

# Software Product Lines

## Part 2: Development process and feature modeling

**Daniel Strüber**, Radboud University

with courtesy of: **Sven Apel**, **Christian Kästner**, **Gunter Saake**

# Agenda

---

- ▶ Software product lines and features
- ▶ Process models for introducing software product lines
  - ▶ Practical example: Microsoft
- ▶ Feature modeling

# Product lines

## Diversity of variants

Example: BMW X3



Ceiling:  
90.000  
variant options

Car doors:  
3.000  
variant options

Rear axle:  
324  
variant options

„Variants are a significant leverage for our operating results.“  
— Franz Decker, head of variant management programme, BMW Group

7

## More product lines



## Linux kernel

- about 6.000.000 lines of source code
- highly configurable
  - > 10.000 configuration options! (x86, 64bit, ...)
  - almost all source code is „optional“



## Printer firmware



# Software product lines

---

“A software product line (SPL) is a **set of software-intensive systems** that share a common, managed set of **features** satisfying the specific needs of a particular **market segment or mission** and that are developed from a common set of **core assets** in a prescribed way. ”

*Software Engineering Institute  
Carnegie Mellon University*

# Software product line

---

- ▶ A set of program variants... (*software products*),
  - ▶ ...sharing a common basis of functionality (*features*)
  - ▶ ...tailored towards a particular market segment (*domain*)
  - ▶ ...with the goal of *reusing* common software artifacts
- 
- ▶ or example:
    - ▶ database product line for embedded systems

# Domain

---

- ▶ The products of a product line are tailored to a particular application field
- ▶ This application field is called *domain*
- ▶ Horizontal domains
  - ▶ Billing, inventory management, flight booking
- ▶ Vertical domains
  - ▶ Numeric algorithms, network drivers, GUIs, databases

# What is a feature?

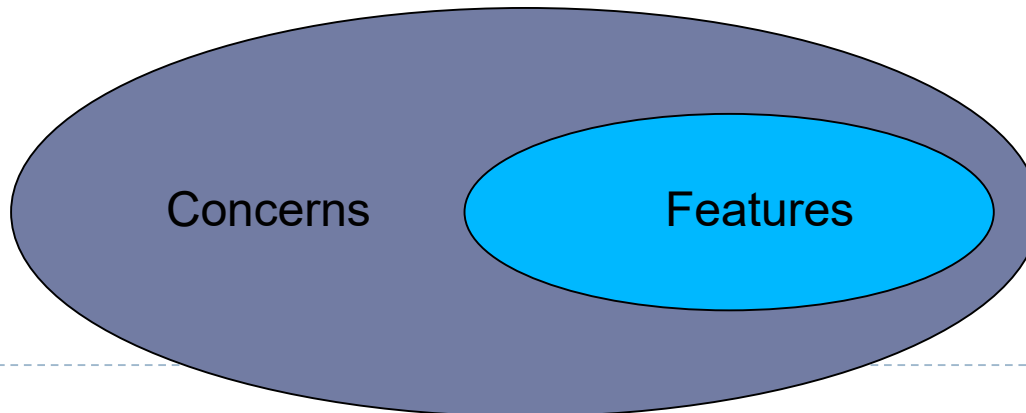
---

- ▶ An object of the domain abstraction
- ▶ Features represent requirements, commonalities and differences of products
- ▶ Means of communication between stakeholders
- ▶ Allows the specify products
  - ▶ Feature selection as input for product generation
- ▶ Related to a system's *concerns*
  - ▶ In the sense of the „separation of concerns“ principle
  - ▶ But there are concerns that are not a feature

# Concern vs. Feature

---

- ▶ Concern (explored further in later chapters)
  - ▶ Any problem statement of interest
- ▶ Feature
  - ▶ Problem statement of interest in the domain
  - ▶ Might be a configuration option

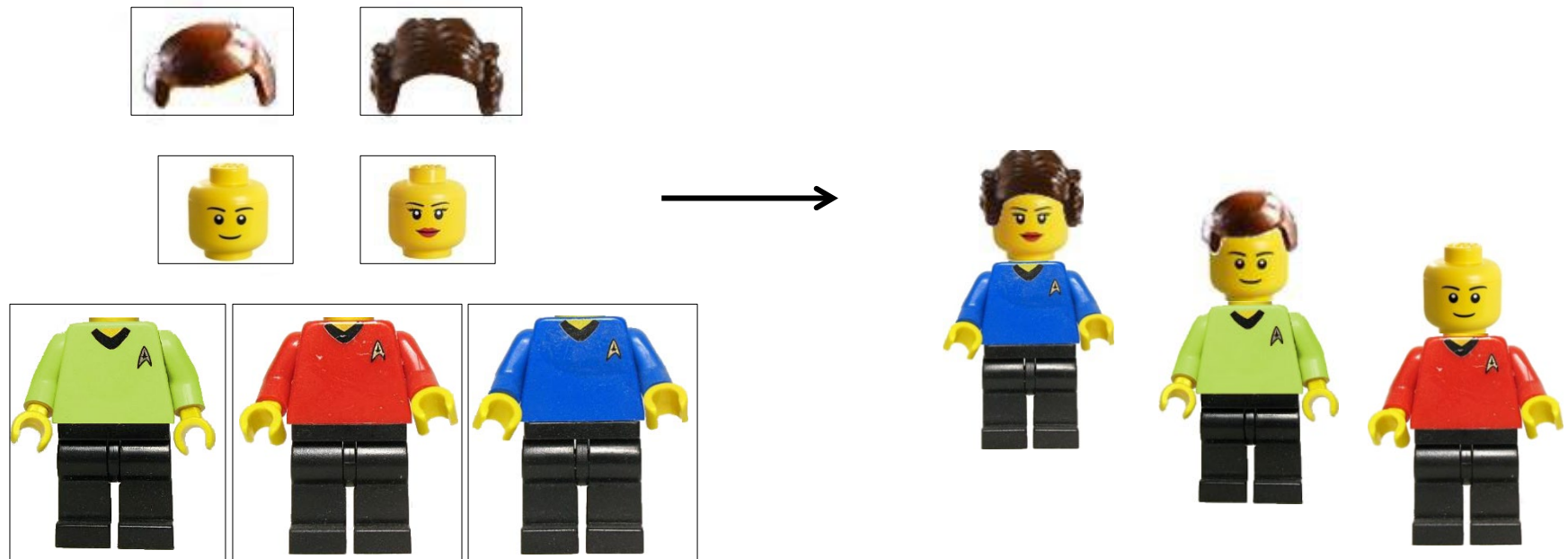




# Features vs. products/variants

---

- ▶ Features are building blocks of a product line (implemented, for example, as components, packages...)
- ▶ Combinations of features lead to products = variants



# Features of database systems

---

- ▶ Transaction management
- ▶ Logging & recovery
- ▶ Write access
- ▶ Persistence / In-Memory
- ▶ Caching: LRU / LFU / Clock /...
- ▶ Sorting algorithm
- ▶ Datatypes of variable length
- ▶ Grouping, aggregation
- ▶ Windows / Unix / NutOS / TinyOS / ...



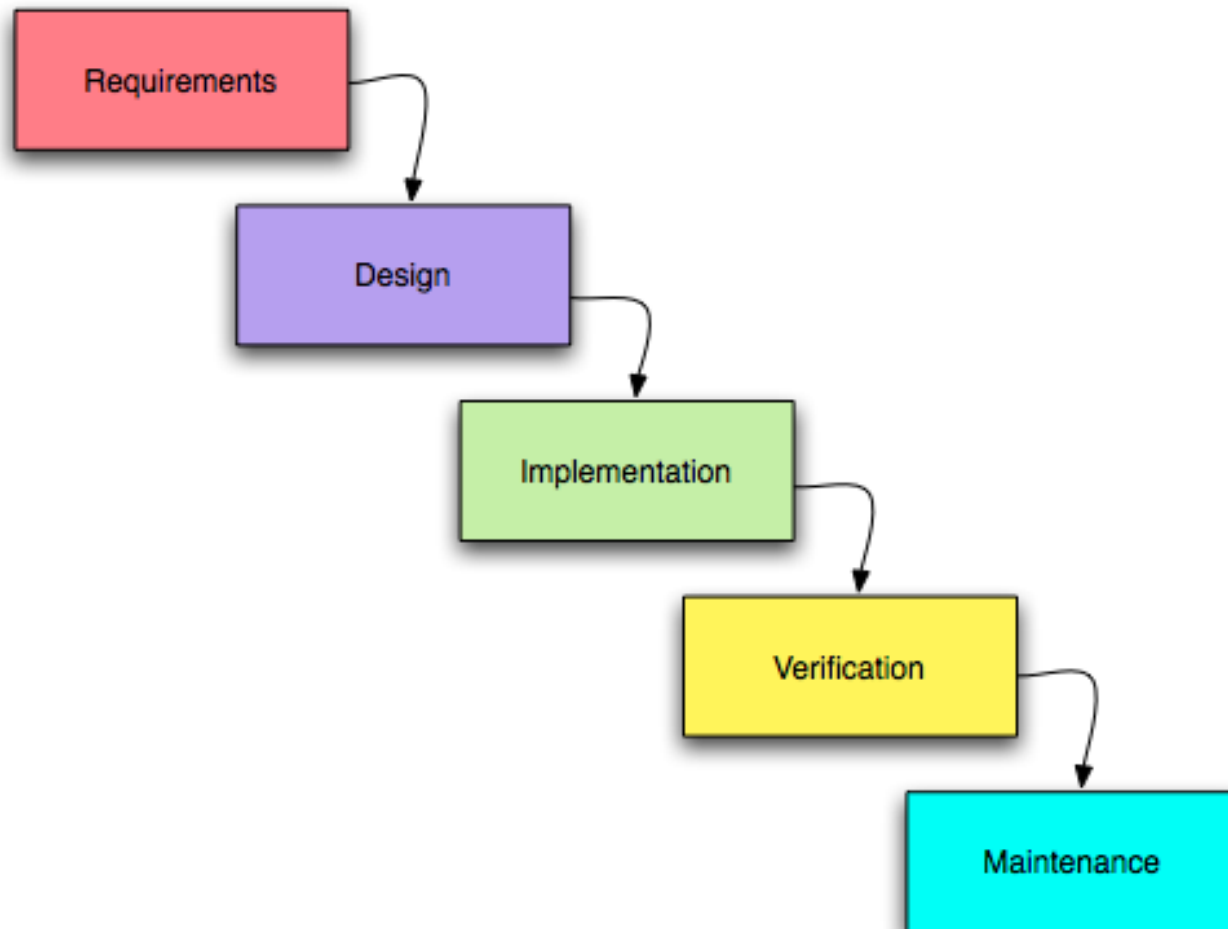
# Development of a software product line

---

- ▶ Develop a product line instead of individual products
- ▶ Product line covers requirements of entire domain
- ▶ Deviate from classic development processes and lifecycle
- ▶ Distinguish two levels
  - ▶ Domain Engineering
  - ▶ Application Engineering

