



VERification-oriented & component-based model Driven Engineering for real-time embedded systems













Properties of Realistic Feature Models make Combinatorial Testing of Product Lines Feasible

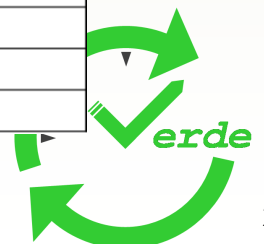
Martin F. Johansen^{1,2}, Øystein Haugen¹ and Franck Fleurey¹

¹SINTEF ICT, Oslo, Norway

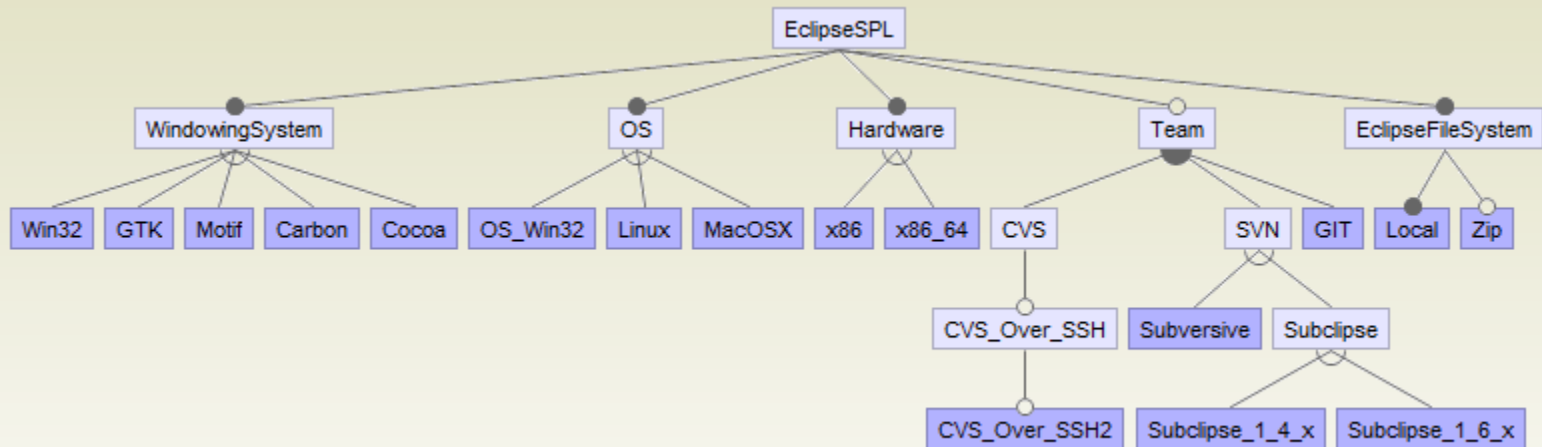
²Institute for informatics, University of Oslo, Norway

Example Product Line: Eclipse

	 Java	 Java EE	 C/C++	 C/C++ Linux	 RCP/RAP	 Modeling	 Reporting	 Parallel	 Scout	 Testers	 Javascript	 Classic
RCP/Platform	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
CVS	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
EGit			✓	✓	✓	✓						
EMF	✓	✓				✓	✓					
GEF	✓	✓				✓	✓					
JDT	✓	✓			✓	✓	✓		✓			✓
Mylyn	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	
Web Tools		✓					✓				✓	
Linux Tools			✓	✓				✓				
Java EE Tools		✓					✓					
XML Tools	✓	✓			✓		✓	✓				
RSE		✓	✓	✓			✓	✓				
EclipseLink		✓					✓			✓		
PDE		✓			✓	✓	✓		✓			✓
Datatools		✓					✓					
CDT			✓	✓				✓				
BIRT							✓					
GMF						✓						
PTP								✓				
MDT						✓						
Scout									✓			
Jubula										✓		
RAP					✓							
WindowBuilder	✓											
Maven	✓											



