

An Overview of Techniques for Detecting Software Variability Concepts in Source Code

Angela Lozano

Université catholique de Louvain, Belgium

Variability

- Customized *and* Affordable
 - Maximize reuse of common features
 - e.g. reuse hw. customize with sw.
 - e.g. sw families = { similar apps with shared provenance }
 - e.g. context-aware, fault tolerant and intelligent apps

Why mining for variability?


- **Scattering & tangling** of vp's & Fs
- **pull-up** Fs to the core or **push down** Fs because they are variable

- To recuperate from architectural degradation 

- To expand a successful single product to new markets $\leftrightarrow \updownarrow$

possible variable features

- **Cost-benefit**
 - over-generalization vs. over-trivialization (**cost** of a making a F variable)
 - Evaluate != variability **mechanisms** (flexibility vs. performance)

- Effect of a variation in the development of the product 

- **Trace** variability REQ -> IMPL
- Explicit **dependencies**
- Appropriateness of binding times & variability mechanisms (flexibility vs.

Single vs. Multiple features

Variable features

Feature Diagram

Feature dependencies

Mandatory vs. Optional features

Feature

Variability

Optional vs. Alternative variants

Variants

Binding

Variation points

Domain instances

Domain instances
Feature

Variable features

Single vs. Multiple features

Mandatory vs. Optional features

Variability

Feature Diagram

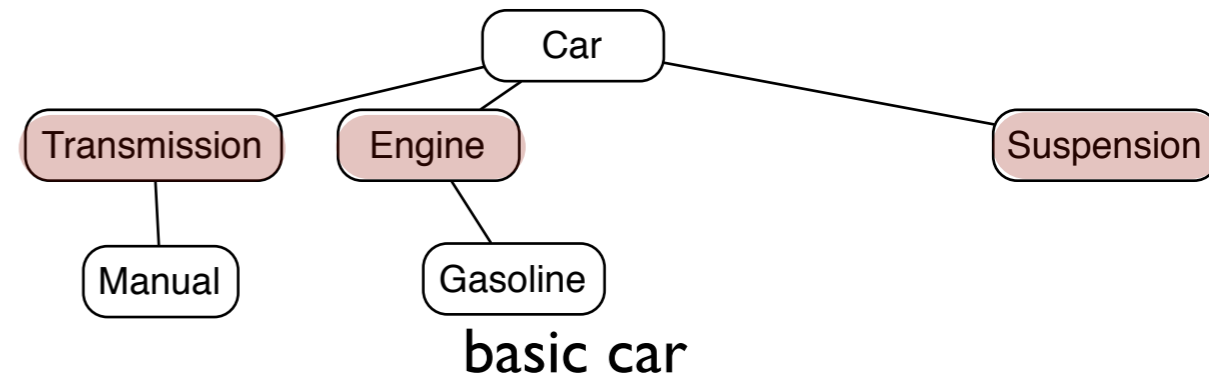
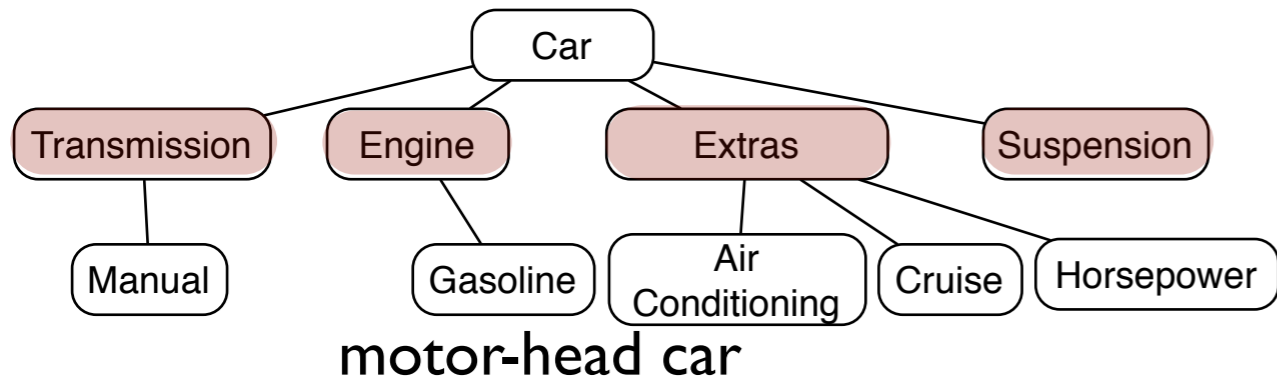
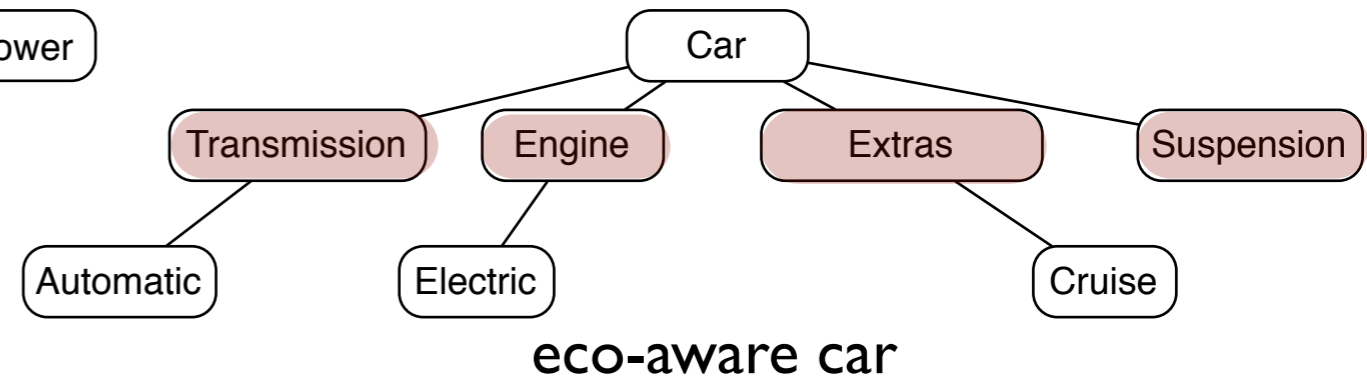
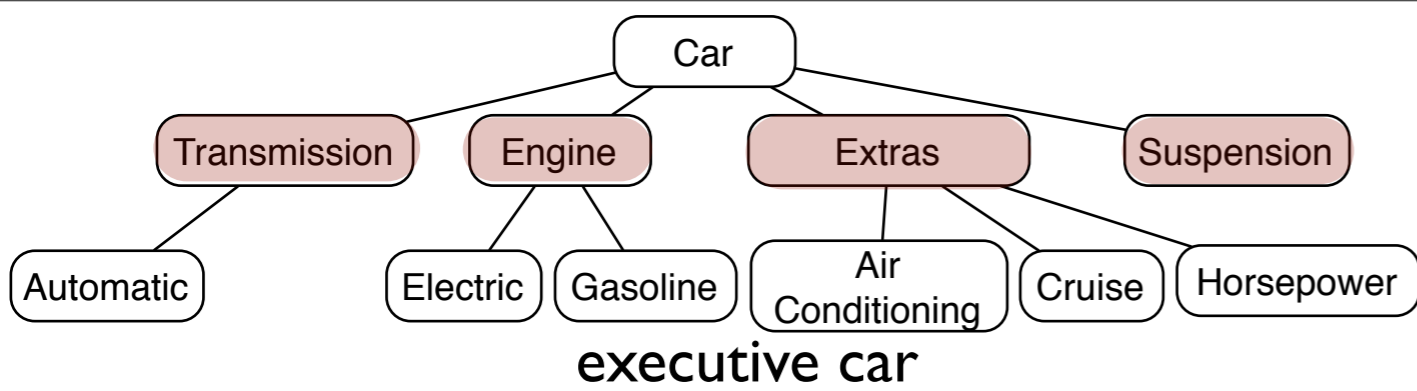
Variants

Optional vs. Alternative variants

Binding

Variation points

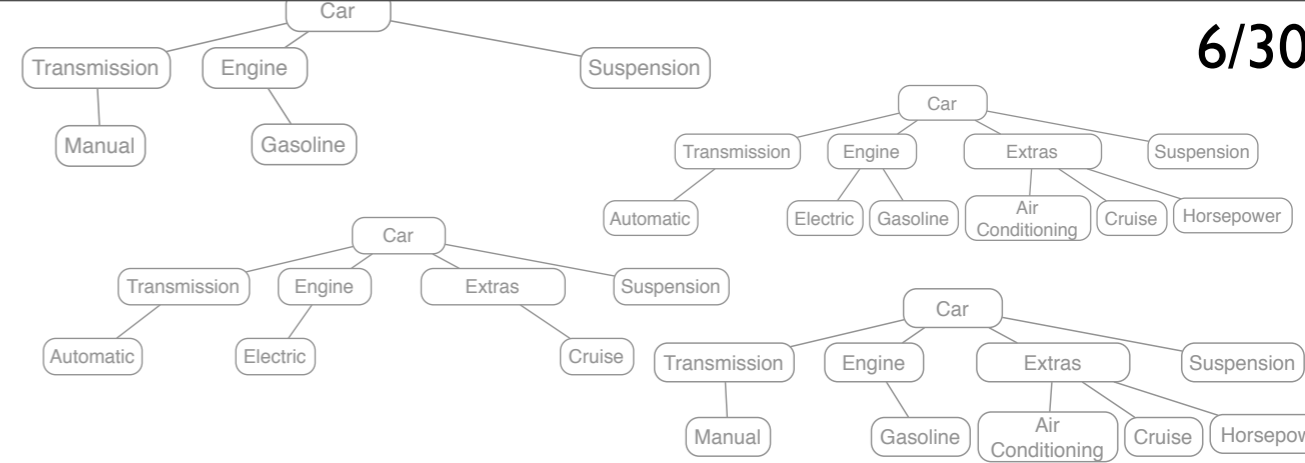
Feature dependencies



some products of the same domain

Feature: unit/increment of functionality

Mining products of the same domain



Code that varies

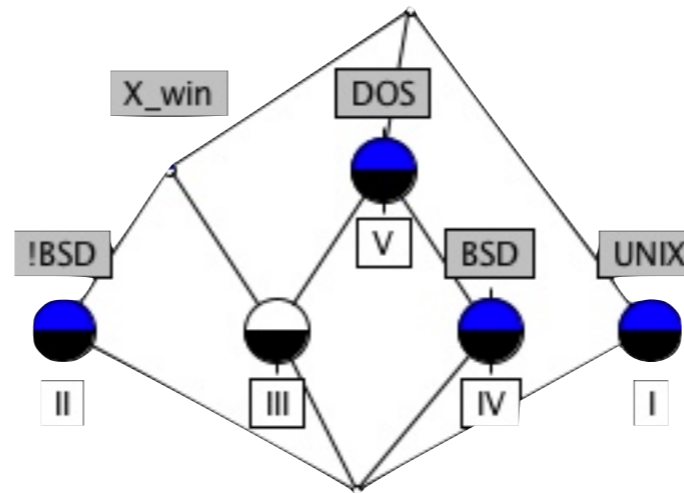
Variables that trigger changes

```

#ifdef UNIX
...I...
#endif
#if defined(X_win)
  && !defined(BSD)
...II...
#endif
#ifdef DOS
  #ifdef X_win
...III...
  #endif
  #ifdef BSD
...IV...
  #endif
...V...
#endif

```

	UNIX	DOS	X_win	BSD	!BSD
I	X				
II			X		X
III		X	X		
IV		X		X	
V		X			



[Snelting TOSEM 96]  

