Coverage with Six Configurations

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

T-Wise Interaction Testing

T-Wise Interaction Testing

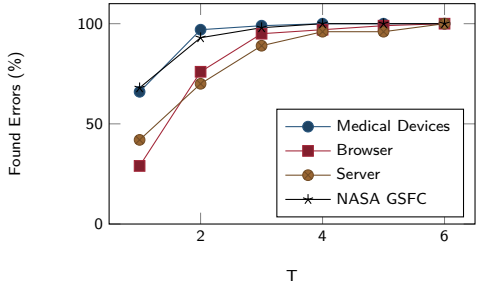
- generalization of pairwise interaction testing
- t-wise coverage: every t-wise interaction is covered by at least one configuration in the sample
- t=1: every feature is selected and also deselected
- t=2: pairwise interaction coverage
- t=3: every combination of three features

T=3 Interactions for the Features G, P, and D

$G \wedge P \wedge D$
$G \wedge P \wedge \neg D$
$G \wedge \neg P \wedge D$
$G \wedge \neg P \wedge \neg D$
$\neg G \wedge P \wedge D$
$\neg G \wedge \neg P \wedge D$
$\neg G \wedge \neg P \wedge \neg D$

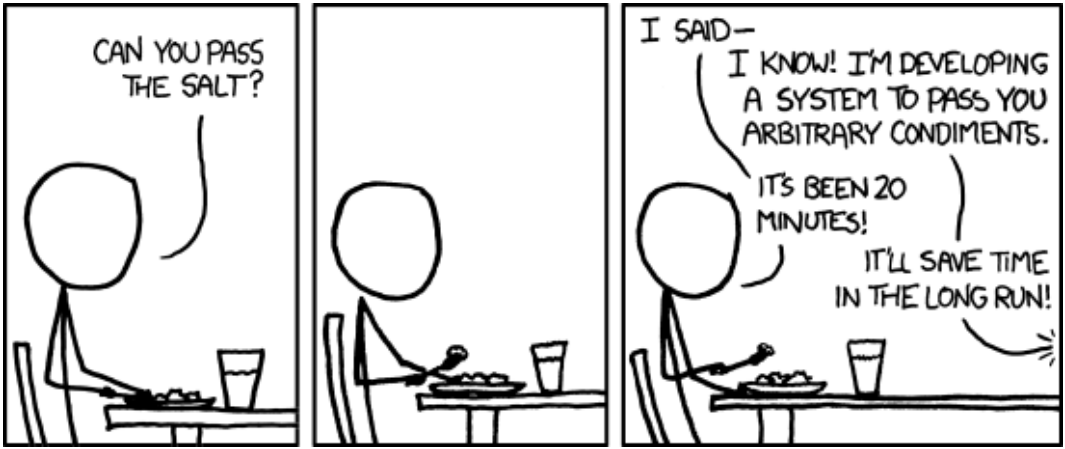
Effectivity of Interaction Testing

[Kuhn et al. 2004]



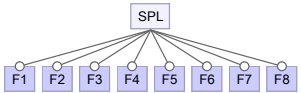
Trade-Off

large t: high coverage (more effective)
small t: low testing effort (more efficient)

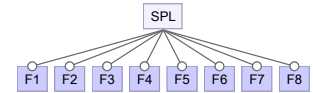

CAN YOU PASS THE SALT?
I SAID—
I KNOW! I'M DEVELOPING A SYSTEM TO PASS YOU ARBITRARY CONDIMENTS.
IT'S BEEN 20 MINUTES!
IT'LL SAVE TIME IN THE LONG RUN!

Combinatorial Interaction Testing with ICPL [Johansen et al. 2012]