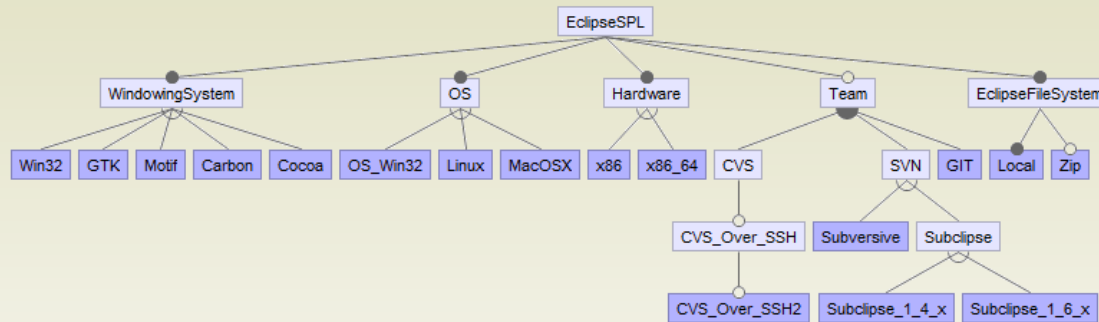
Example: Feature model



$\text{Carbon} \wedge \text{MacOSX} \wedge \text{x86} \vee \text{Cocoa} \wedge \text{MacOSX} \wedge (\text{x86} \vee \text{x86_64}) \vee \text{GTK} \wedge \text{Linux} \wedge (\text{x86} \vee \text{x86_64}) \vee \text{Motif} \wedge \text{Linux} \wedge \text{x86} \vee \text{Win32} \wedge \text{OS_Win32} \wedge (\text{x86} \vee \text{x86_64})$

Basic Question of SPL Testing

- How do we gain confidence in that any valid product will work?



Carbon \wedge MacOSX \wedge x86 \vee Cocoa \wedge MacOSX \wedge (x86 \vee x86_64) \vee GTK \wedge Linux \wedge (x86 \vee x86_64) \vee Motif \wedge Linux \wedge x86 \vee Win32 \wedge OS_Win32 \wedge (x86 \vee x86_64)

	Java	Java EE	CC++ Linux	CC++ Linux	RCPIRAP	Modeling	JEE BIRT Reporting	Parallel	Scout	Testers	Javascript	Classic
RCPPlatform	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
CVS	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
EGit			✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
EMF	✓	✓					✓					
GEF	✓	✓					✓					
JDT	✓	✓			✓		✓			✓		✓
Mylyn	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	
Web Tools	✓	✓					✓					✓
Linux Tools			✓	✓			✓					
Java EE Tools	✓	✓					✓					
XML Tools	✓	✓			✓		✓	✓				
RSE		✓	✓	✓			✓	✓				
EclipseLink	✓	✓					✓			✓		
PDE	✓				✓	✓	✓		✓			✓
Datatools	✓						✓					
CDT			✓	✓				✓				
BIRT							✓					
GMF							✓					
PTP								✓				
MDT							✓					
Scout									✓			
Jubula										✓		
RAP												
WindowBuilder	✓											
Maven	✓											