# Software lifecycle: classic

---



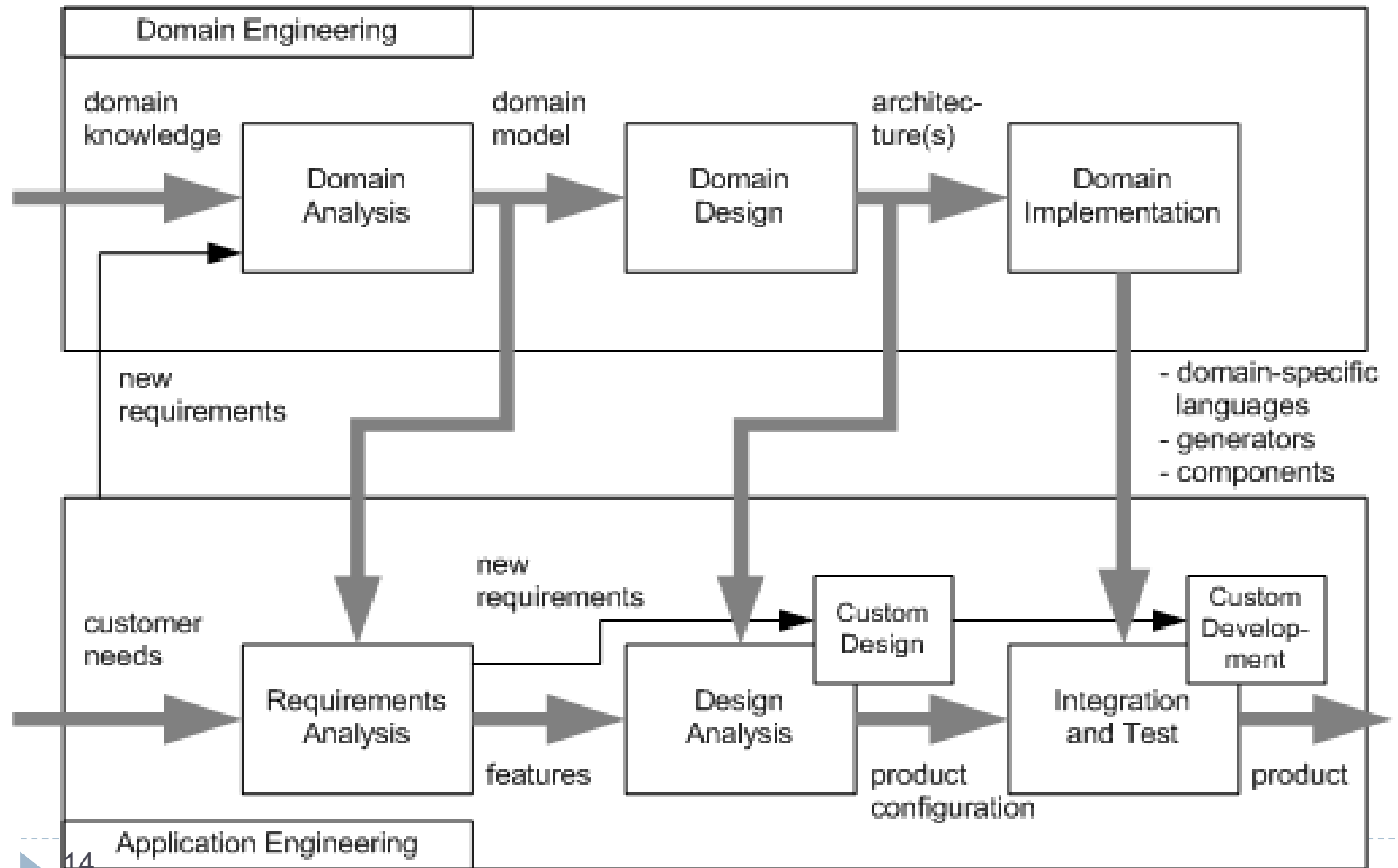
# Domain Engineering

---

“The activity of collecting, organizing, and storing past experience in building systems [...] in a particular domain in the form of reusable assets [...], as well as providing an adequate means for reusing these assets (i.e., retrieval, qualification, dissemination, adaptation, assembly, and so on) when building new systems.”

*K. Czarnecki and U. Eisenecker*

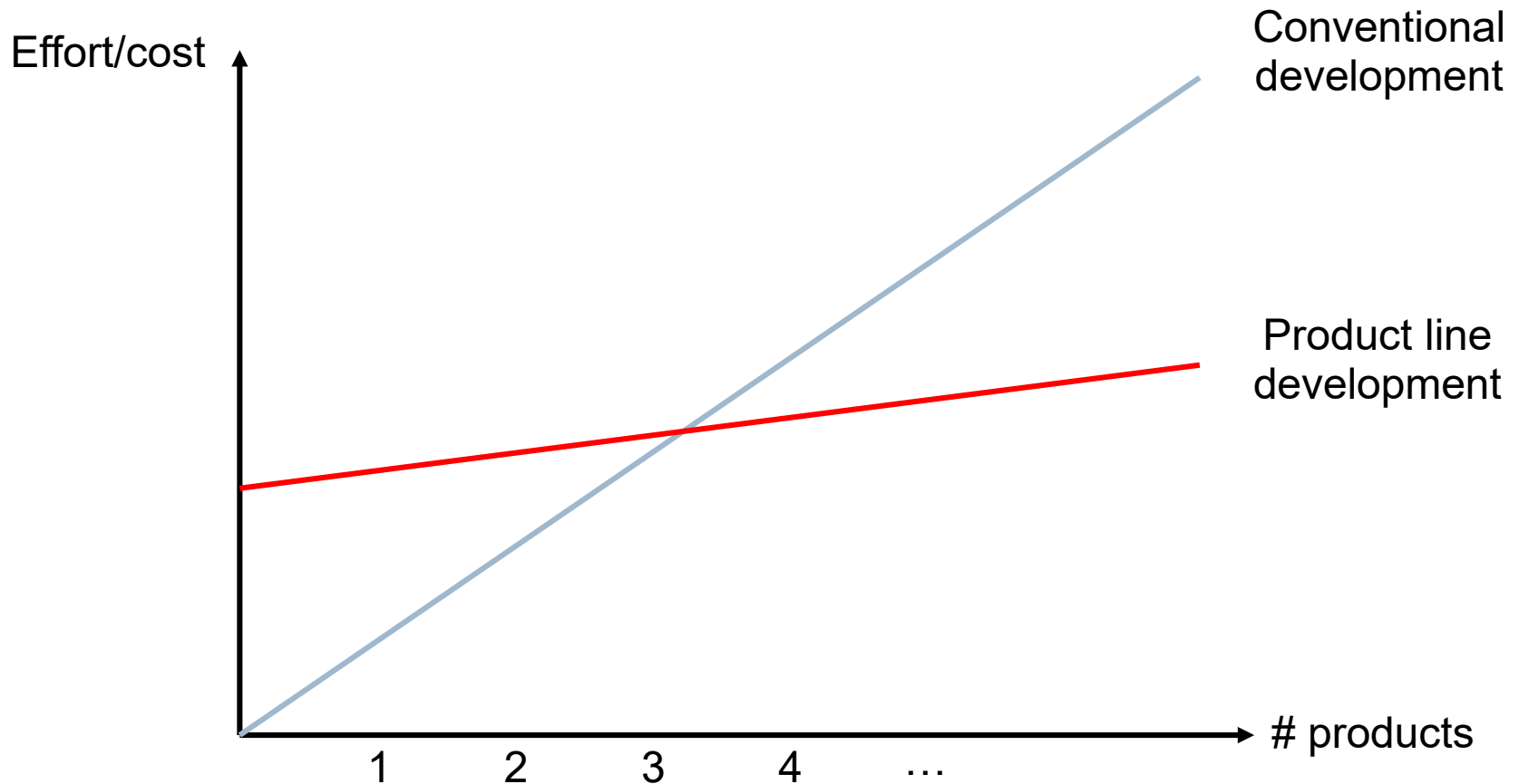
# Application and Domain Engineering



# Development effort

---

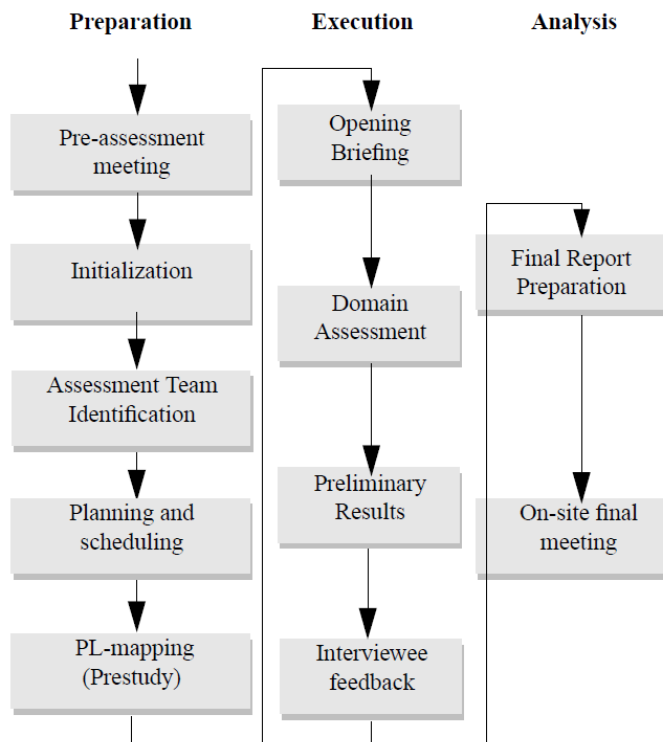
- ▶ Effort for developing a software product line



# Scoping

- ▶ Defining the boundaries of the domain
- ▶ Which features are relevant/should be developed?
- ▶ Decision often depends on economic reasoning

[Schmid 2002]



			exist.	planned		potent.
			P1	P2	P3	P4
Domain 1	Sub-Domain 1.1	Feature 1.1.1	X	X	X	X
		Feature 1.1.2	—	X	X	X
		Feature 1.1.3	X	X	—	X
	...	...	...	...	...	...
	Sub-Domain 1.n	Feature 1.n.1	X	—	X	X
		...	...	...	...	...
Domain 2	Sub-Domain 2.1	Feature 2.1.1	—	X	X	—
		...	...	...	...	...
...	...	...	...	...	...	...
...	...	Feature m.1.1	—	X	—	X





# Process models

# Process models for introducing product lines

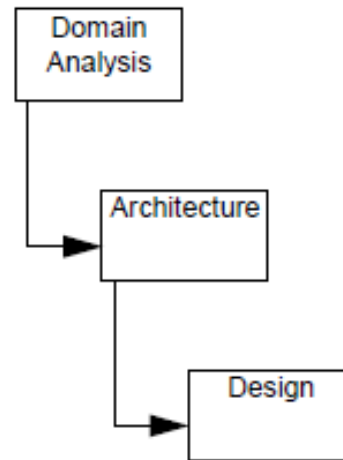
---

- ▶ There are three main approaches to introducing a product line
  - ▶ Proactive process model
  - ▶ Reactive process model
  - ▶ Extractive process model
- ▶ Independent of the used implementation technique
- ▶ Selection follows business concerns (costs, risks, chances...)