Which parts of code are chosen by which triggers: **selection of variants**

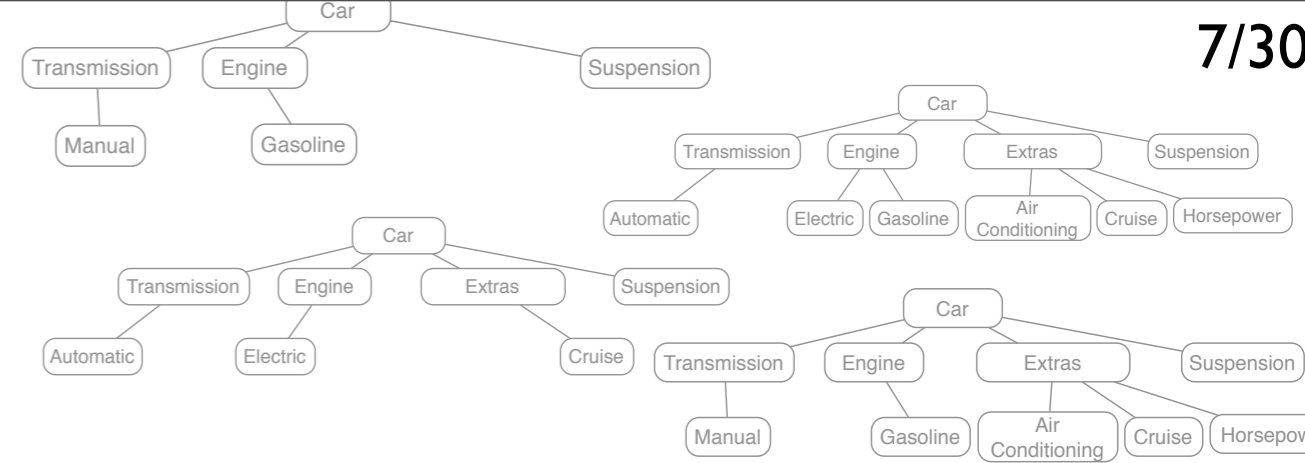
Which triggers are subsumed by other triggers: **dependencies**

X: No relation with features

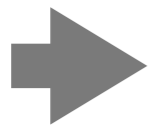
X: Need a specific syntax for the configuration of the products

X: A trigger does not necessarily store the selection of a variant

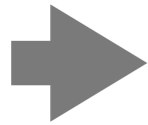
Mining products of the same domain



Name of a Class



Database of classes & their methods (OSS)

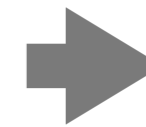


Count same signatures for classes with similar names

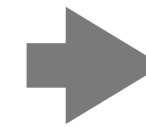
Search Term(s)	Averaged Interface	#
Deck ≈ 50 msec 3, 238 results 323 methods	Deck shuffle():void; dealCard():Card; toString():String; getCard():Card; cardsLeft():int; main(String[]):void;	55 23 19 17 16 10
CreditCard ≈ 40 msec 1, 148 results 229 methods	CreditCard getNumber():String; getCardNumber():String; toString():String; setCardNumber(String):void; getCardType():String; setNumber(String):void;	30 22 20 13 13 12

[Hummel RSSE 10] ↔↕

Missing methods in the class:
common functionality



Some occasional methods:
variants?



X: Business-specific concepts are likely to be missing in the database.

Domain instances
Feature

Variable features

Single vs. Multiple features

Mandatory vs. Optional features

Variability

Feature Diagram

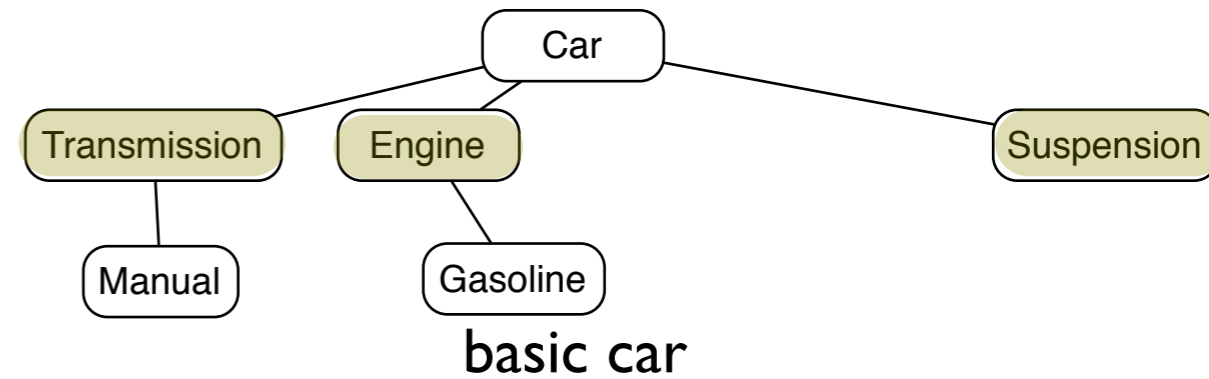
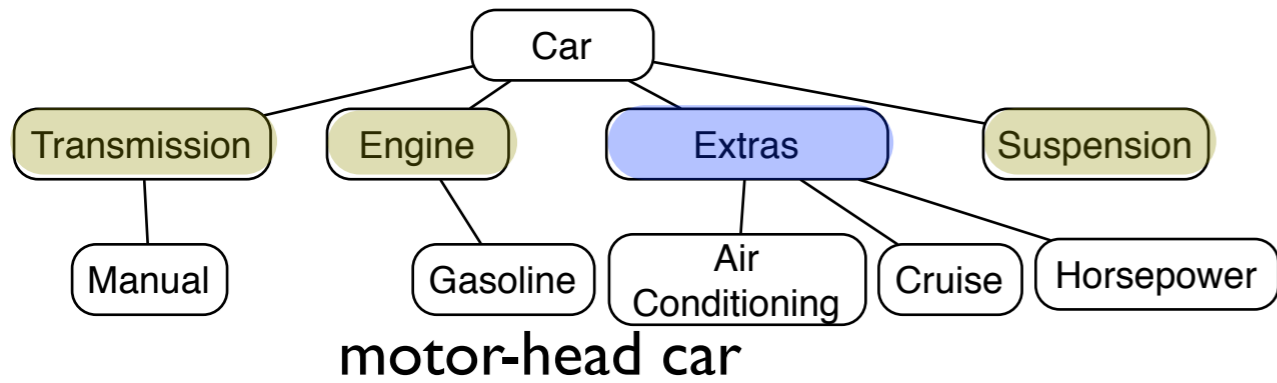
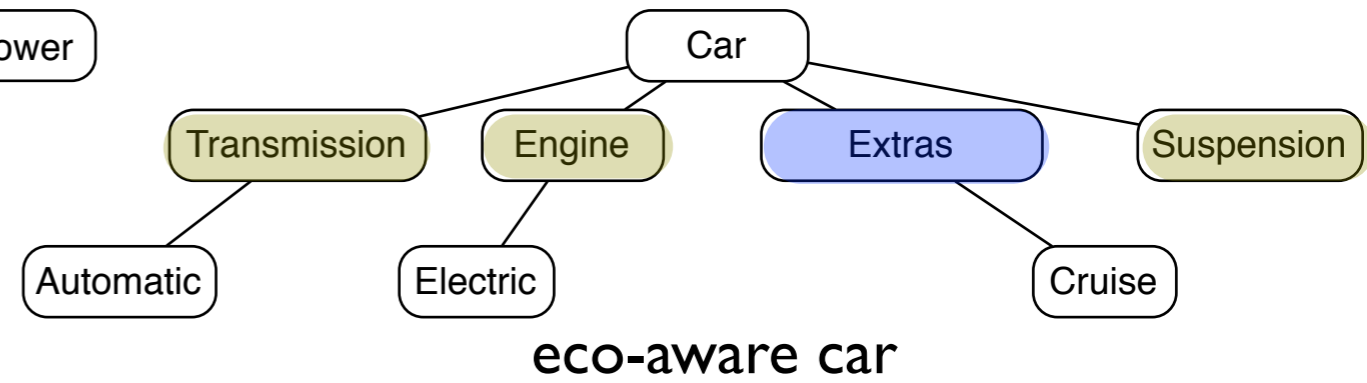
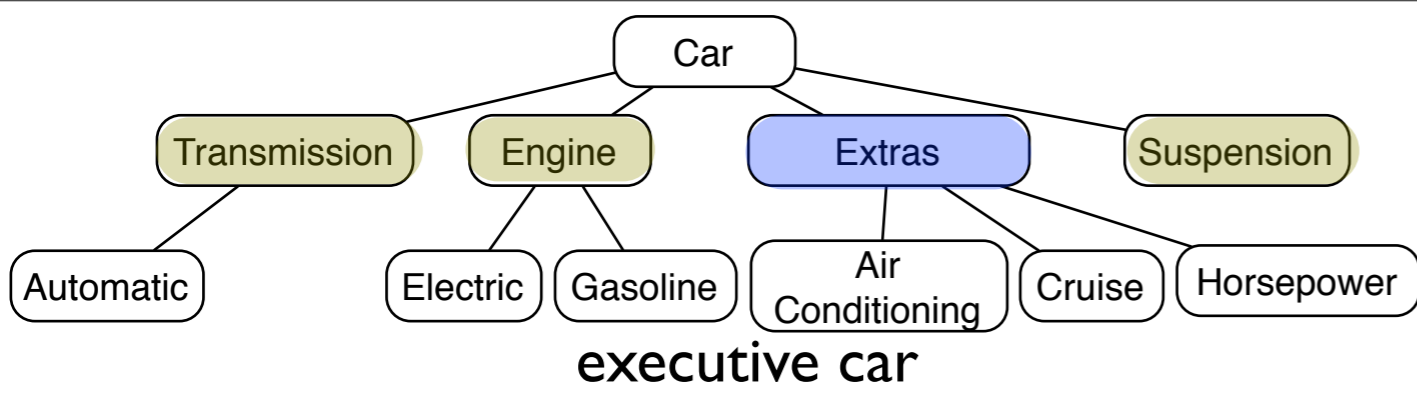
Variants