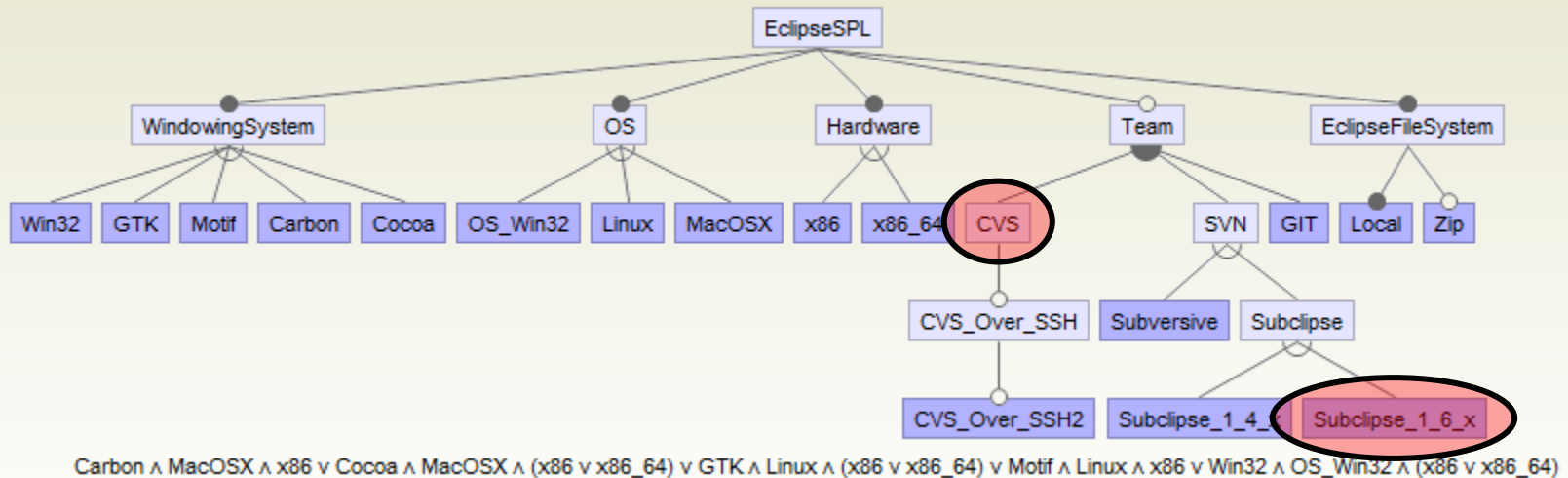


Many suggestions

- Generally, not done in industry
- Problems of scalability
 - ScenTED and PLUTO
 - Variability stereotyped onto a UML model.
 - UML model becomes large (it is the union of all products)
 - How to generate the tests is still unsolved.
 - Incremental approach
 - Differences between products modeled precisely.
 - Involves intractable formal analysis.
 - Large, unsolved problem in informatics.
 - Combinatorial Interaction Testing (CIT)
 - ...