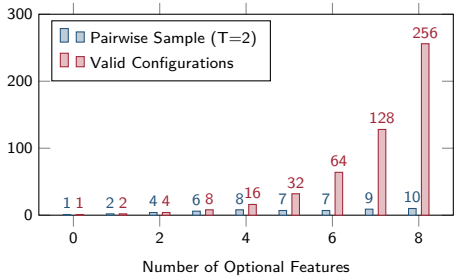
Assumption: All Features are Optional



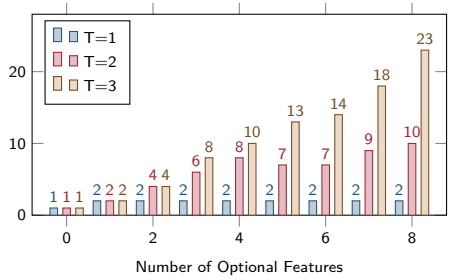
Assumption: All Features are Optional



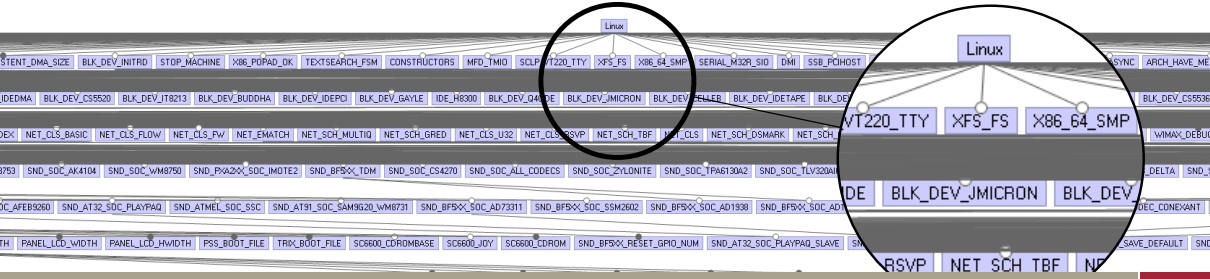
Number of Configurations in Pairwise Sample



Number of Configurations in T-Wise Sample

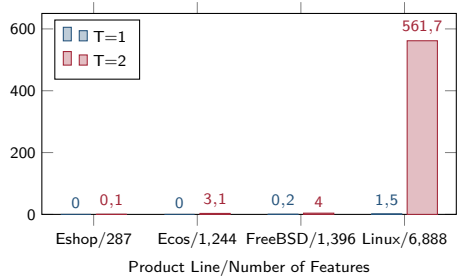


Recap: Linux Feature Model



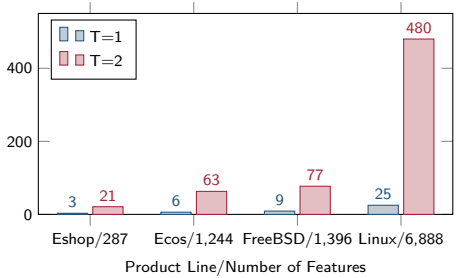
Pairwise Interaction Testing in Practice [Johansen et al. 2012]

Time in Minutes to Compute Sample



- about 9h for Linux
- 480 configuration in pairwise sample

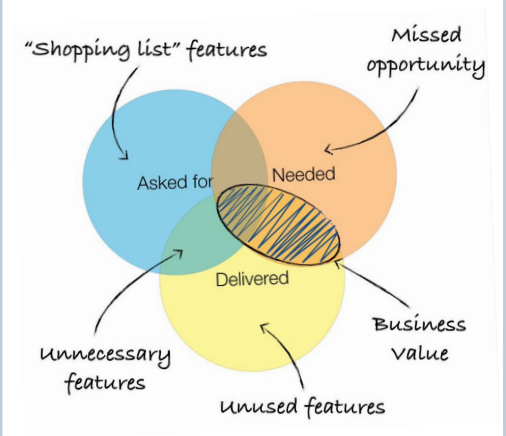
Number of Configurations in Sample



- Linux kernel v2.6.28.6 (February 2009)
- 6,888 features, 187,193 clauses in conjunctive normal form

Choose Features Wisely

John Ferguson Smart (2017)

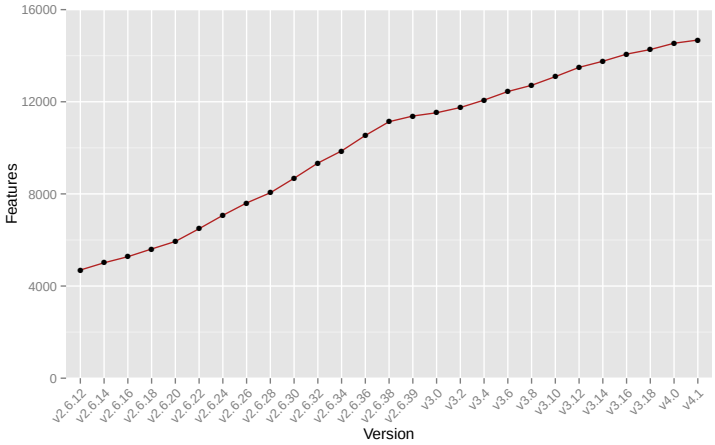


John Carmack (born 1970) [uci.edu]

"The important point is that the cost of adding a feature isn't just the time it takes to code it. The cost also includes the addition of an obstacle to future expansion. [...] The trick is to pick the features that don't fight each other."

Number of Features in Linux [Hengelein 2015]

2005–2015: Number of Features Tripled



Testing of Software Product Lines

Lessons Learned

- Testing software product lines
- All configurations? one configuration? feature interactions!
- Combinatorial interaction testing: pairwise and t-wise
- Sample coverage (effectiveness) vs sample size (testing efficiency) vs time to compute the sample (sampling efficiency)
- Further Reading: Johansen et al. 2012

Practice

- Quiz on the complete product-line lecture
- Post questions to Moodle, if any