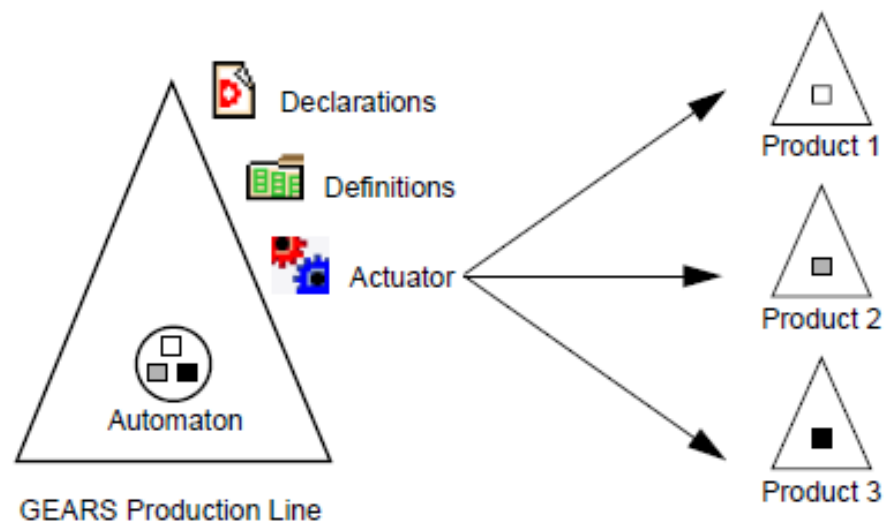
# Proactive process model

---

- ▶ Design and implement product line from scratch, in the way discussed before
- ▶ Complete domain analysis in beginning
- ▶ It might be necessary to pause the normal development for a few months, until the product line supports the already-existing products
- ▶ Potentially high cost, high risk
- ▶ Useful if the requirements are clearly defined



Proactive  
Implementation

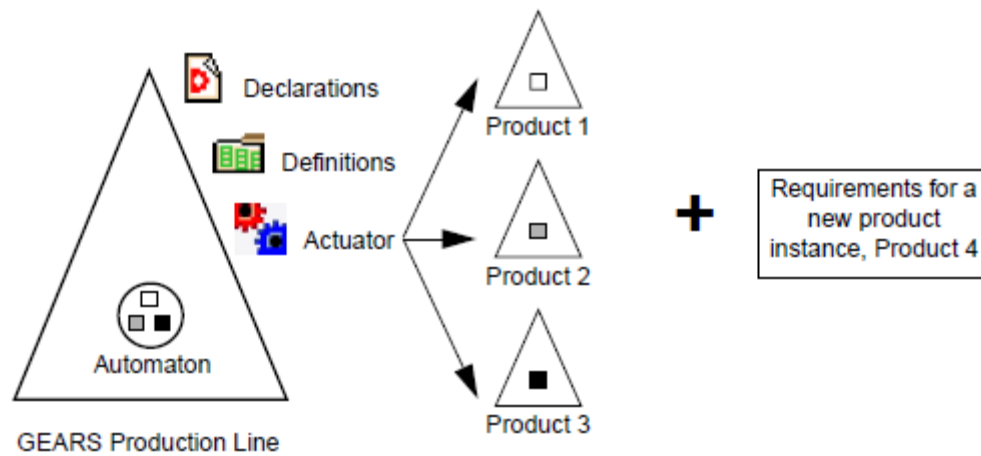


[Krueger 2002]

# Reactive process model

---

- ▶ Related to the *spiral model* and *extreme programming*
- ▶ Implement a few products first, and incrementally add more products later
- ▶ Suitable if the required products are not known in advance, and if unforeseeable changes are likely
- ▶ Smaller project size, smaller starting costs; fewer resources bound, faster results
- ▶ Might need restructuring later

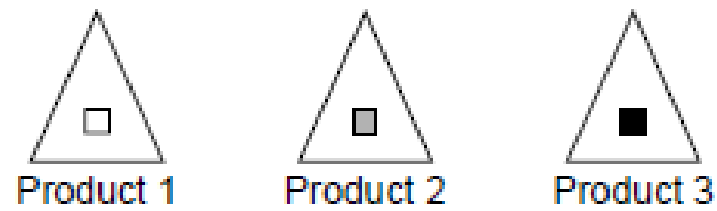


[Krueger 2002]

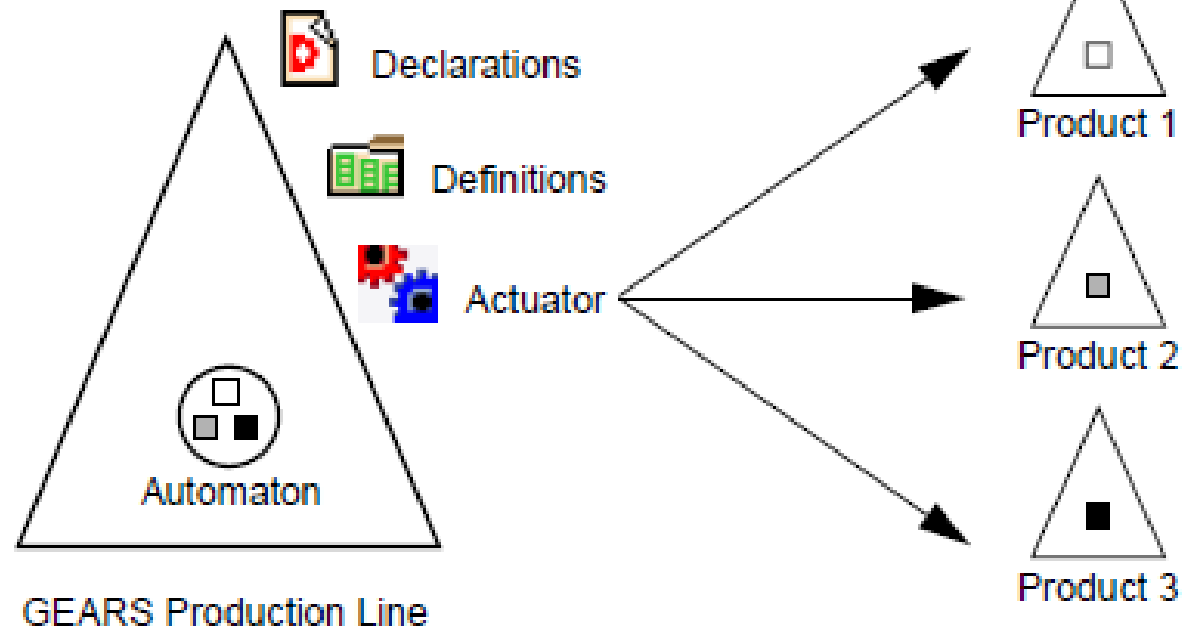
# Extractive process model

---

- ▶ Uses one or several existing products as basis
- ▶ Extracts features from there, which then allows to generate products
- ▶ Useful for rapid change from traditional applications to product lines
- ▶ Relatively small resource demands and risk
- ▶ Challenging for tools and programming languages:
  - ▶ how to turn an application that was not designed as a product line into one?
  - ▶ how do identify and locate features in existing products?



Extract



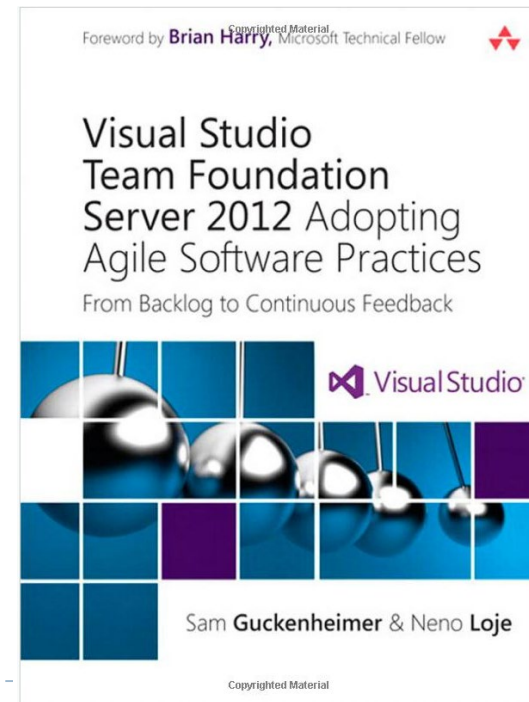
[Krueger 2002]



# Practical example: Microsoft

---

- ▶ Development department
  - ▶ Visual Studio
  - ▶ .Net Framework
- ▶ Going from single-product to product-line development, VS2005 vs. VS2008 (chapter 9)
  - ▶ 3 500 developers
  - ▶ 20 000 000 source files
  - ▶ 700 000 work elements
  - ▶ 2 000 nightly builds
  - ▶ 15 terabytes of data
  - ▶ Much variability (editions, components, ...)



# The situation in 2005 (i)

---

- ▶ 5 systems, not interoperable
  - ▶ Code management, bug tracking, build automation, test case management
  - ▶ „silos“, developed in isolation for decades
- ▶ Heterogenous business context
  - ▶ Freeware (Silverlight, Express versions, .Net) with the goal of winning over customers and have them develop for Microsoft-based systems
  - ▶ Commercial systems (z.B. Visual Studio, MSDN) with subscription model
  - ▶ Tension between these objectives, often in conflict