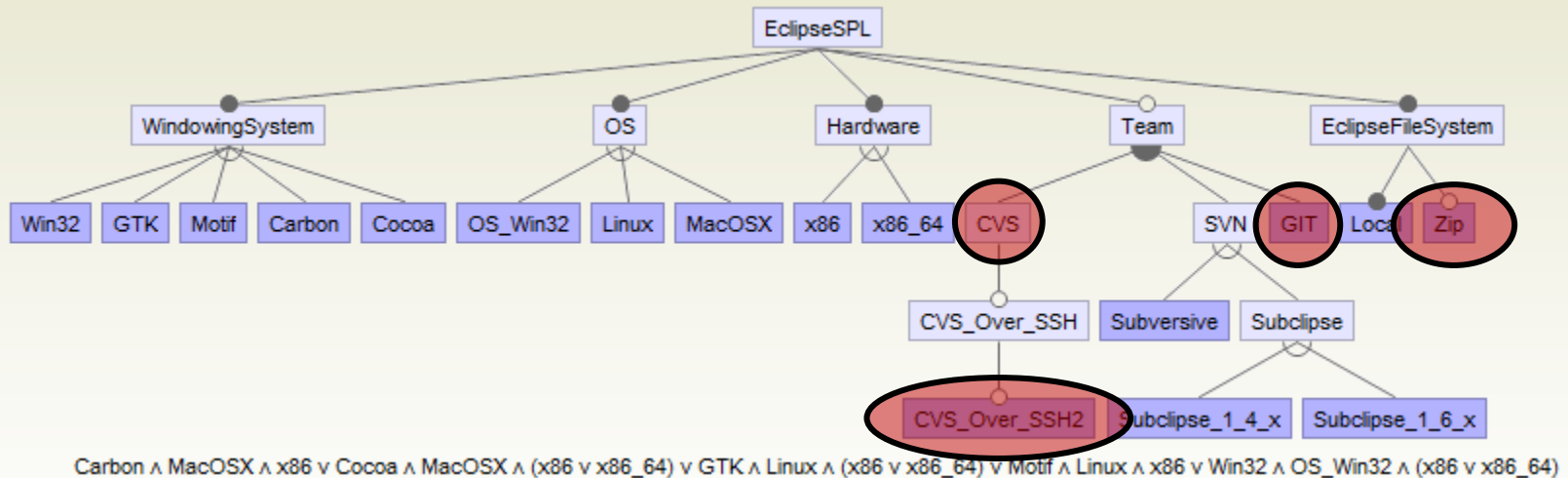
Interaction faults

- A bug that occurs when 2 features are in a product
 - 2-wise interaction fault
- The other do not matter



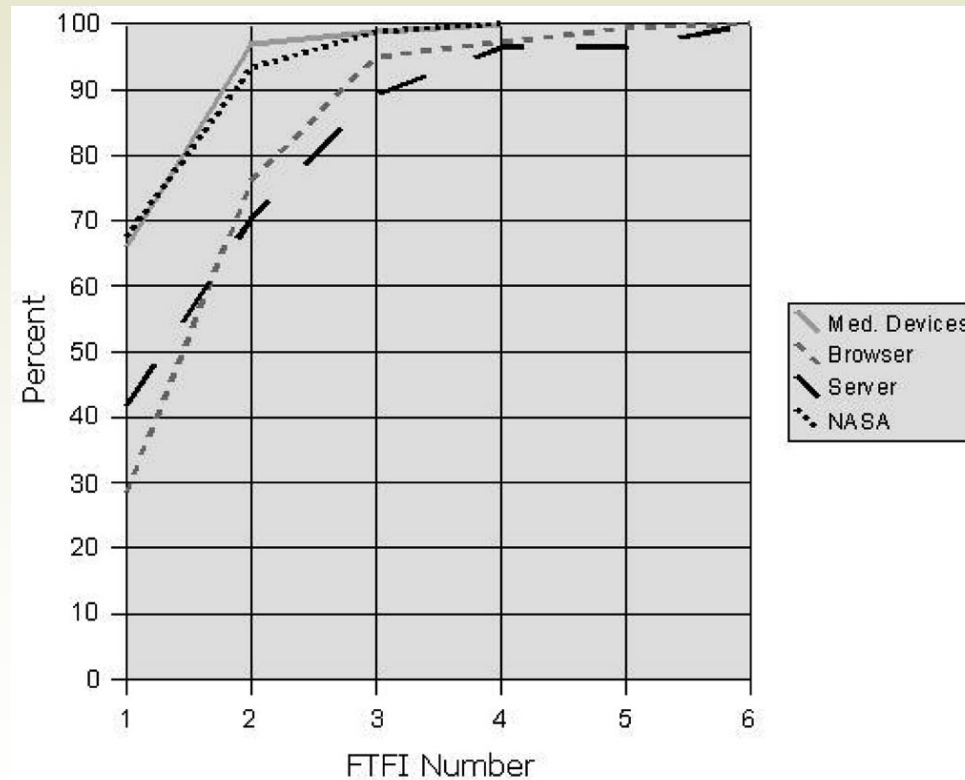
Interaction faults

- A bug that occurs when 4 features are in a product
 - 4-wise interaction fault
- The other do not matter