Optional vs. Alternative variants

Binding

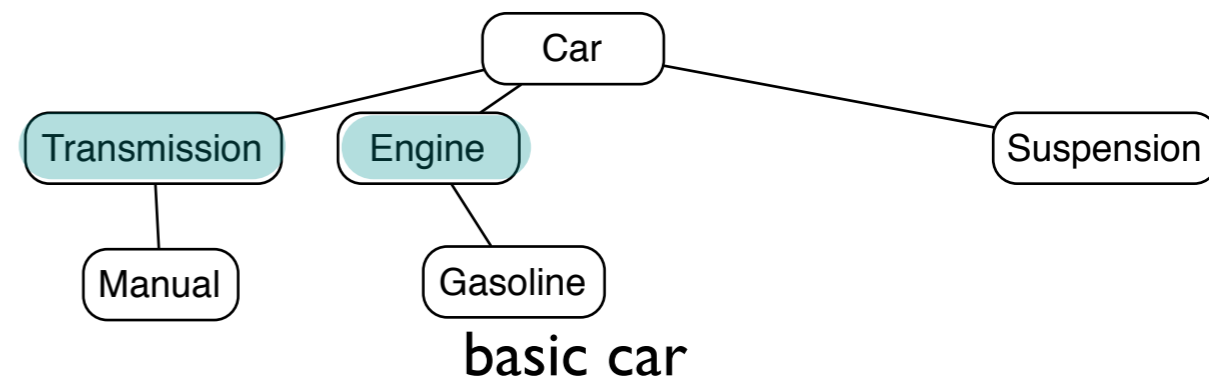
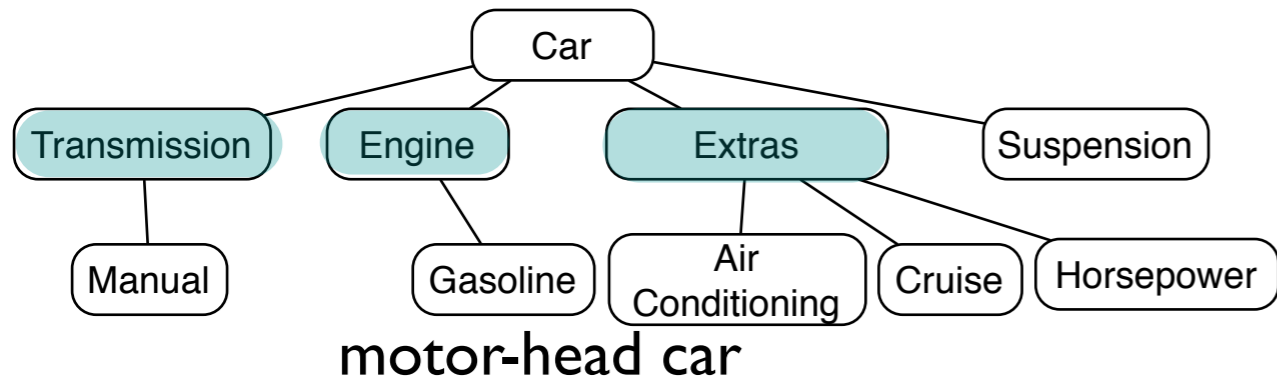
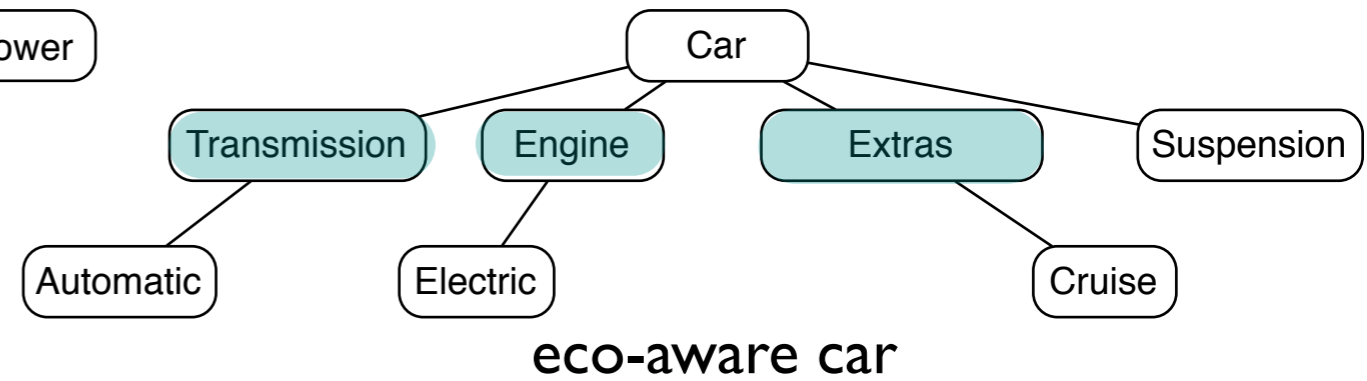
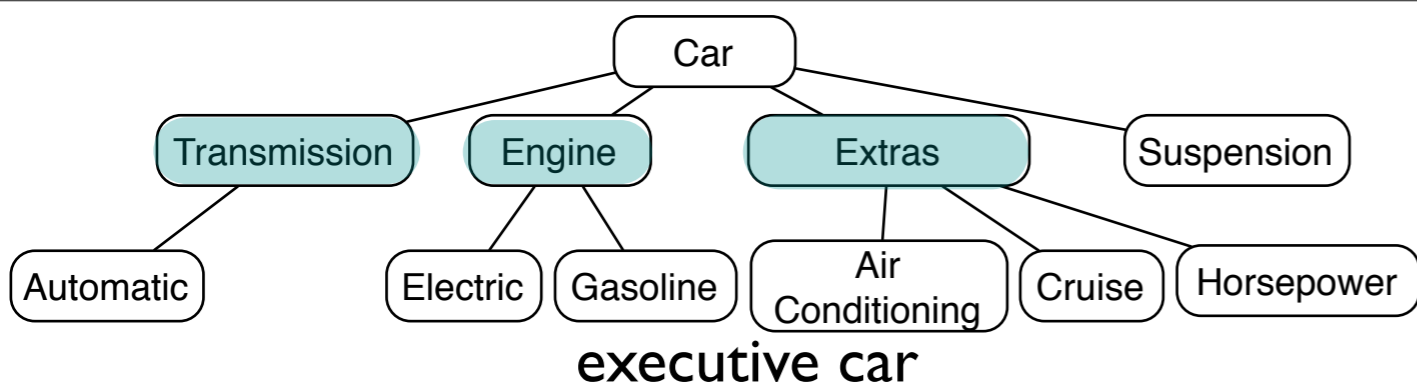
Variation points

Feature dependencies



Feature: unit/increment of functionality

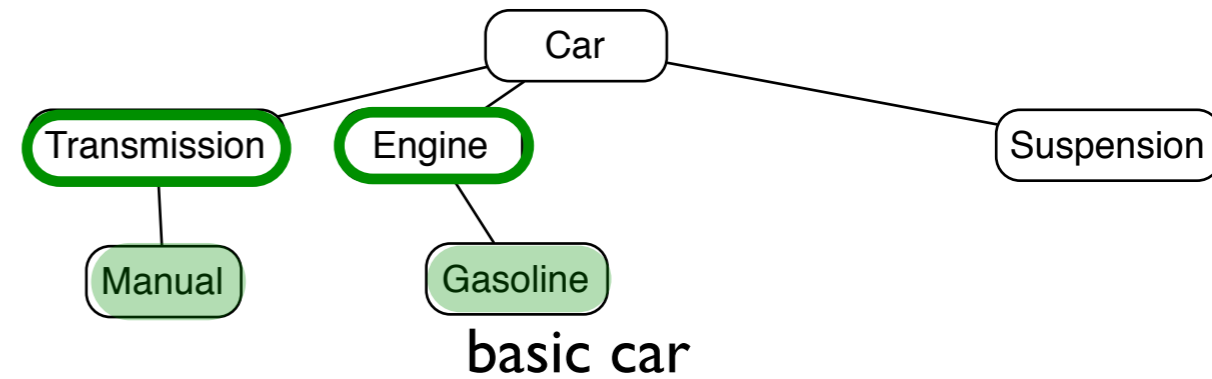
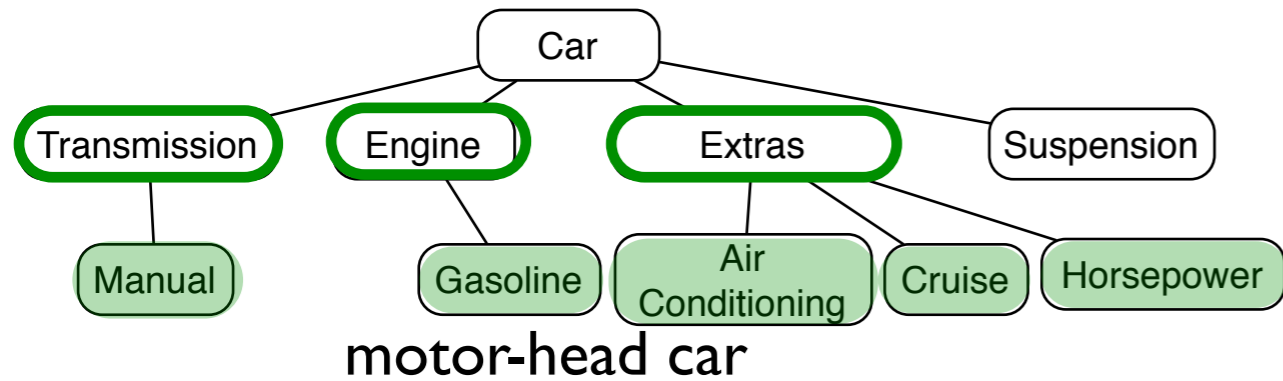
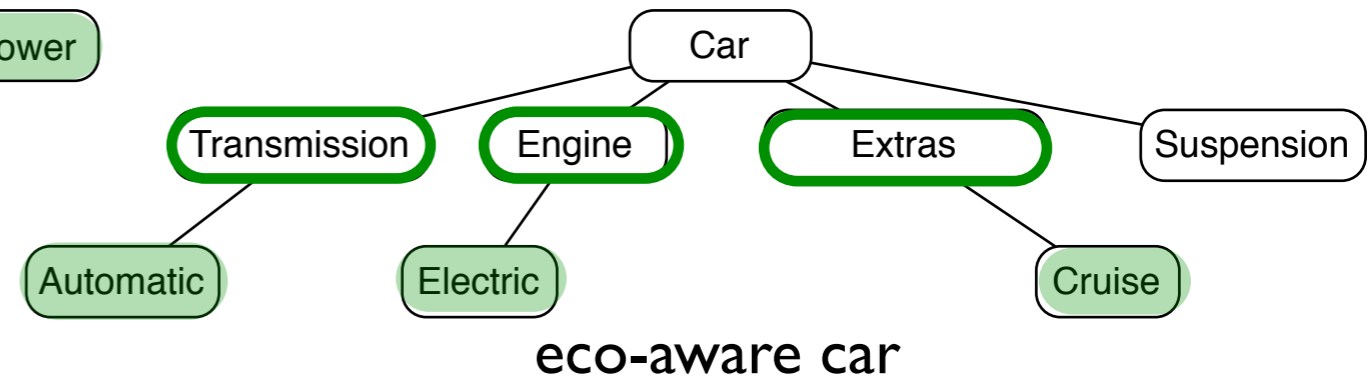
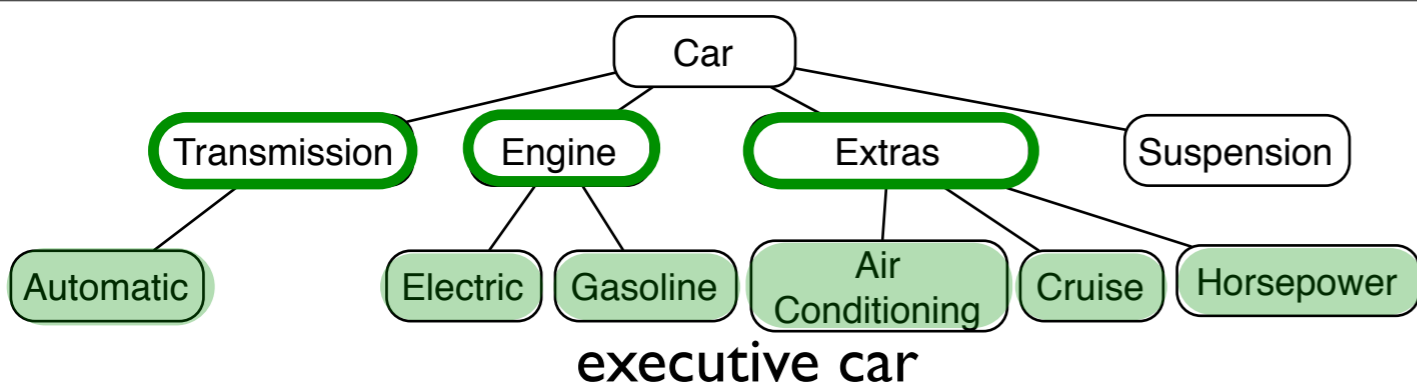
↳ Required in all (**Mandatory**) or in some (**Optional**) products