## The situation in 2005 (ii)

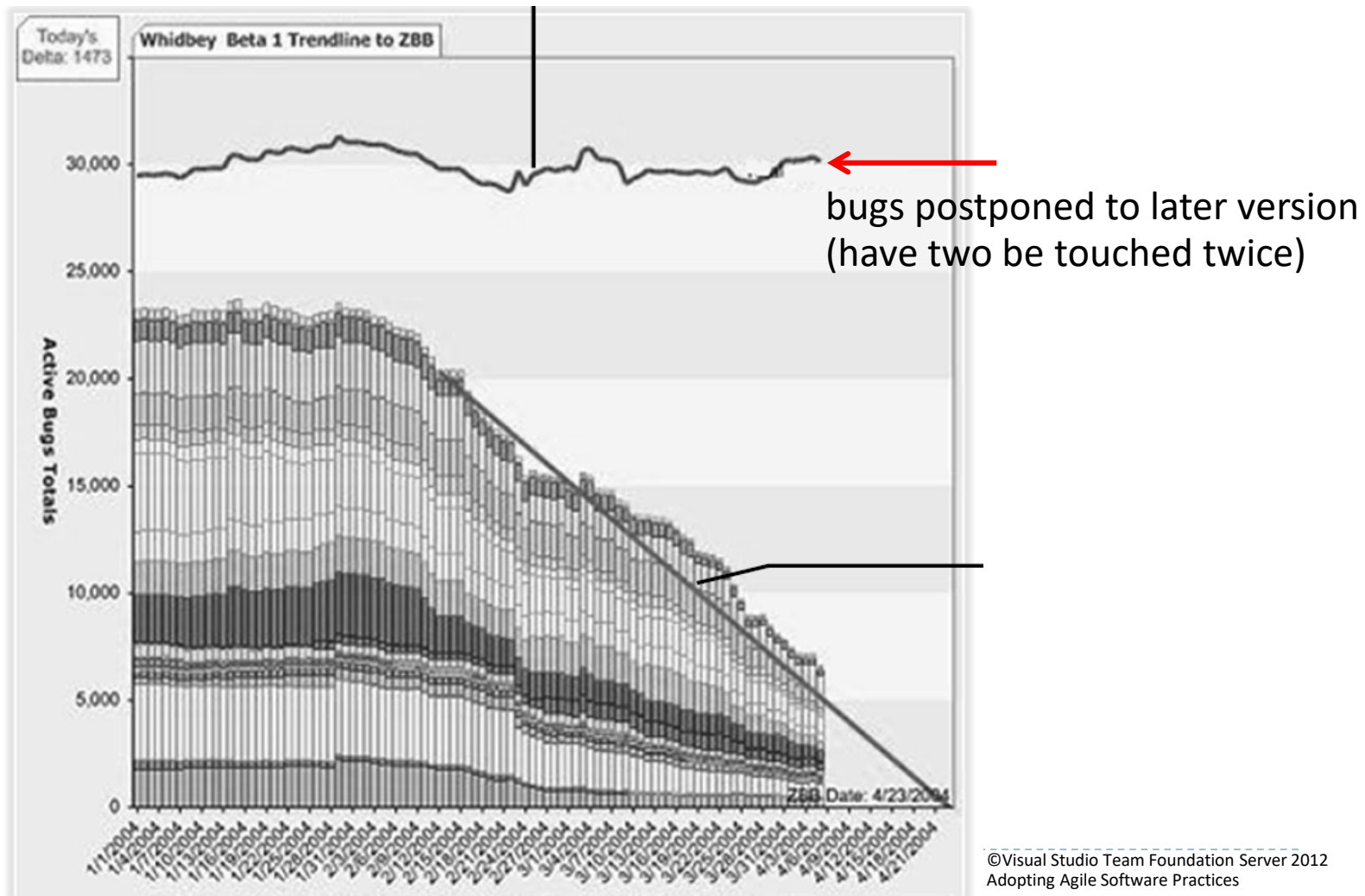
---

- ▶ No central decisionmaker
- ▶ No “product backlogs” for summarizing requirements and enabling comparisons
  - ▶ “Putting together a list of over 1000 features just seemed impossible”
- ▶ 5 behavior patterns of departments
  - ▶ Non-aggression pact between managers
  - ▶ Schedule Chicken: No department could make the deadline, but departments knew that other departments also couldn’t make them
  - ▶ Performance indicators are for other departments (but not me)
  - ▶ Our customers are different (because of broadness of domain)
  - ▶ Our group is better



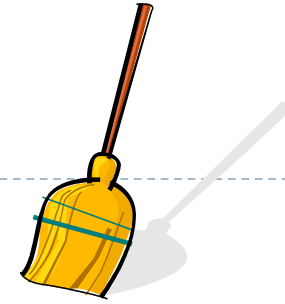
# Waste of resources


## ► VS2005 Beta-1: number of open bugs over time



# Improvements after 2005 (i)

---

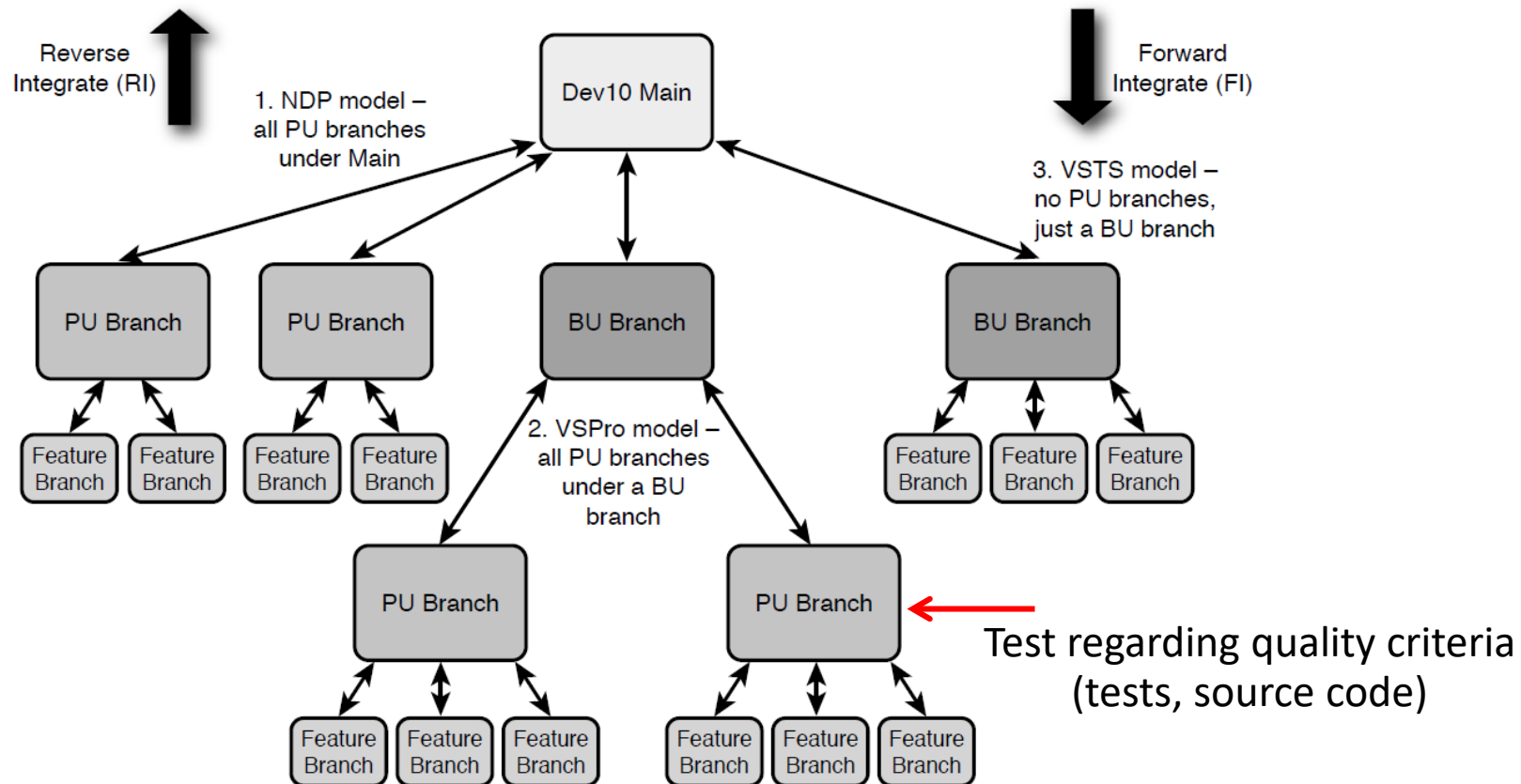


- ▶ Clean up and establish rules
  - ▶ Invest in fixing open bugs from beginning
- ▶ Stricter timeboxing
  - ▶ Instead of 3-month plans, 3-week sprints for the completion of features
  - ▶ Goal: After each sprint, deployable products with new functionality increments (features)
- ▶ Feature crews 
  - ▶ Small, interdisciplinary teams (5—6 developers, tester, manager)
  - ▶ Work on an isolated branch of the code base with special insularity rules (**modularity!**)



# Feature crews and product units (PUs)

- Integration (in main branch) und isolation (from changes of other crews)



# Improvements after 2005 (ii)

---

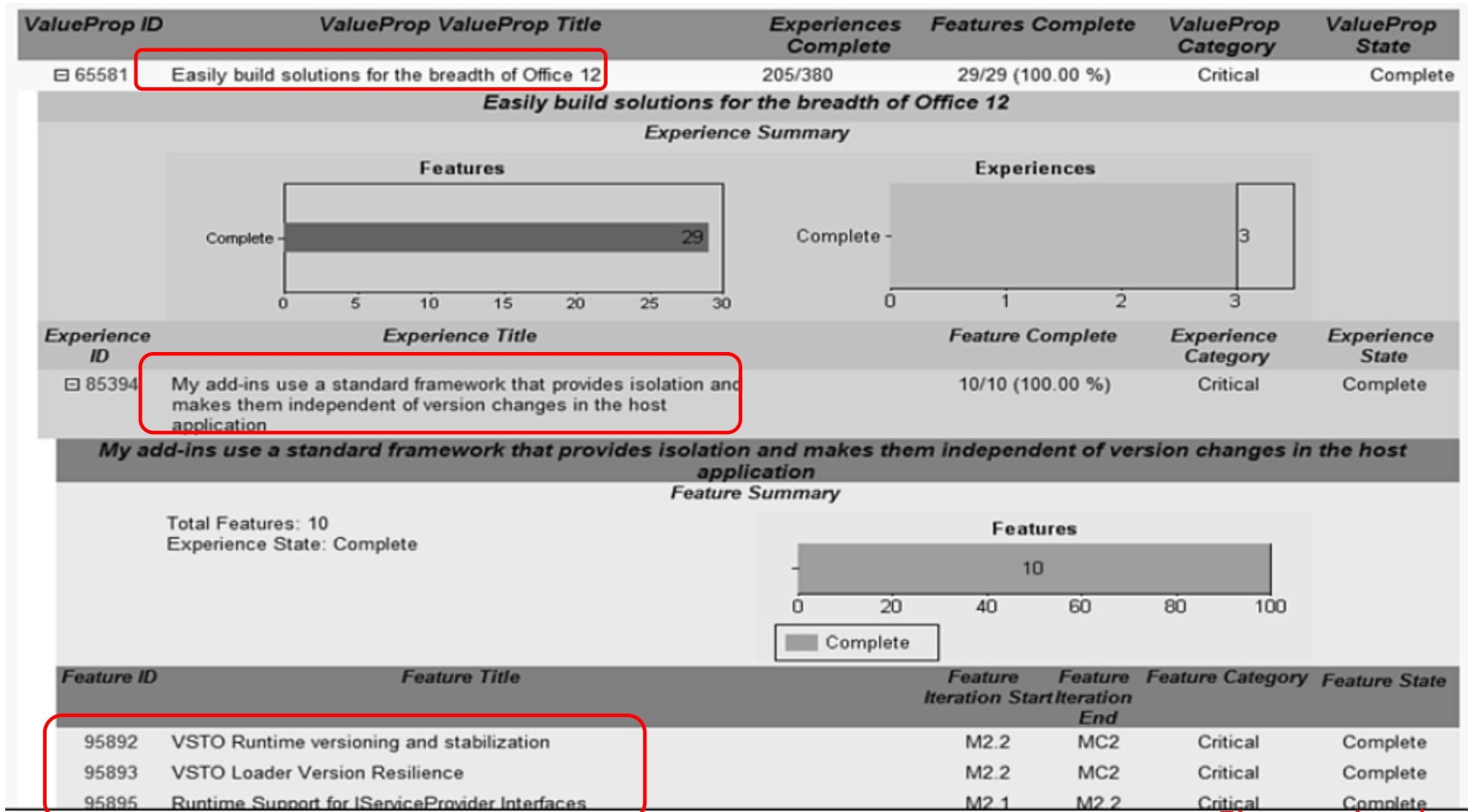
## ▶ Product backlogs

- ▶ “DevDiv was an organization conditioned over a decade to think in terms of features. Define the features, break them down into tasks, work through the tasks, and so on. The problem with this granularity is that it encourages “peanut buttering”, an insidious form of overproduction. Peanut buttering is the mindset that whatever feature exists in the product today needs to be enhanced in the next release. [...] From a business perspective, this is obviously an endless path into bloat.”
- ▶ New structure: focus on getting the granularity right. Describe added value, user experience and functionality by using canonical questions.