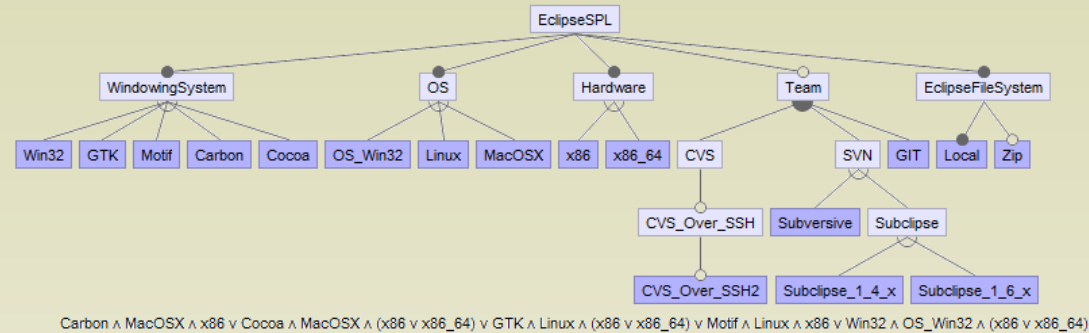
Empirical basis for CIT

- Kuhn et al. 2004:
 - Most bugs can be attributed to the interaction of a few features.



Combinatorial Interaction Testing (CIT)

- Produce a covering array
 - The valid products that includes all interactions between e.g. 2 features.
 - Generating such an array is regarded as intractable (takes too long).
- Apply a single system testing technique to each product



Carbon \wedge MacOSX \wedge x86 \vee Cocoa \wedge MacOSX \wedge (x86 \vee x86_64) \vee GTK \wedge Linux \wedge (x86 \vee x86_64) \vee Motif \wedge Linux \wedge x86 \vee Win32 \wedge OS_Win32 \wedge (x86 \vee x86_64)

Feature \ Product	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
EclipseSPL	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
WindowingSystem	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Win32	-	-	X	-	-	X	-	-	X	-	-	-	-	X	-	-	-	-	-	-	-	X
GTK	-	X	-	-	-	X	-	-	X	-	-	-	X	-	-	X	-	-	-	-	-	-
Motif	-	-	-	X	-	-	X	-	-	X	-	-	-	-	-	-	-	X	-	-	-	-
Carbon	-	-	X	-	-	-	-	-	X	-	-	-	-	X	-	X	-	-	-	-	-	-
Cocoa	X	-	-	-	-	-	-	-	-	-	X	-	-	-	-	-	-	-	X	X	-	-
OS	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
OS_Win32	-	-	X	-	X	-	X	-	-	-	-	-	X	-	-	-	-	-	-	-	-	X
Linux	-	X	-	-	X	-	X	X	-	X	-	X	-	X	-	-	X	-	X	-	-	-
MacOSX	X	-	X	-	-	-	-	-	X	-	X	-	X	-	X	-	X	-	X	X	-	-
Hardware	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
x86	X	-	X	X	-	X	-	X	-	X	X	-	X	X	-	X	X	-	X	X	-	-
x86_64	-	X	X	-	-	X	X	-	X	-	-	X	X	-	-	X	-	-	X	X	-	X
Team	-	-	-	-	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
CVS	-	-	-	-	-	-	-	-	-	-	-	-	-	-	X	X	X	X	X	X	X	X
CVS_Over_SSH	-	-	-	-	-	-	-	-	-	-	-	-	-	-	X	X	X	X	X	X	X	X
CVS_Over_SSH2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	X	X	X	X	X	X	X
SVN	-	-	-	-	X	X	X	X	X	X	X	X	X	X	X	-	X	X	X	X	X	X
Subversive	-	-	-	-	-	-	-	-	-	X	X	X	X	X	-	-	-	-	-	-	-	X
Subclipse	-	-	-	-	X	X	X	X	X	-	-	-	-	-	X	-	X	X	X	X	X	-
Subclipse_1_4_x	-	-	-	-	-	X	X	X	-	-	-	-	-	-	X	-	-	-	-	X	-	-
Subclipse_1_6_x	-	-	-	-	X	X	-	-	-	-	-	-	-	-	-	-	X	X	X	-	-	-
GIT	-	-	-	-	-	-	-	X	-	-	-	-	-	-	X	X	-	X	X	-	X	X
EclipseFileSystem	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Local	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Zip	-	-	X	X	-	-	X	-	-	-	-	-	X	-	-	-	-	-	-	X	-	X



CIT: Is it really intractable?