Feature: unit/increment of functionality

↳ Required in all (Mandatory) or in some (Optional) products

↳ **Variable**: if customization is required

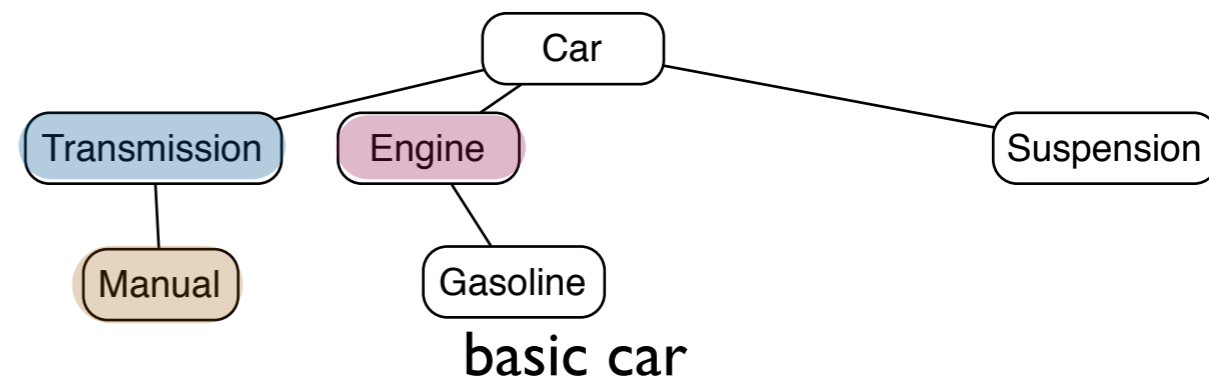
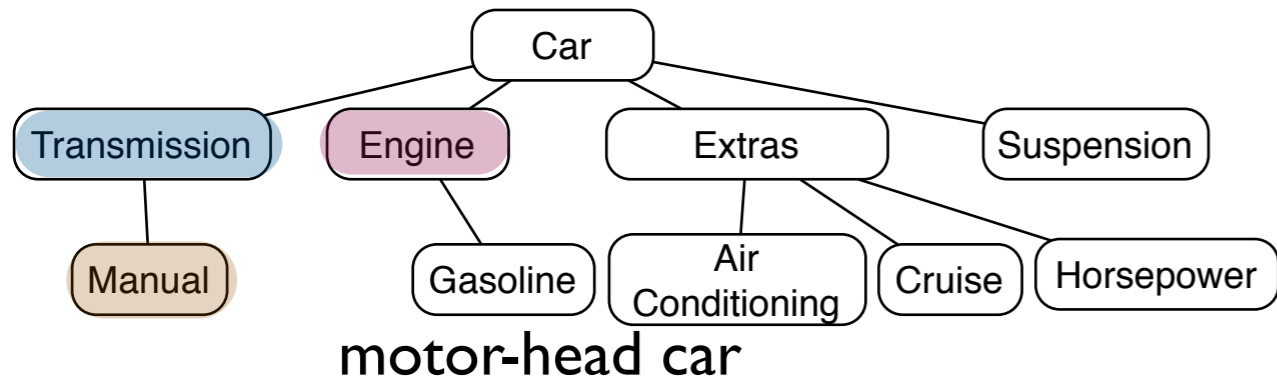
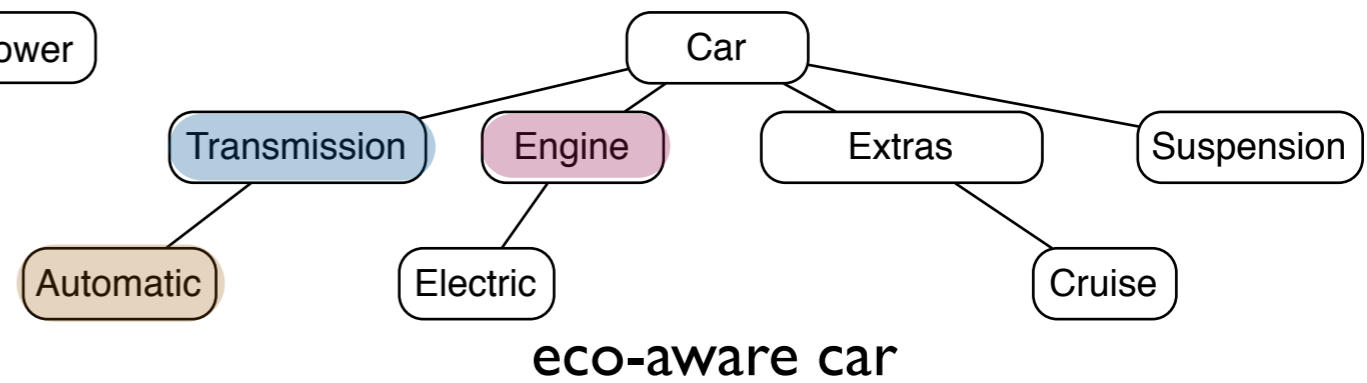
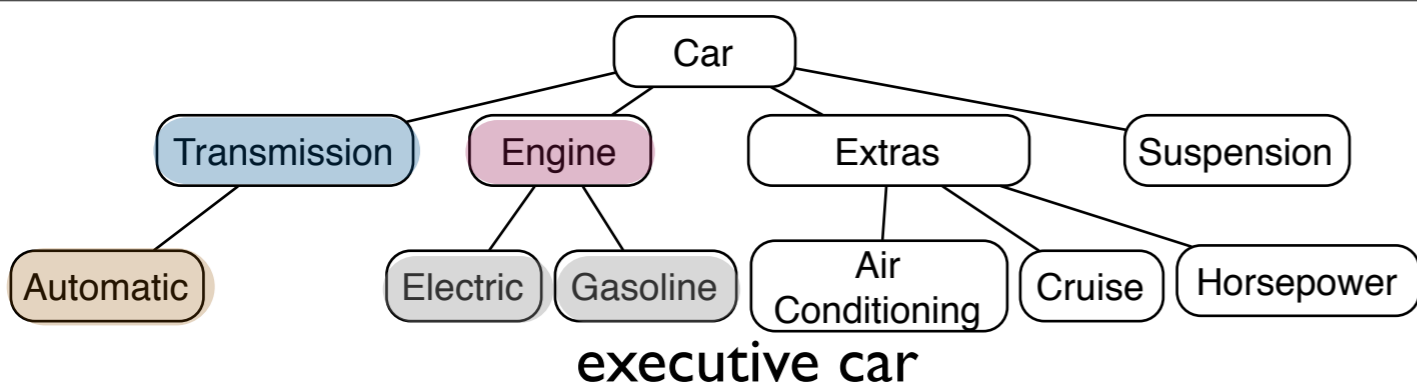


Feature: unit/increment of functionality

↳ Required in all (Mandatory) or in some (Optional) products

↳ Variable: if customization is required

Variant: option available for a variable feature



Feature: unit/increment of functionality

↳ Required in all (Mandatory) or in some (Optional) products

↳ Variable: if customization is required

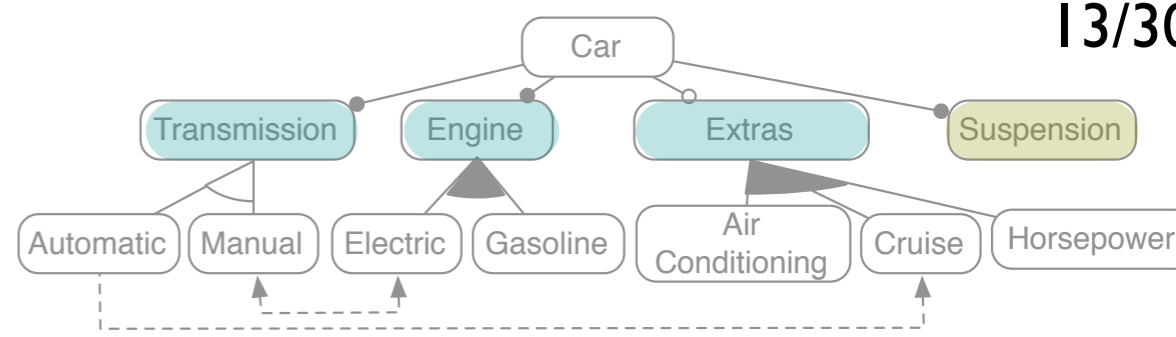
Can a variable feature have several variants?

↳ YES: Multiple / NO: Single

Variant: option available for a variable feature

↳ YES: Mutually inclusive / NO: Mutually exclusive

Mining for variable & mandatory features



Several products of the same domain

Their source code repository

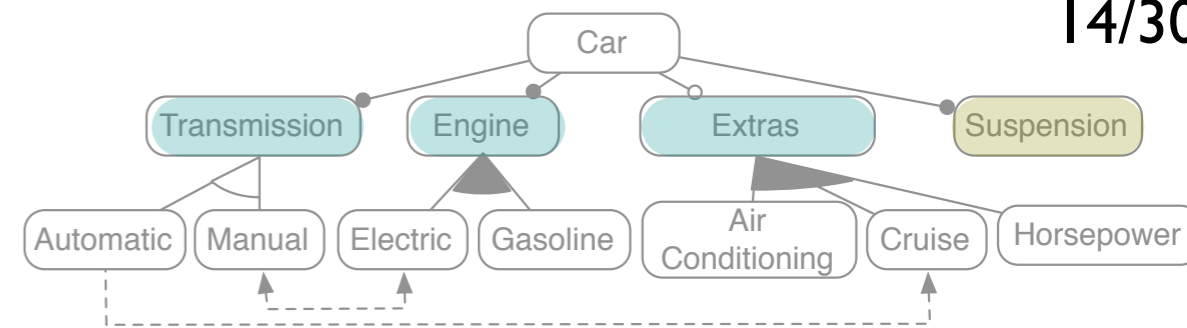
Utilization = Users of the SCE
Length = LOC
Complexity = McCabe
Volatility = $0.7 * \text{changes last year} + 0.3 * \text{exp. changes this year}$
Specificity = LOCs per variation
Mitosis = % clone differences

[Faust SPE 03]

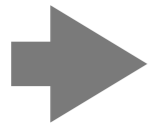
Metrics per SCE - Placement of a SCE:
 mandatory/
 optional features,

- X: Cannot identify dependencies
- X: No support for refactoring

Mining for variable & mandatory features



One product of the domain



clone detection → candidate functions

$$\text{sim}(f1, f2) = \frac{\max(|f1|, |f2|) - \text{LD}(f1, f2)}{|f1|}$$

LD = Levenshtein Distance

FI: Identical Functions (sim=1)

FS: Similar Functions (0<sim<1)

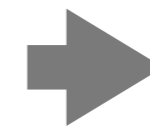
|FI| = 1: Identical correspondence

|FI| > 1: Multiple correspondence

|FI| = 0 & |FS| = 1: Single variant

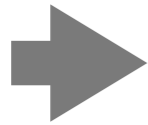
|FI| = 0 & |FS| > 1: Multiple variant

|FI| = 0 & |FS| = 0: No correspondence



Metrics per SCE -
How to merge a variant into the basic product:
mandatory/optional functions, single/multiple variable features