# Improvements after 2005 (ii)

## ▶ Product backlogs

Coarse-grained,  
understandable for customers



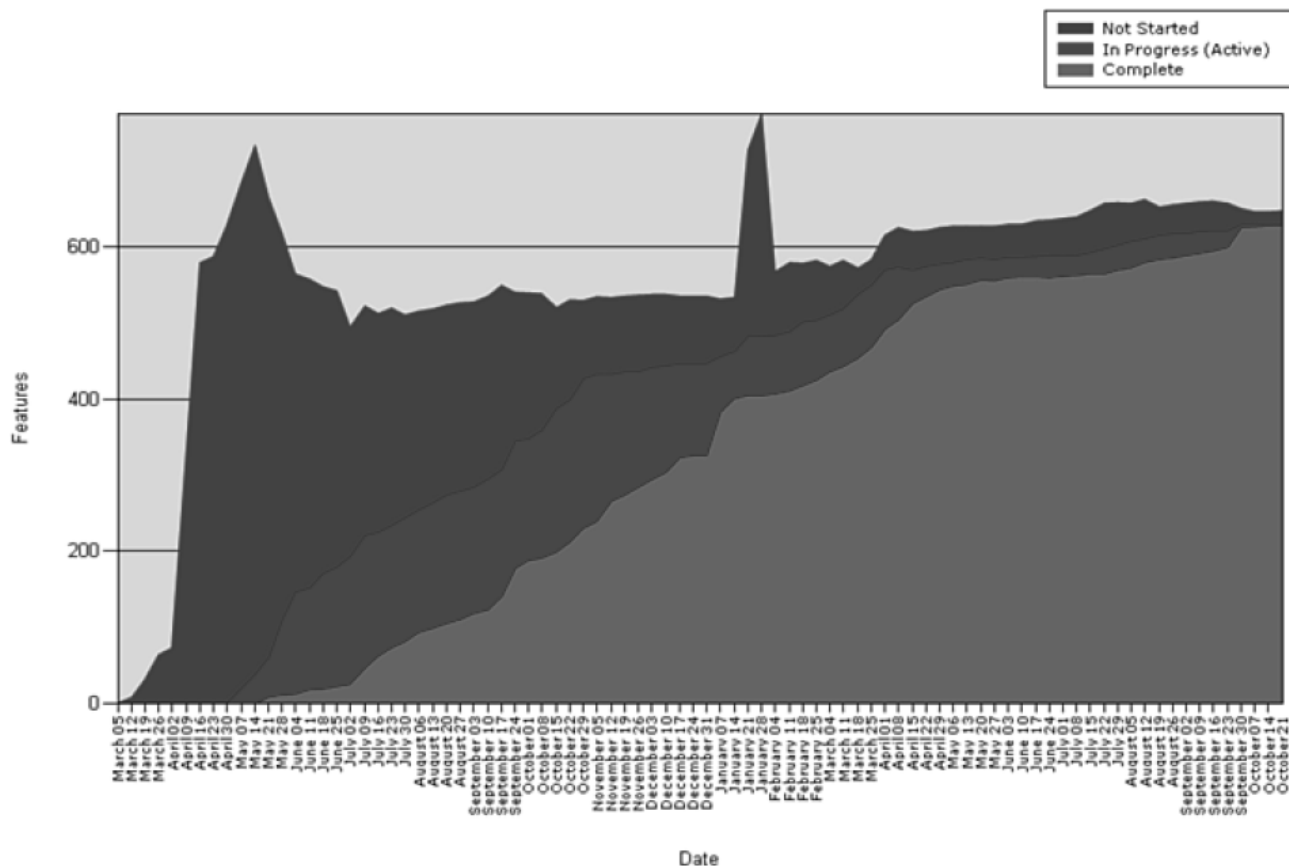
©Visual Studio Team Foundation Server 2012  
Adopting Agile Software Practices

Fine-grained,  
precise for developers



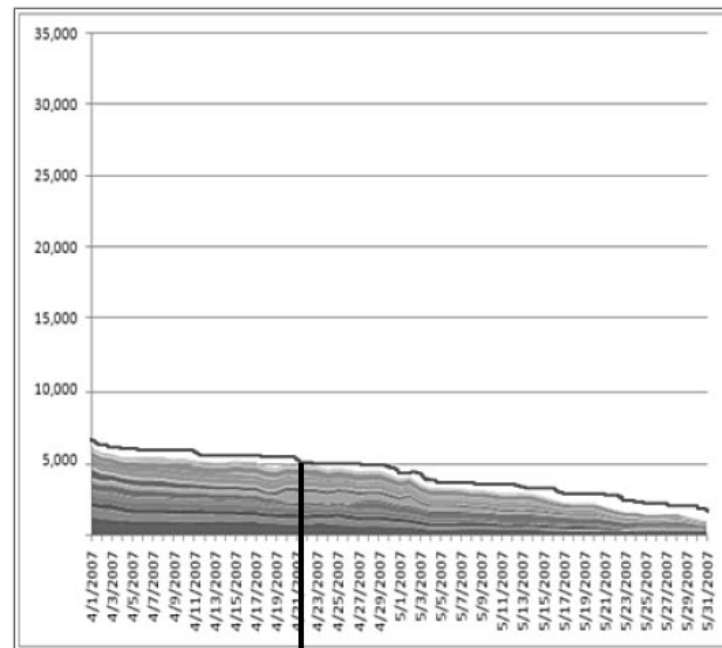
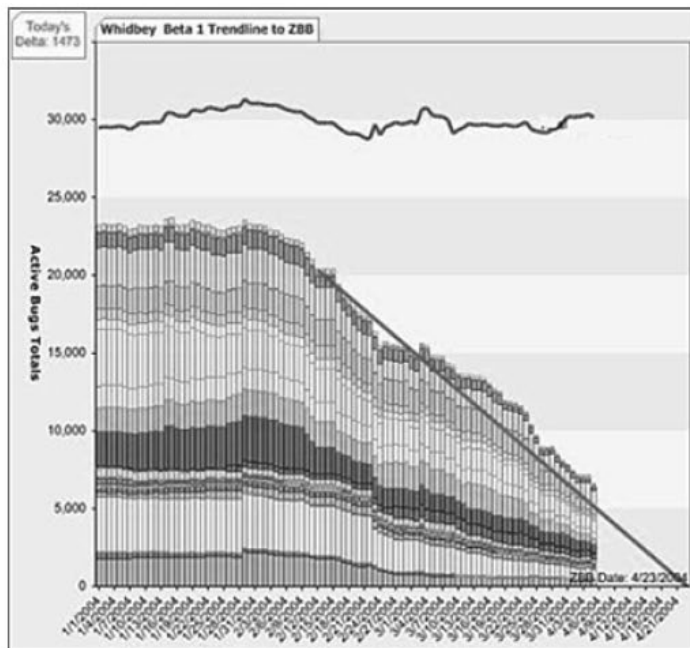
# Results (i)

- Features were abstract enough to be interesting for customers and usable for another feature, but detailed enough to be implementable in 3 weeks



## Results (ii)

- ▶ Development time to new release cut to 50%
- ▶ 15 times less bugs
- ▶ Increased customer satisfaction



Total bug debt at Beta 1 of VS 2008.

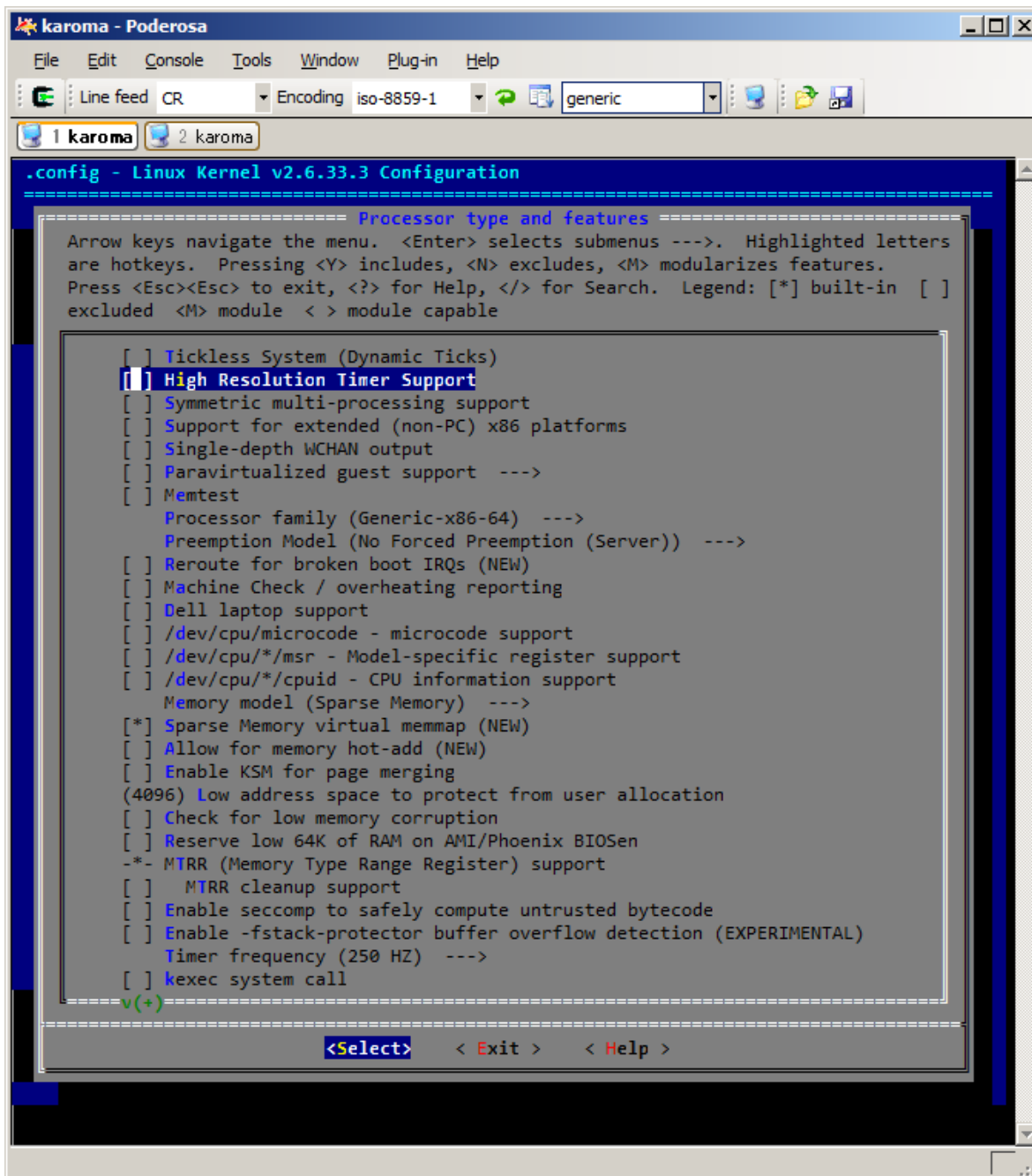


# Feature modeling

# Feature modeling

---

- ▶ Define the features of a domain
- ▶ Support visualization and communication
- ▶ A feature model describes:
  - ▶ the fundamental concepts of a domain and their relationships
  - ▶ the set of products of the product line
- ▶ Graphically, feature models are represented as feature diagrams, visualizing the relationships between features



## VEGETARIAN

WHICH WICH WOULD YOU LIKE?