- Answer: No!
- Outline of the argument
 - Analyze the complexity of CIT.
 - Identify the intractable part.
 - Identify the reason and whether that reason is valid.
 - Experiment that motivated the analysis.
 - Is our conclusions consistent with empirics?

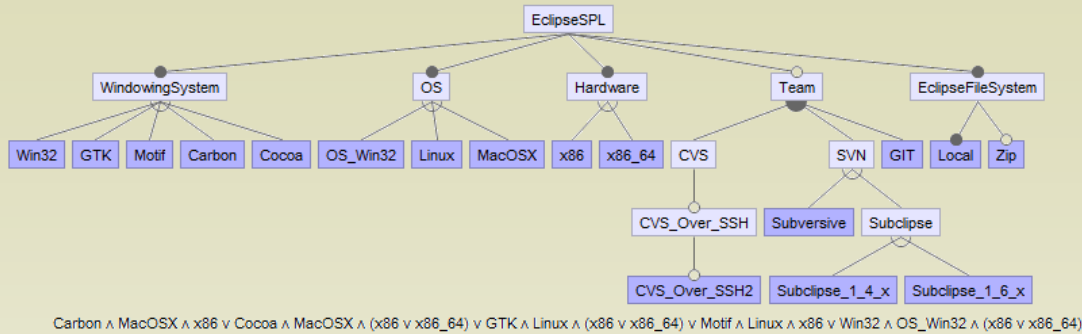
Covering array generation

- Set covering is NP-Complete
- Chvátal's algorithm
 - Greedy approximation of set cover
 - Add the product that cover as many new interactions as possible – until all are covered.
 - Add a product \equiv SAT
 - is NP-hard: No approximation is possible...
 - Therefore covering array generation is intractable...

Feature\ Product	1	2	3			
EclipseSPL	X	X	X			
WindowingSystem	X	X	X			
Win32	-	-	X			
GTK	-	X	-			
Motif	-	-	-			
Carbon	-	-	-			
Cocoa	X	-	-			
OS	X	X	X			
OS_Win32	-	-	X			
Linux	-	X	-			
MacOSX	X	-	-			
Hardware	X	X	X			
x86	X	-	-	+		
x86_64	-	X	X		+	
Team	-	-	-			
CVS	-	-	-			
CVS_Over_SSH	-	-	-			
CVS_Over_SSH2	-	-	-			
SVN	-	-	-			
Subversive	-	-	-			
Subclipse	-	-	-			
Subclipse_1.4.x	-	-	-			
Subclipse_1.6.x	-	-	-			
GIT	-	-	-			
EclipseFileSystem	X	X	X			
Local	X	X	X			
Zip	-	-	-			



Feature models and SAT



$$q = ((\sim p \ \& \ d \ \& \ c) \vee (p \ \& \ d \ \& \ c) \vee (p \ \& \ d \ \& \ \sim c) \vee (p \ \& \ \sim d \ \& \ \sim c)) \vee \dots$$



$p=\text{true}, d=\text{true}, c=\text{false}, \dots ?$

2
X
X
-
X
-
-
-
X
-
X
-
-
-
-
-
-
-
-
X
X
-

Takes exponential time on a SAT Solver...



Finding a single product

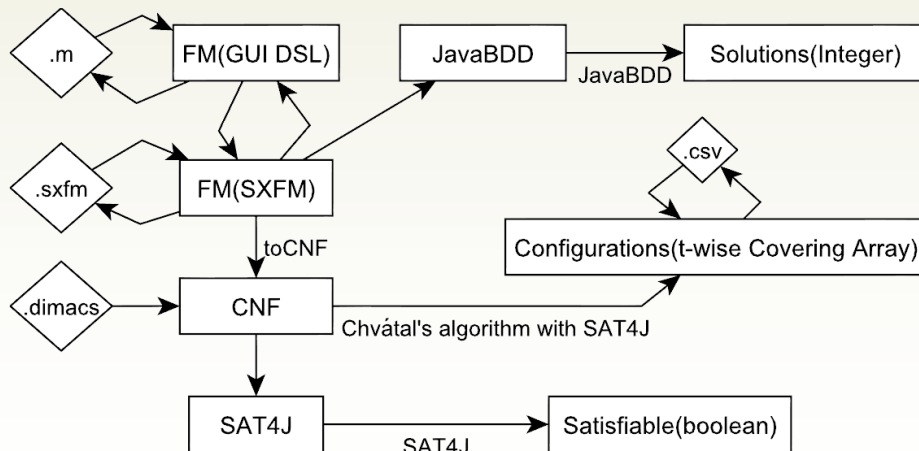
- A company has developed a product line and modeled the valid configurations/products with a feature model
- Scenario 1:
 - No one is able to configure it... (SAT is NP-hard)
 - Then testing is not your concern
- Scenario 2:
 - A lot of products in the market
 - Then it must have been relatively quick to configure it