App. w. basic functionality of the domain (core of the line)



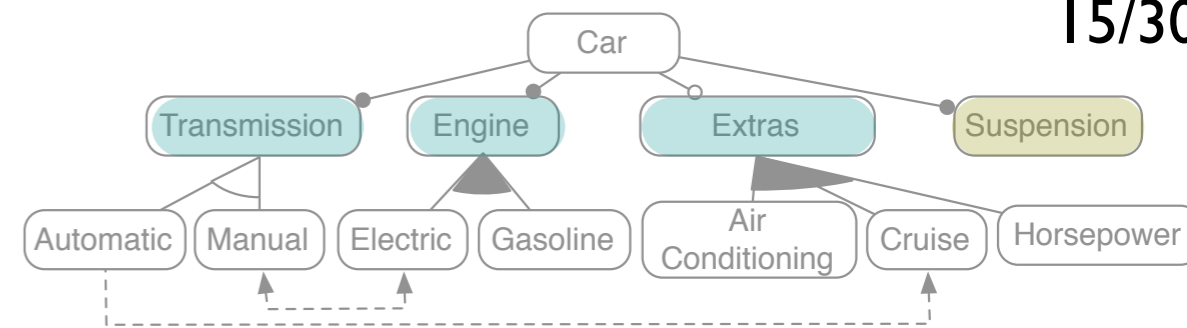
[Mende CSMR 04] 

X: No support for non-corresponding code

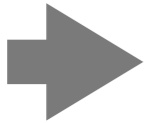
X: Cannot identify dependencies

X: No link to feature diagram

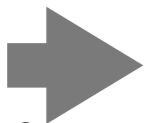
Mining for variable & mandatory features



Several products of the domain



App. w. basic functionality of the domain (core of the line)



$$\text{sim}(s1,s2) = 1 - \frac{\text{LD}(f1,f2)}{\max(|f1|, |f2|)}$$

IF entities are identical -> 'kernel'
 IF variant in all products -> 'variant'
 IF variant in some products -> 'optional'

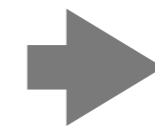
$$\mathbf{Mn} = \frac{M_{n,i} \cap M_{n,j}}{M_{n,i} \cup M_{n,j}} \quad \mathbf{Dn} = \frac{D_{n,i} \cap D_{n,j}}{D_{n,i} \cup D_{n,j}}$$

M_{ni} = Modules for variant i at nesting n

D_{ni} = Dependencies for variant i at nesting n

[Frenzel WCRE 07]

Metrics per SCE - How to merge several variants into a product line



(UML-like diagram): mandatory/optional functions, single/multiple variable features

X: No support for non-corresponding code

X: Can identify mutually exclusive variants, but no feature dependencies

X: No link to feature diagram

Domain instances
Feature

Variable features

Single vs. Multiple features

Mandatory vs. Optional features

Variability

Feature Diagram

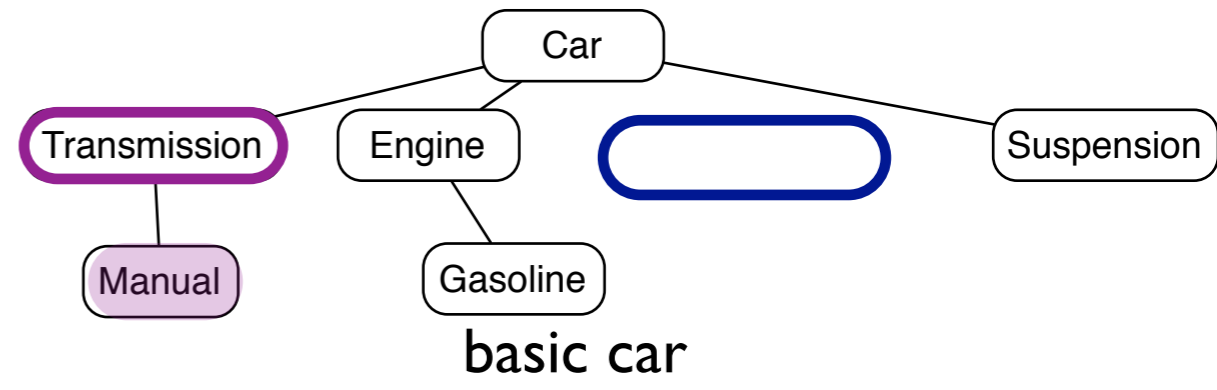
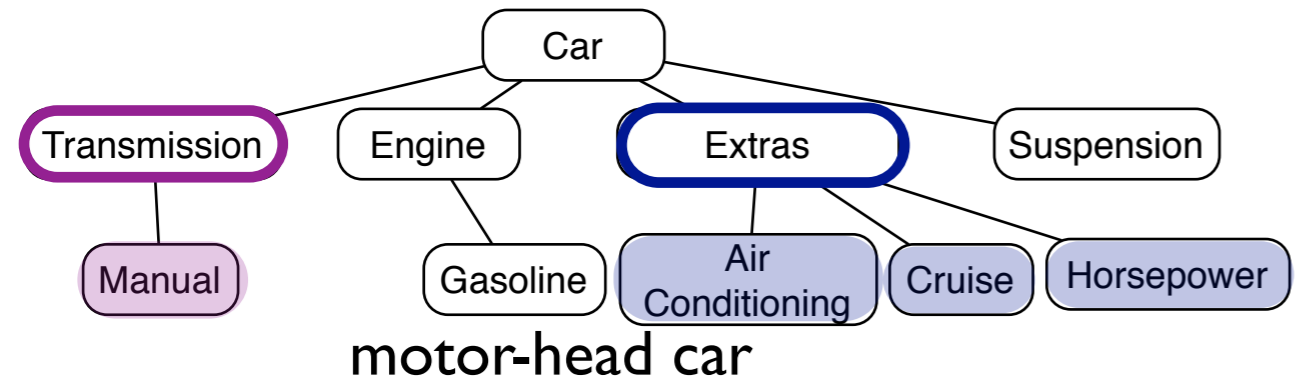
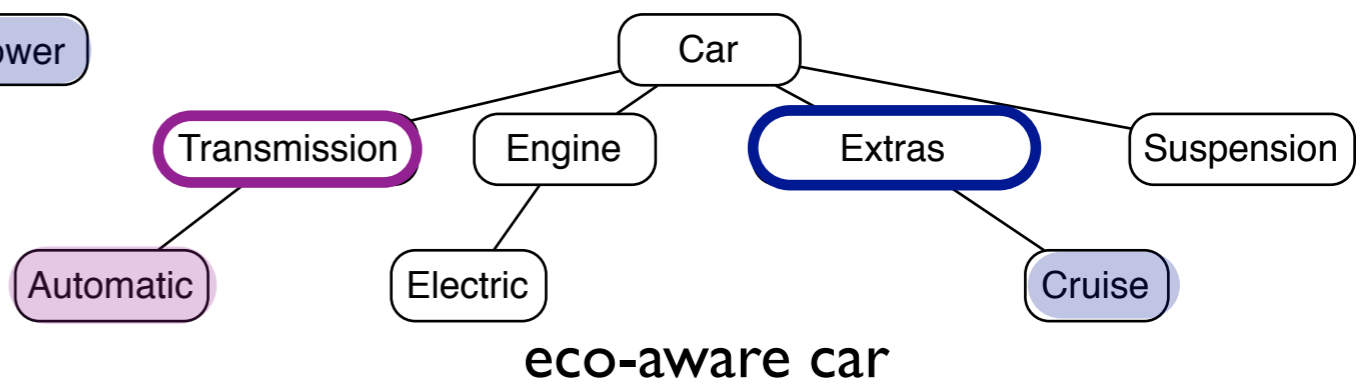
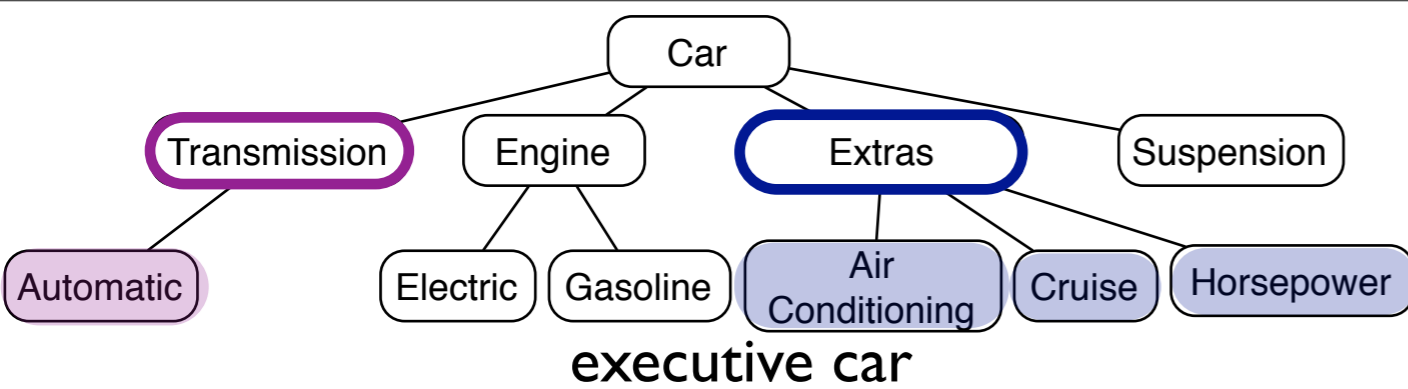
Variants

Optional vs. Alternative variants

Binding

Variation points

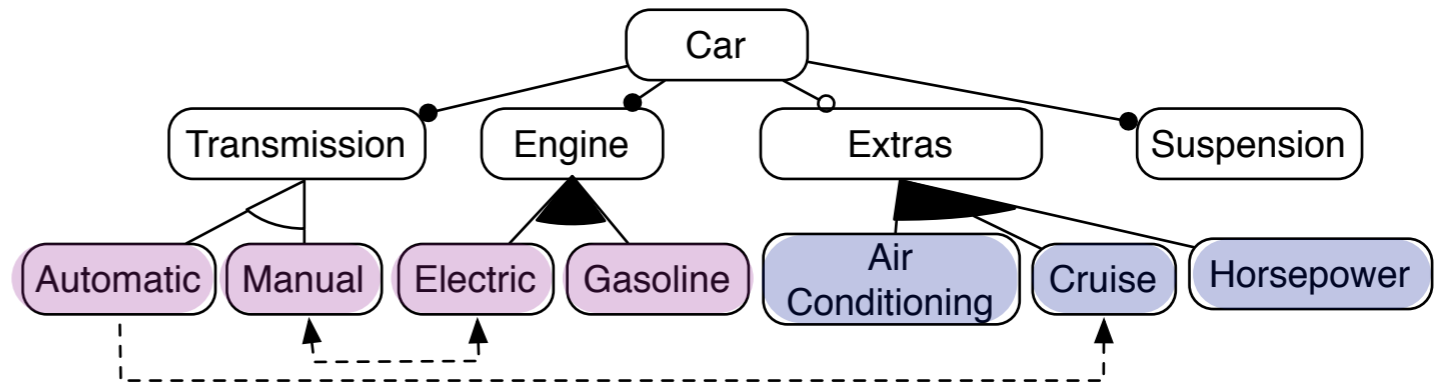
Feature dependencies



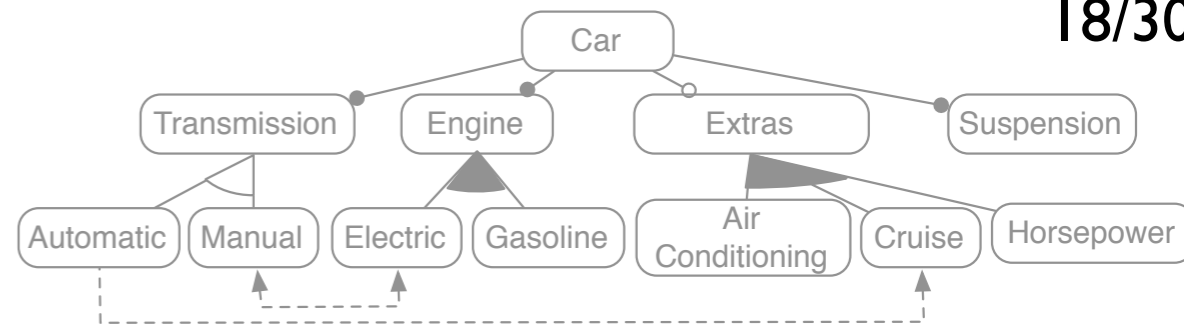
Is the corresponding variable feature required? YES: Optional / NO: Alternative variants