- ☐ TRIPLE CHEESE MELT  
☐ ELVIS WICH (P, Honey & Banana)  
☐ TOMATO & AVOCADO  
☐ BLACK BEAN PATTY  
☒ HUMMUS & BELL PEPPERS

CHOOSE YOUR BREAD

- ☐ WHITE  
☒ WHEAT

CHOOSE YOUR CHEESE (Optional)

- ☐ AMERICAN ☐ SWISS ☐ PROVOLONE  
☐ CHEDDAR ☒ PEPPER JACK ☐ MOZZARELLA

## How Would You Like Your WICH Worked?

MUSTARDS

- ☐ Yellow ☐ Dijon ☐ Honey ☒ Deli

MAYOS

- ☐ Regular ☐ Lite ☐ Horseradish ☒ Spicy

SPREADS & SAUCES

- ☐ BBQ ☐ Buffalo ☐ Marinara  
☐ 1000 Island ☐ Ranch

ONIONS

- ☒ Red ☐ Grilled ☐ Crispy Strings

VEGGIES

- ☒ Lettuce ☒ Tomato ☐ Pickles ☒ Jalapenos  
☒ Olive Salad ☐ Mushrooms ☐ Sauerkraut  
☐ Coleslaw ☐ Bell Peppers

OILS & SPICES

- ☐ Oil ☐ Vinegar  
☒ Salt ☒ Pepper ☐ Oregano ☐ Parmesan

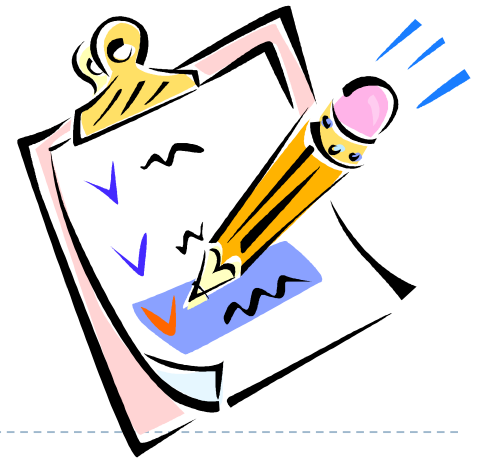
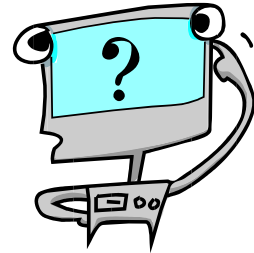
EXTRAS (.75¢ Each)

- ☐ Bacon ☐ Avocado ☐ Pickle (Whole)  
☐ More Meat ☐ More Cheese

# Valid feature selection?

---

- ▶ Transaction management
- ▶ Logging & recovery
- ▶ Write access
- ▶ Persistence / In-Memory
- ▶ Caching: LRU / LFU / Clock / ...
- ▶ Sorting algorithm
- ▶ Datatypes of variable length
- ▶ Grouping, aggregation
- ▶ Windows / Unix / NutOS / TinyOS / ...



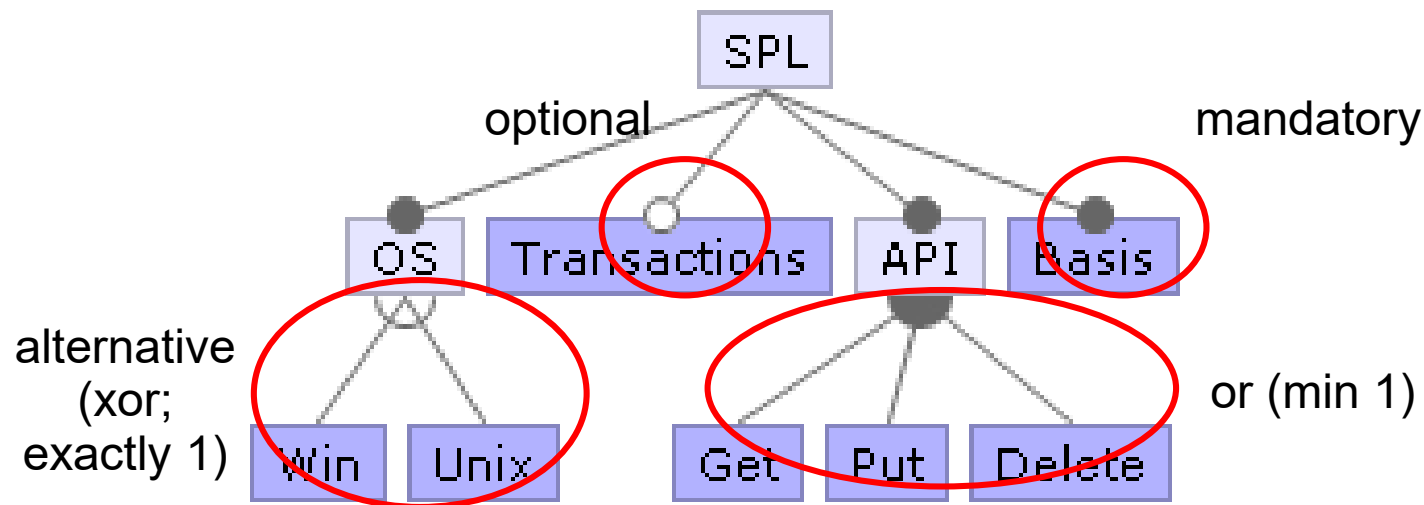
# Feature model: example

---

- ▶ Features: Basis, Txn, Write, Win, Unix
- ▶ Dependencies:
  - ▶ Basis is always selected and requires Win or Unix
  - ▶ Win may never be selected together with Unix
  - ▶ If Txn is selected, Write must be selected as well
- ▶ How many products exist?
  - ▶ Six:
  - ▶ {Basis, Win}, {Basis, Unix}, {Basis, Win, Write}, {Basis, Unix, Write}, {Basis, Win, Write, Txn}, {Basis, Unix, Write, Txn}

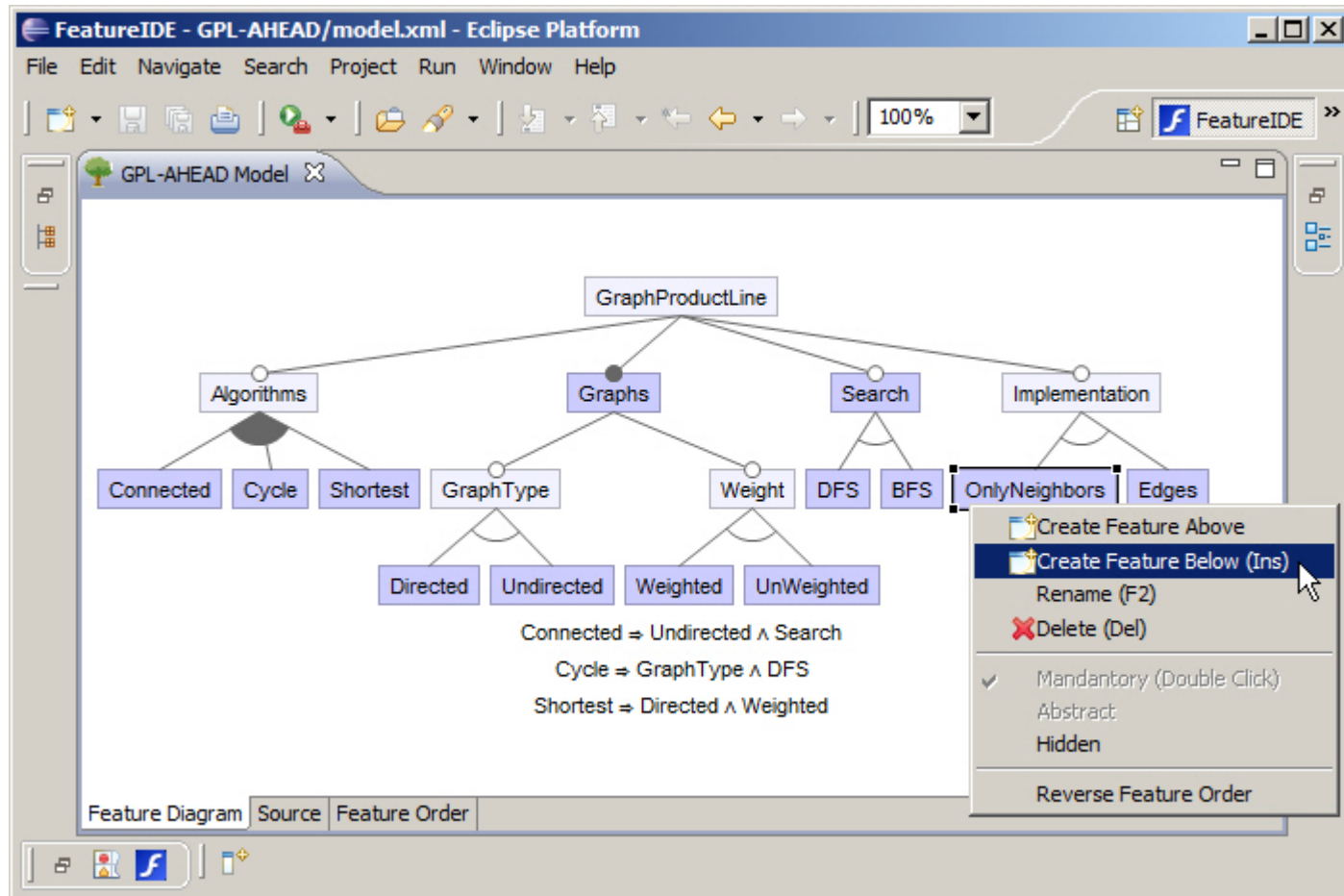
# Feature diagram

- ▶ Graphical representation
- ▶ Tree structure
- ▶ Children: optional, mandatory, or (“or”), alternative (“xor”)
- ▶ Abstract and concrete features, leaves are concrete





# Tool demo: FeatureIDE



Tutorial available at <https://github.com/FeatureIDE/FeatureIDE/wiki/Tutorial>

## From feature models to propositional logic

---

- ▶ Variable for each feature (true if selected)
- ▶ Propositional formula describes feature model
- ▶ Formula becomes true for valid feature selection

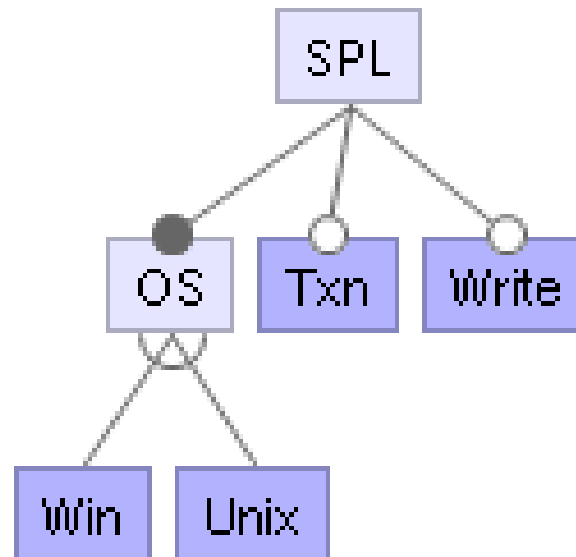
$$\begin{aligned} & \textit{Basis} \wedge (\textit{Unix} \vee \textit{Win}) \wedge \neg(\textit{Unix} \wedge \textit{Win}) \\ & \wedge (\textit{Txn} \Rightarrow \textit{Write}) \end{aligned}$$

- ▶ Allows automated checking and enumeration of valid products, based on reasoning tools: SAT solvers, BDD, ...