Finding a single product

- Scenario 3:
 - A computer takes several hours to find a solution.
 - Violation of the purpose of feature models
 - Should enable customers to configure a product to fit their business needs.
 - Start with an empty configuration and add features as needed, while understanding the implications of their decisions.
 - Both customers and developers will be unable to reason about the configurations
- Therefore, realistic feature models are quickly satisfiable

Experiment

- gathered available feature models (not randomly generated)
- Implemented tool
 - Handles all the formats
- Our source of empirics.
- Feature models and tool available



Feature Model \ keys	Features	Constraints
2.6.28.6-icse11.dimacs	6,888	187,193
freebsd-icse11.dimacs	1,396	17,352
ecos-icse11.dimacs	1,244	2,768
Eshop-fm.xml	287	22
Violet.m	101	90
Berkeley.m	78	47
arcade_game_pl_fm.xml	61	35
Gg4.m	38	23
smart_home_fm.xml	35	1
TightVNC.m	30	4
Apl.m	25	3
fame_dbms_fm.xml	21	1
connector_fm.xml	20	1
Graph-product-line-fm.xml	20	15
stack_fm.xml	17	1
REAL-FM-12.xml	14	3
movies_app_fm.xml	13	1
aircraft_fm.xml	13	1
car_fm.xml	9	3



Is it consistent with empirics?

- SAT generally is $O(2^n)$
- Was a source of inspiration for the quick satisfiability

Feature Model \ keys	Features	Constraints	SAT time (ms)
2.6.28.6-icse11.dimacs	6,888	187,193	125
freebsd-icse11.dimacs	1,396	17,352	18
ecos-icse11.dimacs	1,244	2,768	12
Eshop-fm.xml	287	22	5
Violet.m	101	90	1
Berkeley.m	78	47	1
arcade_game_pl_fm.xml	61	35	3
Gg4.m	38	23	1
smart_home_fm.xml	35	1	9
TightVNC.m	30	4	1
Apl.m	25	3	1
fame_dbms_fm.xml	21	1	3
connector_fm.xml	20	1	3
Graph-product-line-fm.xml	20	15	3
stack_fm.xml	17	1	7
REAL-FM-12.xml	14	3	7
movies_app_fm.xml	13	1	3
aircraft_fm.xml	13	1	7
car_fm.xml	9	3	7



Future work

- A more efficient algorithm is still needed.
 - The quick satisfiability property can be utilized to achieve this.

Feature Model \ keys	Features	Constraints	2-way size	2-way time (ms)
2.6.28.6-icse11.dimacs	6,888	187,193	n/a	n/a
freebsd-icse11.dimacs	1,396	17,352	n/a	n/a
ecos-icse11.dimacs	1,244	2,768	n/a	n/a
Eshop-fm.xml	287	22	22	364,583
Violet.m	101	90	28	21,278
Berkeley.m	78	47	23	11,195
arcade_game_pl_fm.xml	61	35	17	8,219



In conclusion

- Our contributions

- An experiment on a collection of realistic feature models
 - Feature models
 - Tool implementation
- Quick satisfiability is a property of realistic feature models.
- That property makes CIT feasible for testing realistic product lines.