Feature Diagram

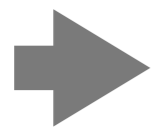
- Optional —○
- Mandatory —●
- Excludes ←- - ->
- Requires - - ->
- Exclusive ▲
- Inclusive ▲



Mining for feature diagrams



Several products of the domain

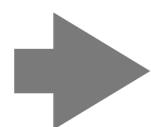


Mapping of entities → operations done over the data (SQL queries)

[0..*] Action <class>
 ![1] extendsAction <ass...>
 [0..1] extendsDispatch...>
 [0..*] actionMethod <...>

Essential features, therefore, are the **minimum** characteristics required to **match a concept** instance. A parent feature cannot exist without all of its essential subfeatures.

Starting SCEs



[0..1] optional
 [1] mandatory
 [0..*] / [1..*] multiple
 ! Essential

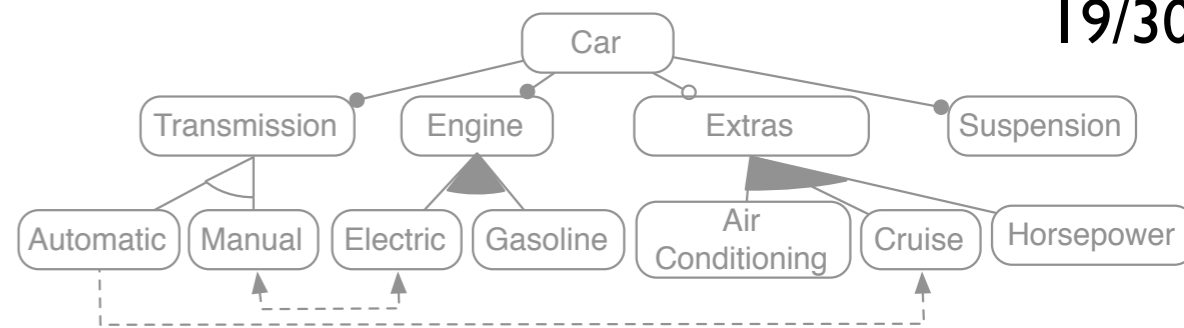
A **mandatory missing** from its parent = config. error.
 A **essential missing** from its parent = no parent feature.

Feature diagram of the domain:
mandatory/optional features, single/multiple variable features

[Antiweckz JASE 09]

- X: Low-level diagram
- X: Cannot identify dependencies
- X: Results depend on the variety of the products of the domain

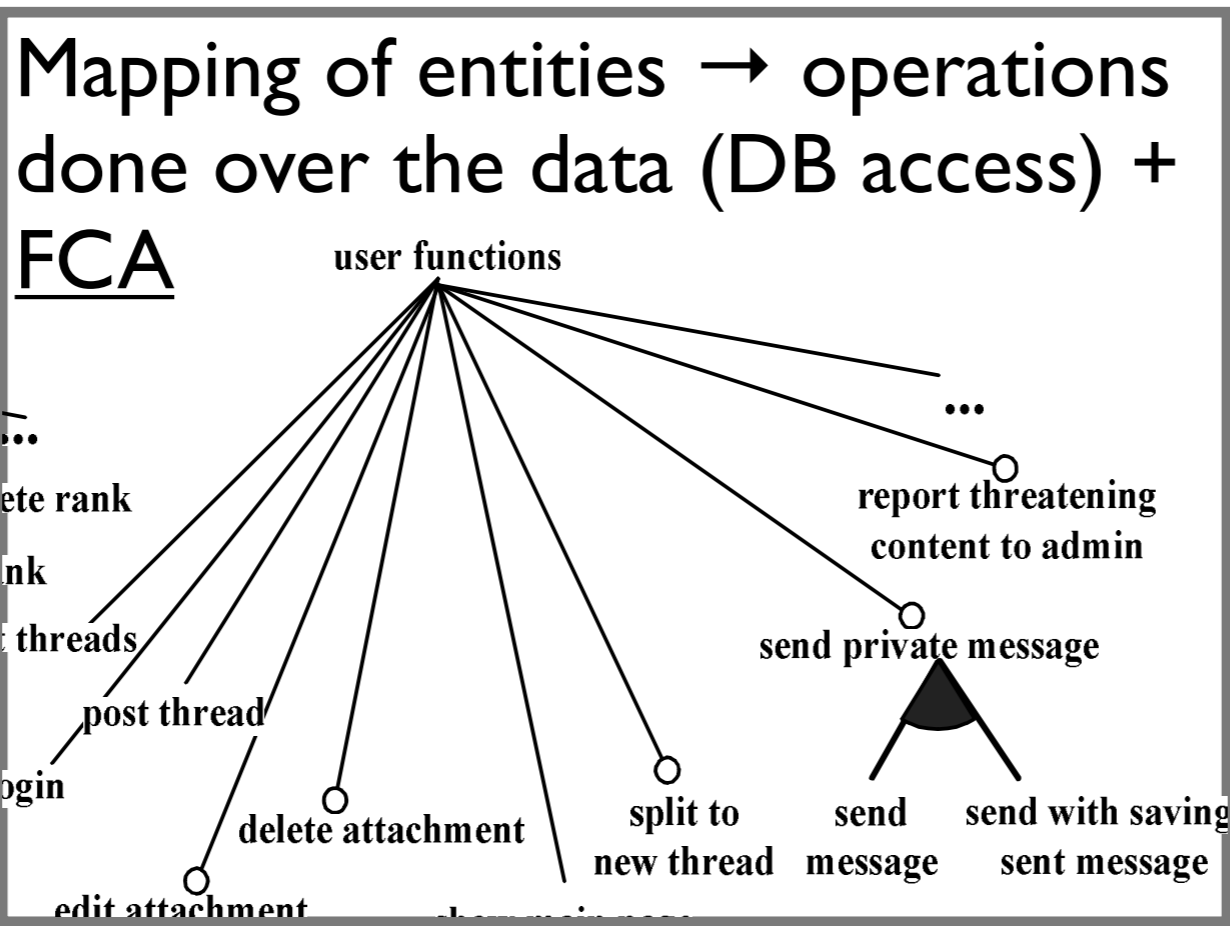
Mining for feature diagrams



Several products of the domain

Mapping of concepts

Consolidate features



[Yang WCRE 09]

Feature diagram of the domain:
mandatory/ optional features,
exclusive/ inclusive variants

X: Cannot identify dependencies

X: Results depend on the variety of the products of the domain

Domain instances
Feature

Variable features

Single vs. Multiple features

Mandatory vs. Optional features

Variability

Feature Diagram

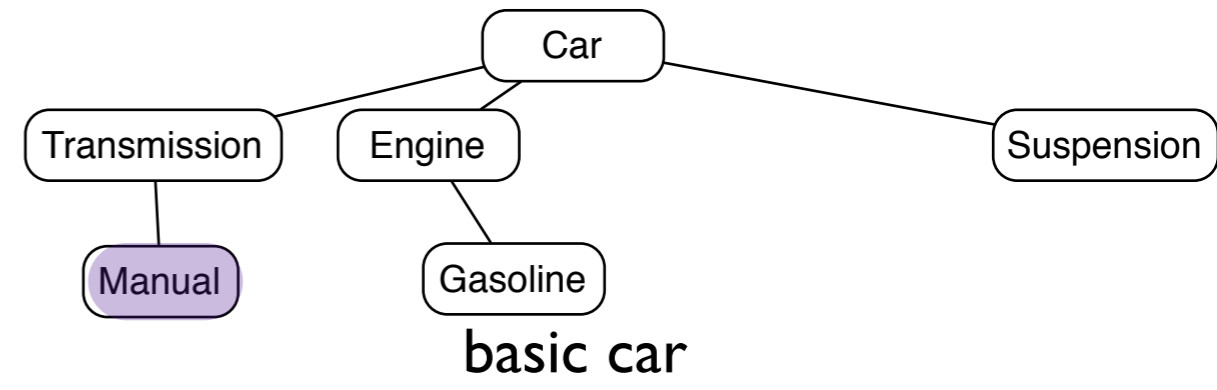
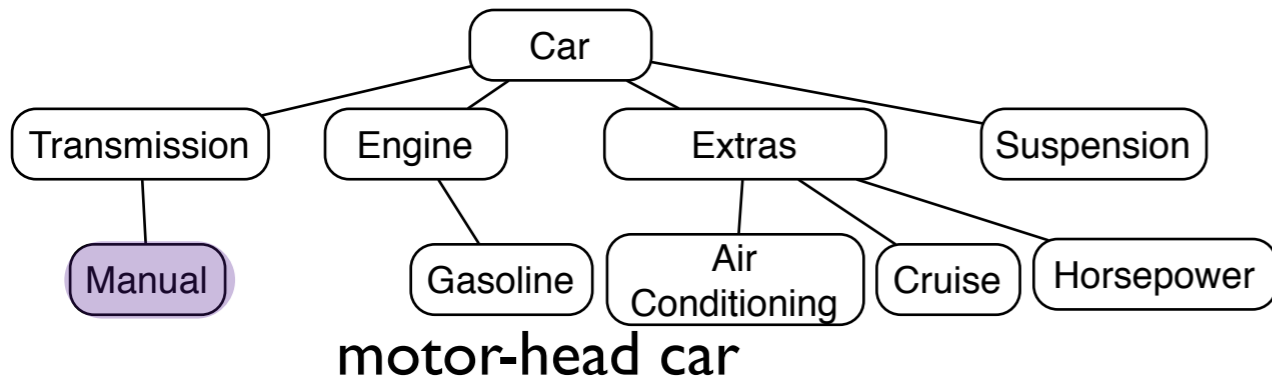
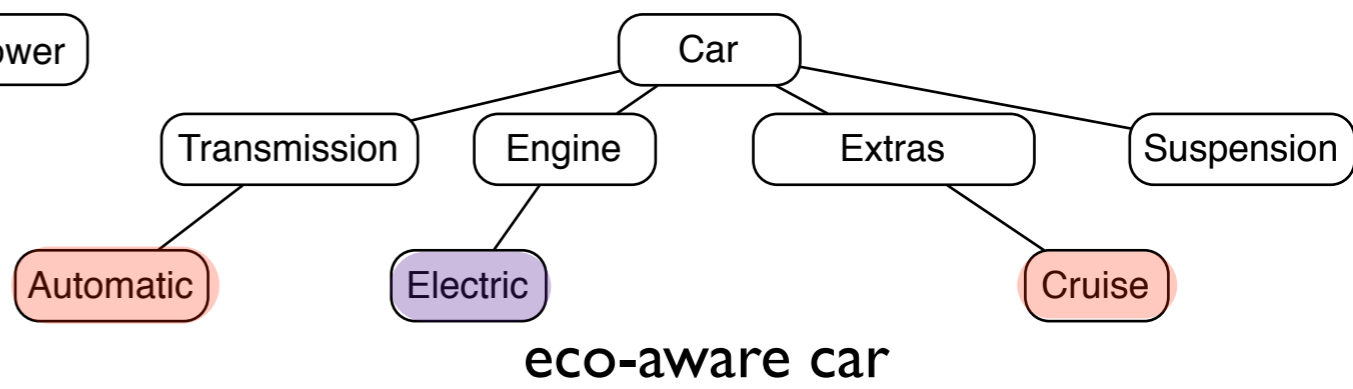
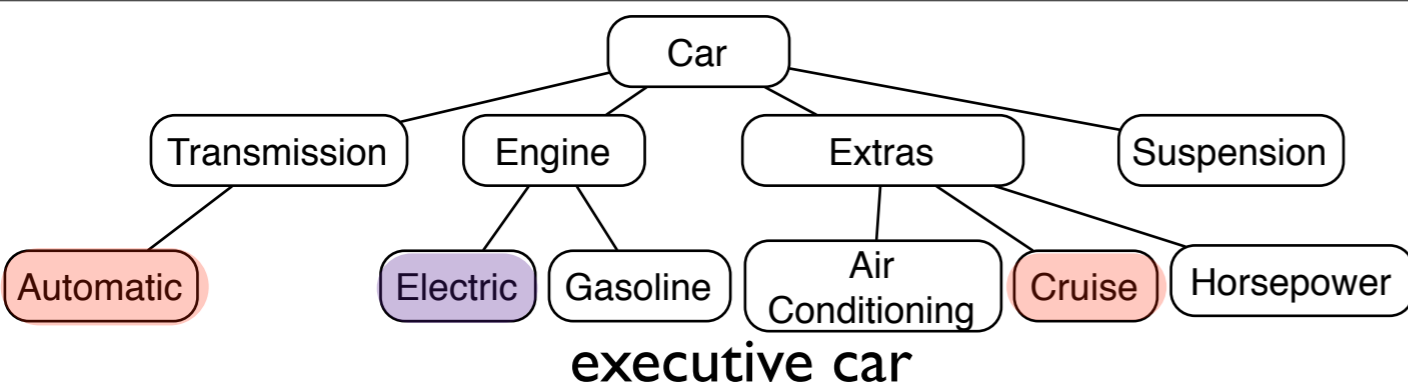
Variants