# Feature diagrams vs. formulas

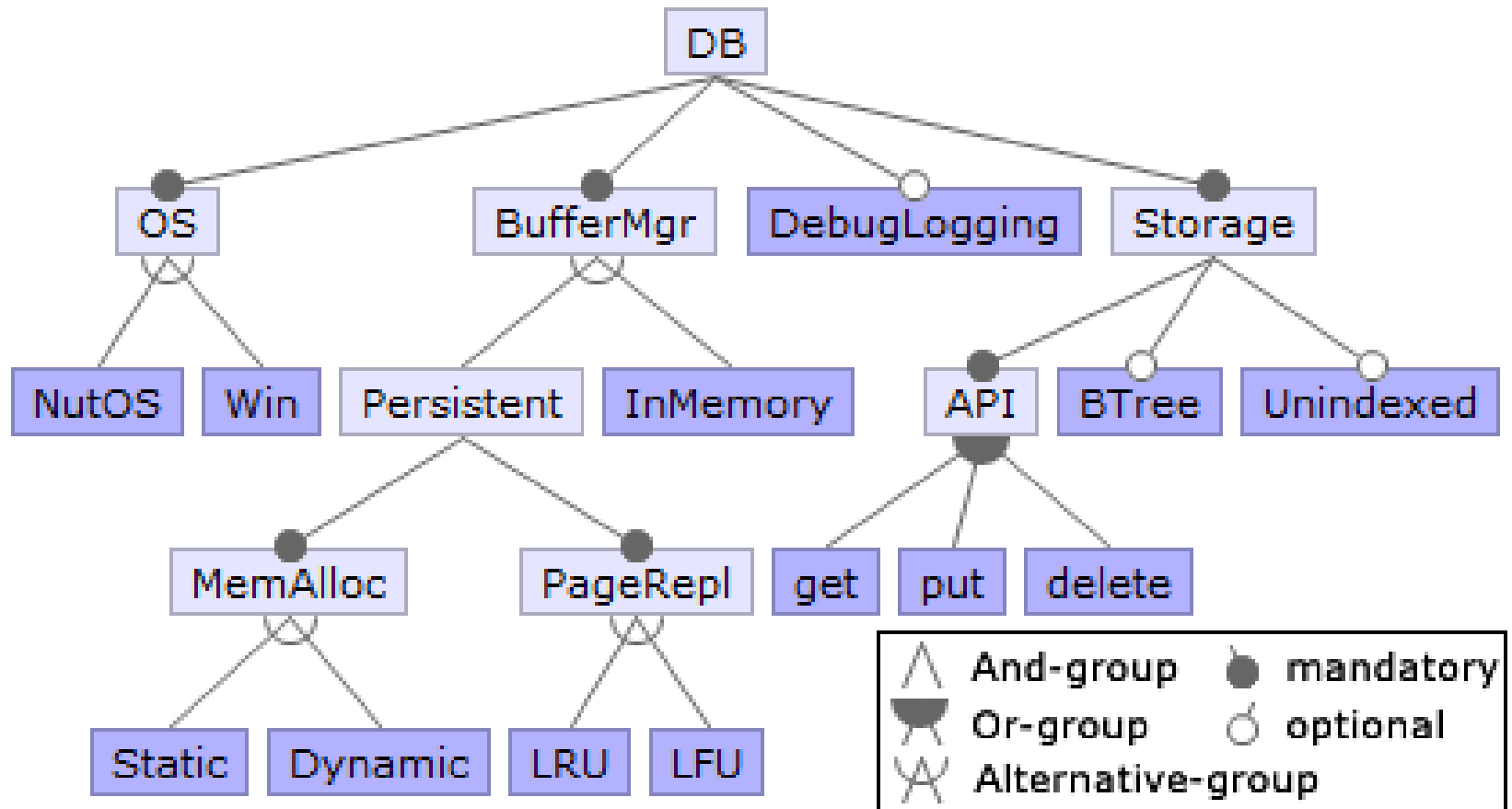
---

- ▶ Diagrams are easier to read (“management compatible”)
- ▶ Less expressive → might need to enrich diagrams
- ▶ Translation „diagram → formula” can be automated



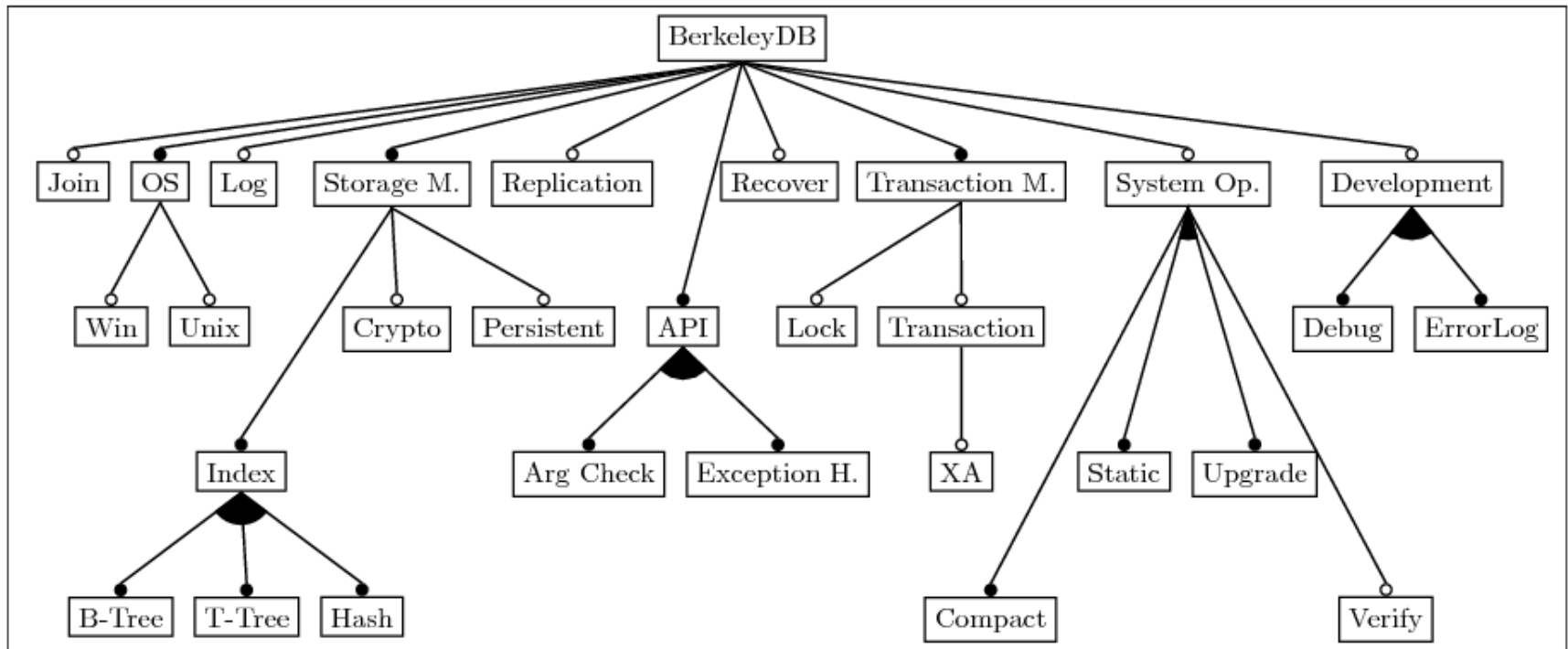
$\text{Txn} \Rightarrow \text{Write}$

## Example – FAME DBMS (Core)

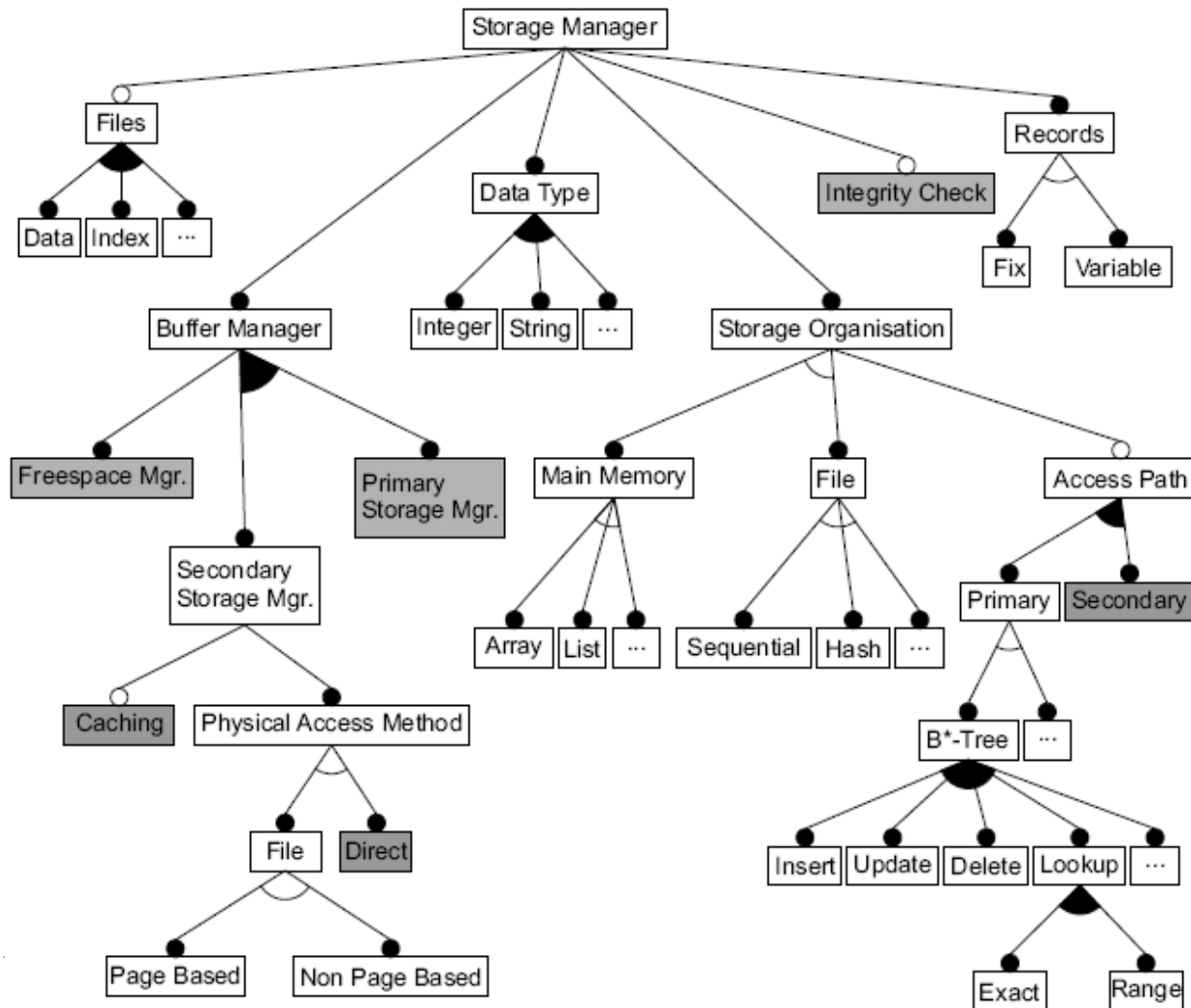


$(\text{Storage} \Rightarrow \text{BTree} \vee \text{Unindexed}) \wedge \neg(\text{BTree} \wedge \text{Unindexed})$