Optional vs. Alternative variants

Binding

Variation points

Feature dependencies

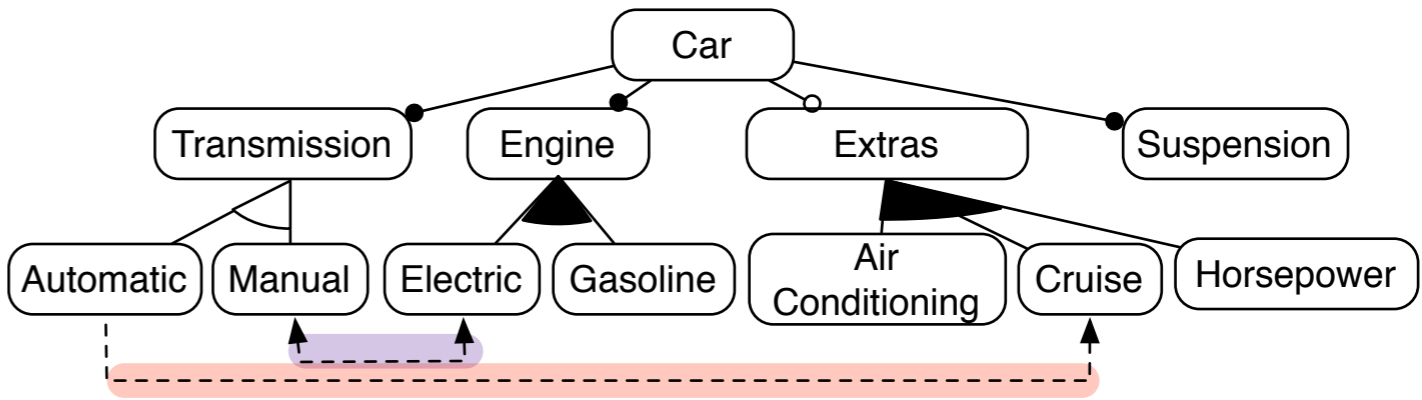


Are there constraints (dependencies) among them ?

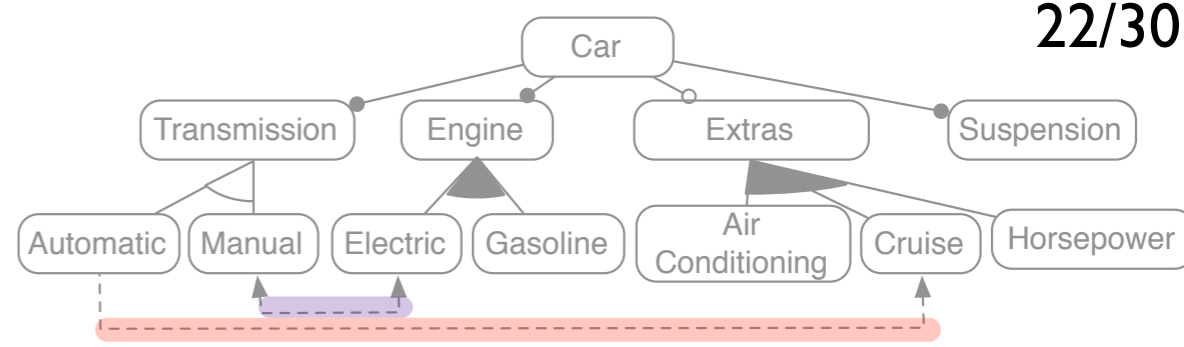
Requires: 1st = 2nd one /
Excludes: 1st or 2nd

Feature Diagram

- Optional —○
- Mandatory —●
- Excludes ←- - - ->
- Requires - - - ->
- Exclusive ▲
- Inclusive ▲

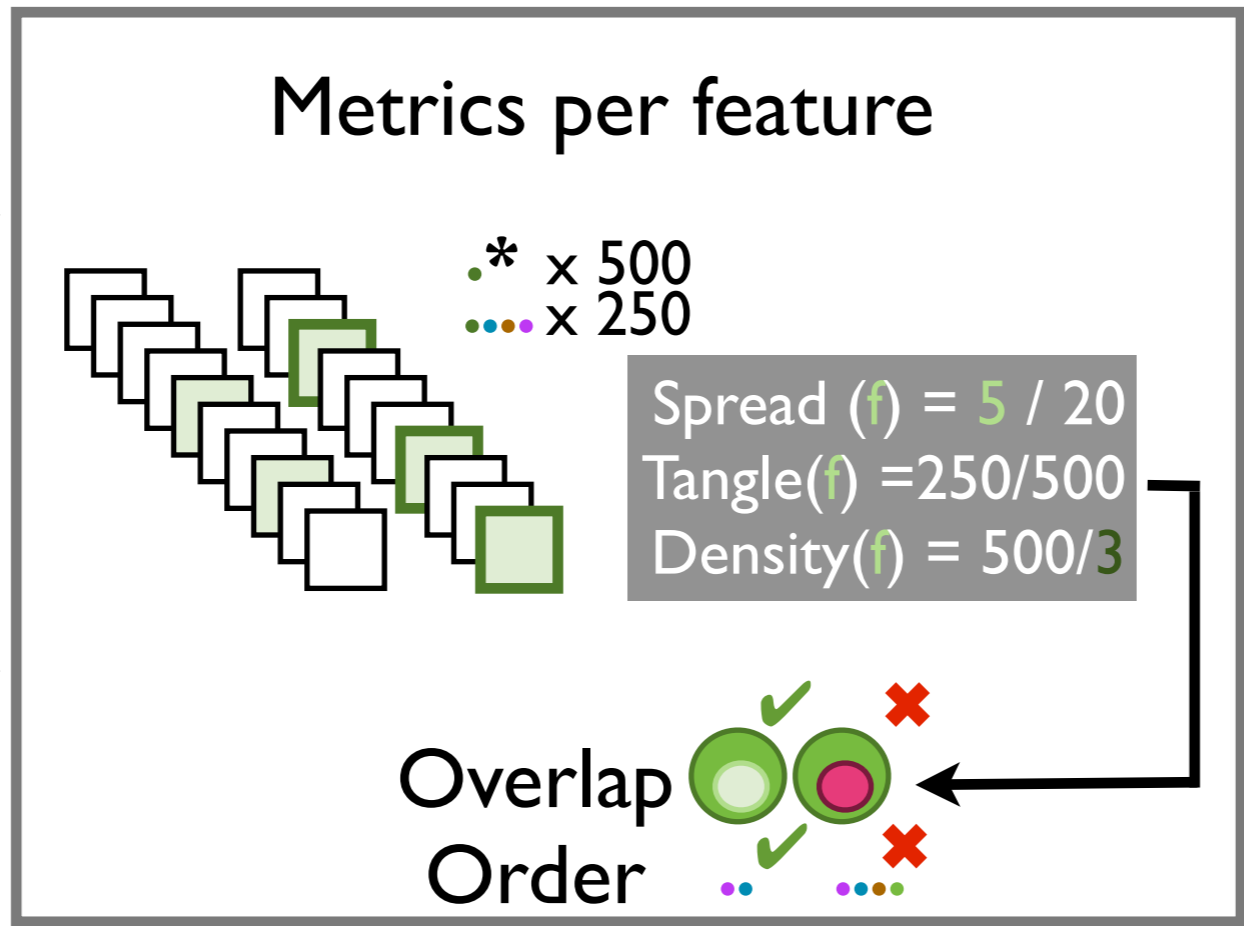


Mining for variability dependencies



Single app. →

Mapping of tokens to features & files to features →

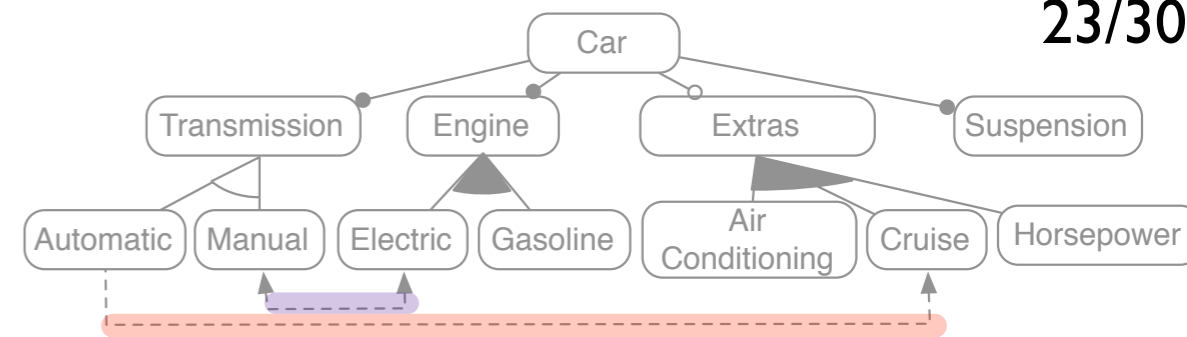


[Lai OOPSLA 99]