# Example – Berkeley DB (after a refactoring)

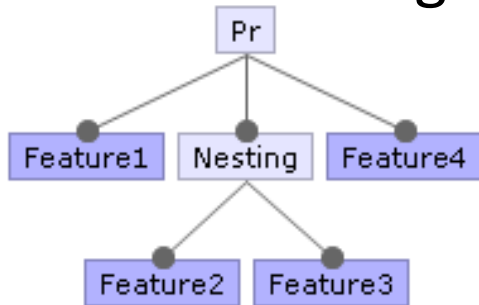


# Feature model of a storage manager

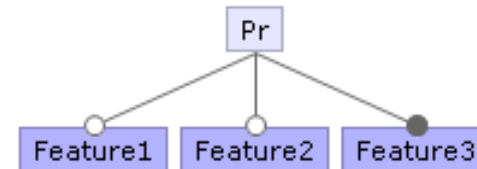


# GUIDSL format

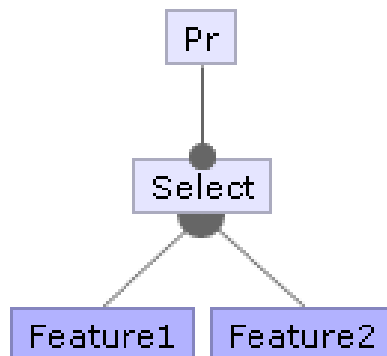
- ▶ Storing the feature diagram as a grammar
- ▶ A sentence of the grammar is a valid configuration



**Pr: Feature1 Nesting Feature4::\_Pr;  
Nesting: Feature2 Feature3::\_Nesting;**



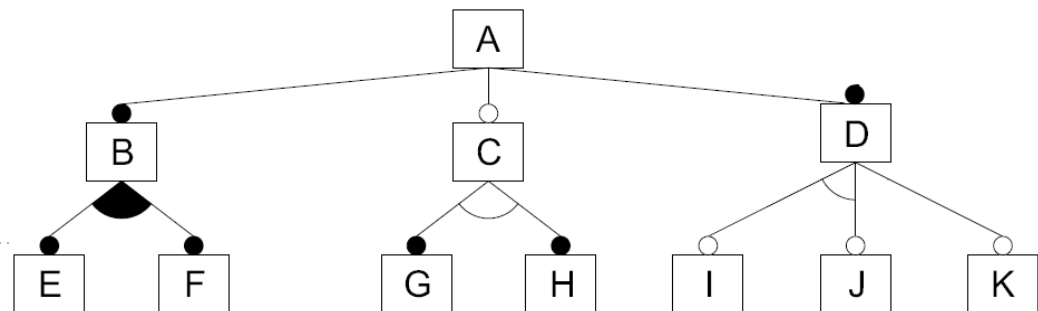
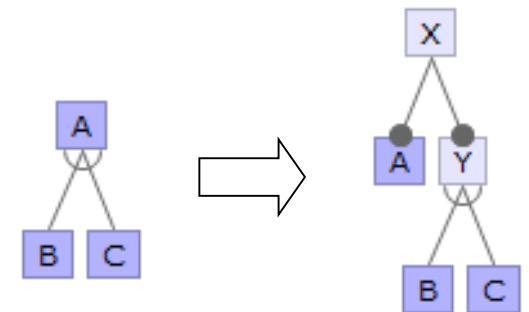
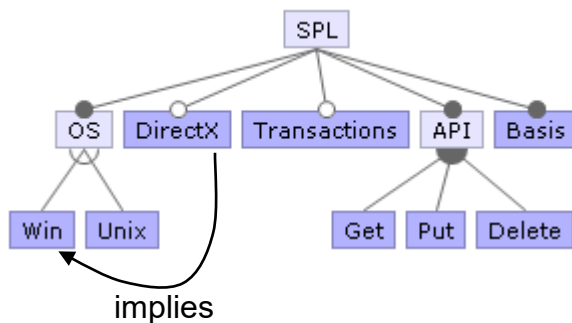
**Pr: [Feature1] [Feature2] Feature3  
::\_Pr;**



**Pr : Select+ :: \_Pr ;  
Select : Feature1 | Feature2 ;**

# Feature diagram - variations

- ▶ Different variations in literature, for example:
  - ▶ Cross-tree constraints: formulas vs. *implies/excludes* arrows
  - ▶ Non-leaf nodes can only be abstract
  - ▶ Group children can be mandatory or optional
- ▶ Usually possible to convert between two given variations

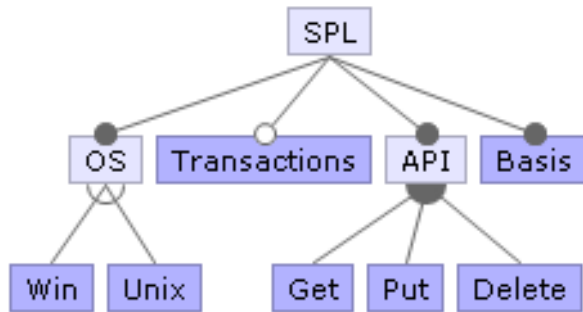




# Quiz

- How many variants are expressed by the feature model?

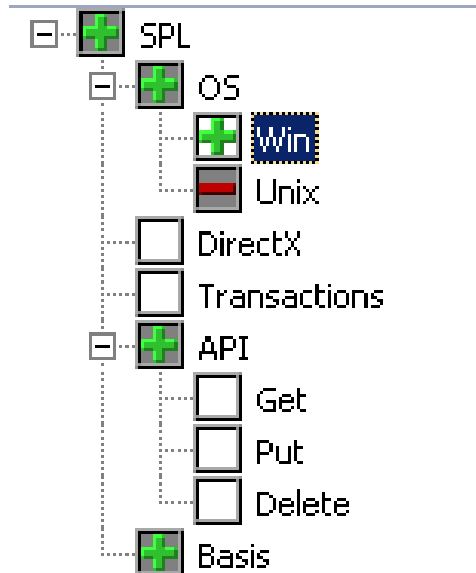
(a)



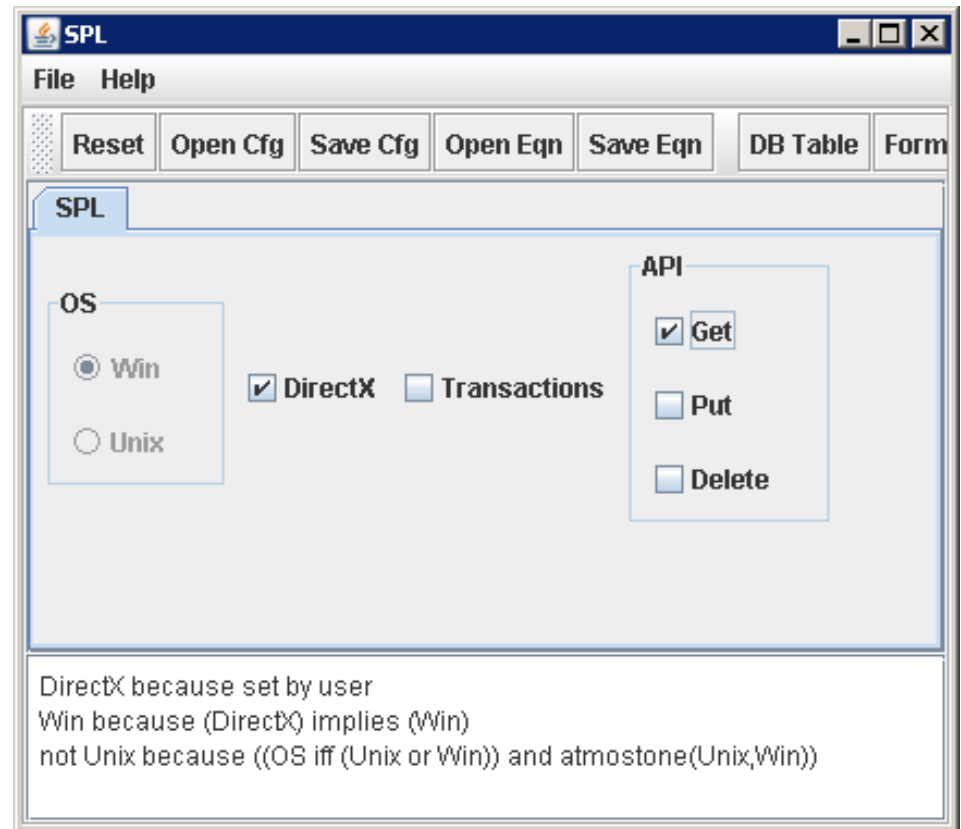
(b)

$$\text{Basis} \wedge (\text{Unix} \vee \text{Win}) \wedge \neg(\text{Unix} \wedge \text{Win}) \\ \wedge (\text{Txn} \Rightarrow \text{Write})$$

# Configuration



FeatureIDE



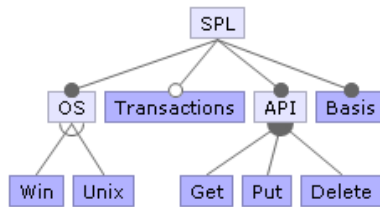
GUIDSL

# Design and implementation of features

- ▶ After feature modeling: need to design and implement features

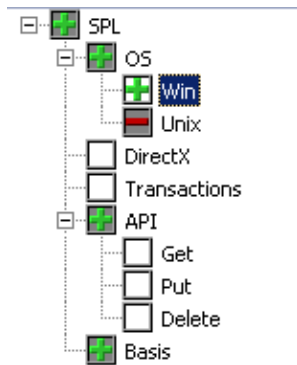
Domain Eng.

Feature model



Reusable  
implementation  
artifacts

Application Eng.



Feature selection



Generator



	CUST_NO	CUSTOMER	CONTACT..	CONTACT..	PHONE
1	1,001	Signature ...	Dale J.	Little	(619) 531
2	1,002	Dallas Tex...	Olen	Brown	(214) 986
3	1,003	Buttle, Grifi	James	Buttle	(617) 481
4	1,004	Central Bank	Elizabeth	Brocket	81 211 9
5	1,005	DT Systems	Tai	Wu	(852) 851
6	1,006	DataServe ...	Tomas	Bright	(613) 221
7	1,007	Mrs. Beauv...		Mrs. Beauv...	
8	1,008	Anini Vacat...	Lellani	Briggs	(809) 831
9	1,009	Max	Max		22 01 23

Final program

# Summary

---

- ▶ Software product lines as a concept for enabling systematic reuse
- ▶ Development split up in domain engineering and application engineering
- ▶ Features represent domain concepts
- ▶ Products of a product line share a common set of features
- ▶ Feature models describe features and their relationships...
- ▶ ...and are used for configuration

# Outlook

---

- ▶ The next chapters are about methods, techniques, and tools for implementing software product lines

# Literature I

---

- ▶ K. Kang, S. Cohen, J. Hess, W. Novak, and A. Peterson. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-21, SEI, 1990. [Early ideas for domain analysis with feature models]
- ▶ K. Czarnecki and U. Eisenecker. Generative Programming: Methods, Tools, and Applications. Addison-Wesley, 2000. [Comprehensive description of domain analysis, feature models and their normalization]

# Literature II

---

- ▶ D. Batory. Feature Models, Grammars, and Propositional Formulas, In Proc. of Software Product Line Conference (SPLC), 2005
- ▶ General books on software product lines:
  - ▶ P. Clements, L. Northrop, Software Product Lines : Practices and Patterns, Addison-Wesley, 2002
  - ▶ K. Pohl, G. Böckle, F. van der Linden, Software Product Line Engineering: Foundations, Principles, and Techniques, Springer, 2005