→ Variability analysis: (un)acceptable feature tangling

X: Cannot mine for require & exclude relations

Mining for variability dependencies



Overlap Sub-feature Equivalence

Class ↓ Scenario →		A	B	C	D	E	F	G	H
CAboutDlg	0	10					10		
CILLDB	1			3	2	2	2	4	
CILLDBSet	2			4				4	
mailreader	3		3	4	7	1	1	5	1
parsing	4			6				6	
POP3	5		8	10	2	2	2	11	1
CPOP3Dlg	6		4	4	1	4	4	4	
CAPP	7					1	1		
CMainWin	8	3	5	5	3	6	7	5	
CSettingsDlg	9				15				

[Egyed ICSE 01] 

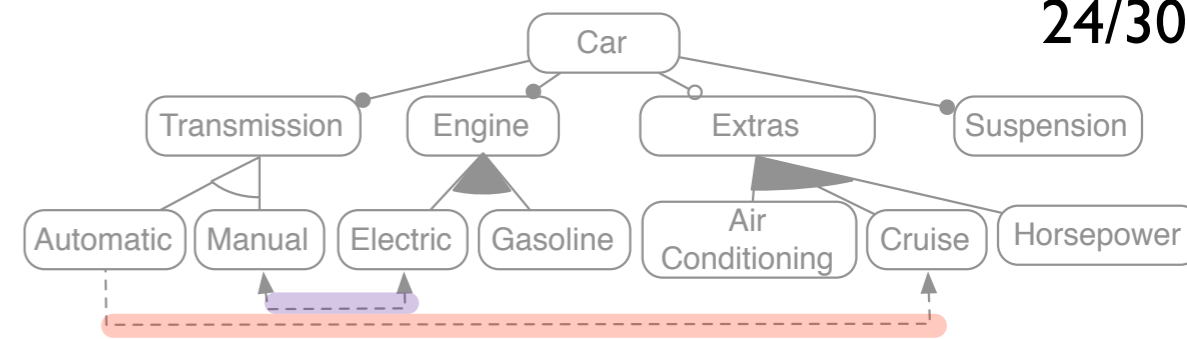
Variability analysis:

Overlap, sub-features, equivalence

Traces of several scenarios (execution of feature)

X: Cannot mine for require & exclude relations

Mining for variability dependencies



Code that varies

Variables representing variants (v_{xm} v_{yn})

Variables representing variation points (vp_x vp_y)

Dependencies
variation points -
variants:

$IF \ !vp_x \rightarrow vp_y$

$IF \ vp_x \rightarrow (vp_y, v_{yn})$

$IF \ (vp_x, v_{xm}) \rightarrow \ !vp_y$

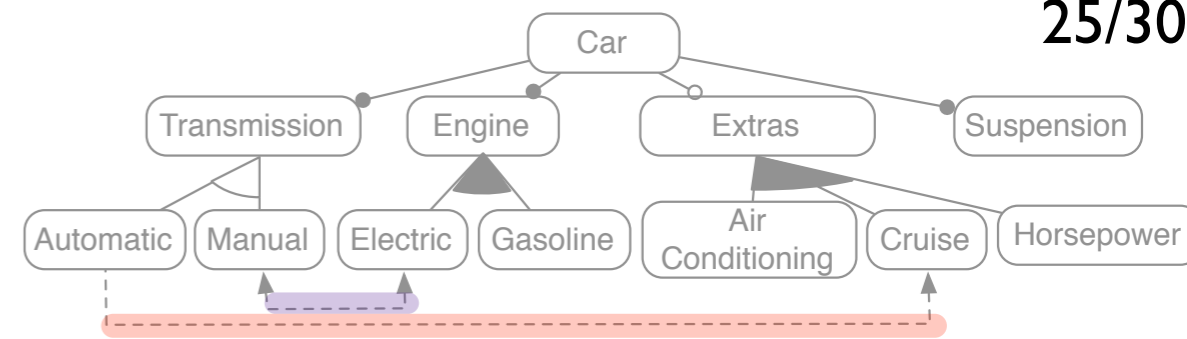
$IF \ (vp_x, v_{xm}) \rightarrow (vp_y, v_{yn})$

[Jaring PhD 05]  

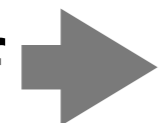
Variability analysis:
configuration dependencies

X: Need a specific syntax for the configuration of the products
X: Just define when the relations occur. No mining proposed.

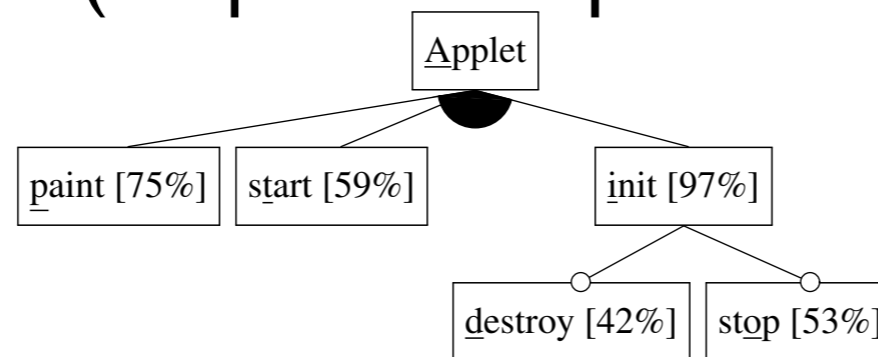
Mining for variability dependencies



Several products of the domain



Association rules = belief
(frequent interpretation)



stop given start [84%]

start given stop [97%]

paint given destroy [88%]

req = $p(A|B)$

exc = $p(!A|B)$

$$p(A|B) = p(AB) / p(A)$$

$$= (A \cap B / A \cup B) / (A)$$

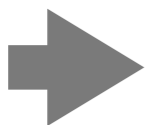
[Czarnecki PLC 08]  

Feature diagram of the domain:

mandatory/optional features,
exclusive/inclusive variants

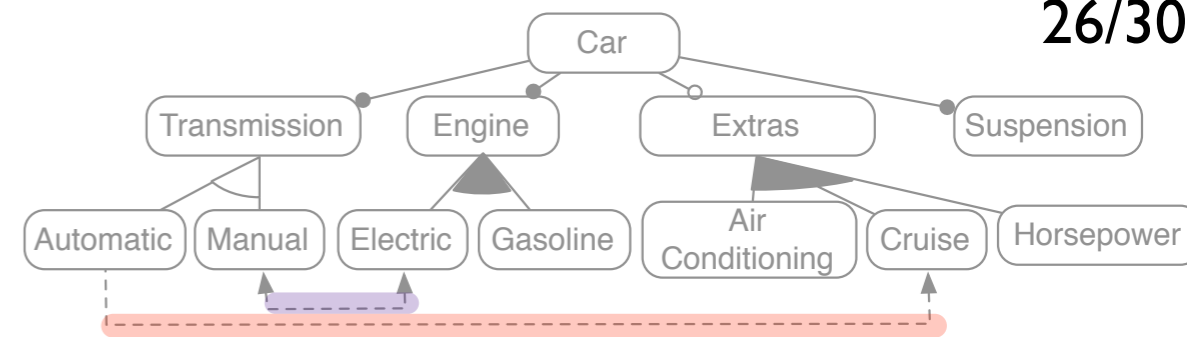


Mapping of SCEs to feature diagram



X: Results depend on the variety of the products of the domain

Mining for variability dependencies

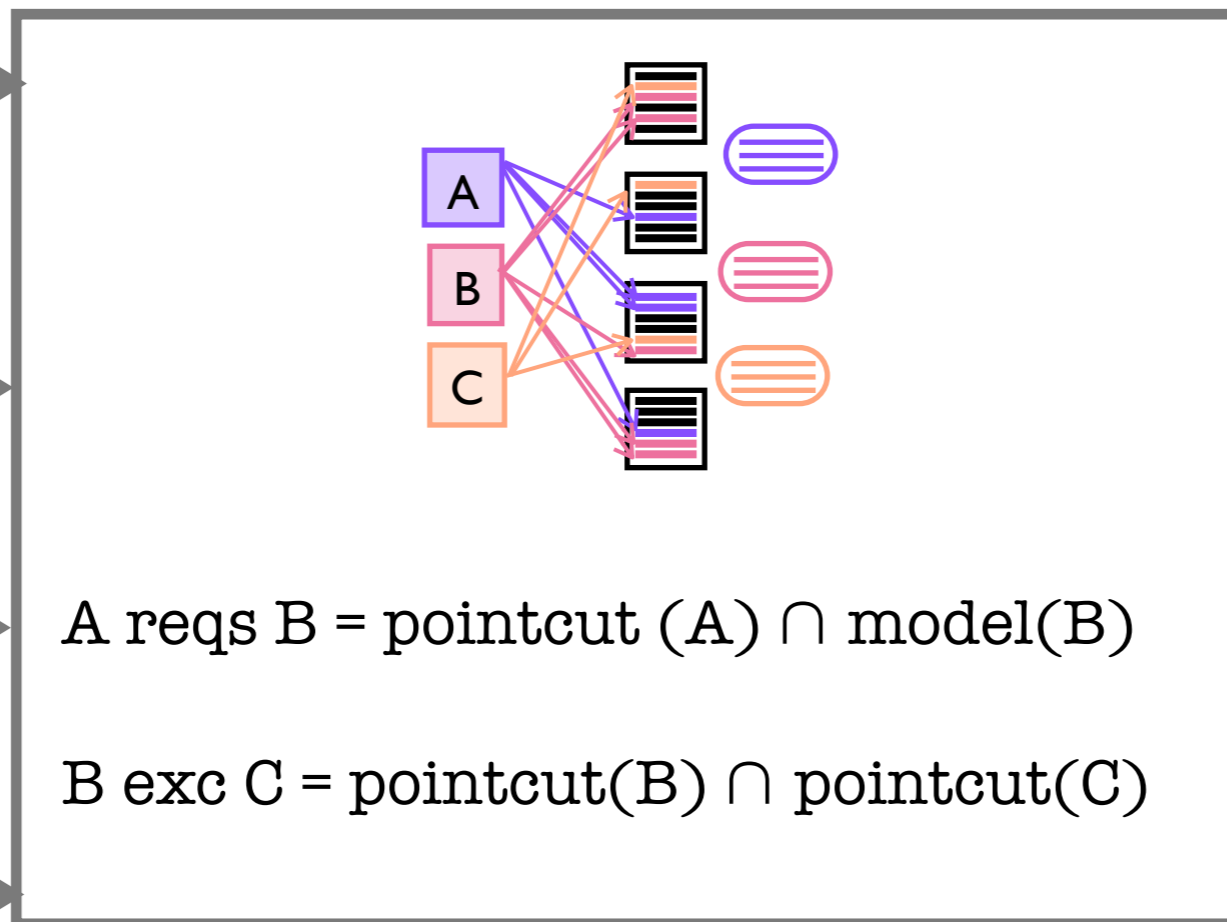


Variable features impl. as aspects

A model of the aspects -SCEs affected

Var. points impl. as pointcuts

Variants impl. as advices



[Parra ECSA 10]  

Feature diagram of the domain:
mandatory/optional features, single/multiple variable features

X: Assumes implementation with aspects.

X: Focuses on composition of features
 i.e. used to detect order or invalid configs.

Domain instances
Feature

Variable features

Single vs. Multiple features

Mandatory vs. Optional features

Variability

Feature Diagram

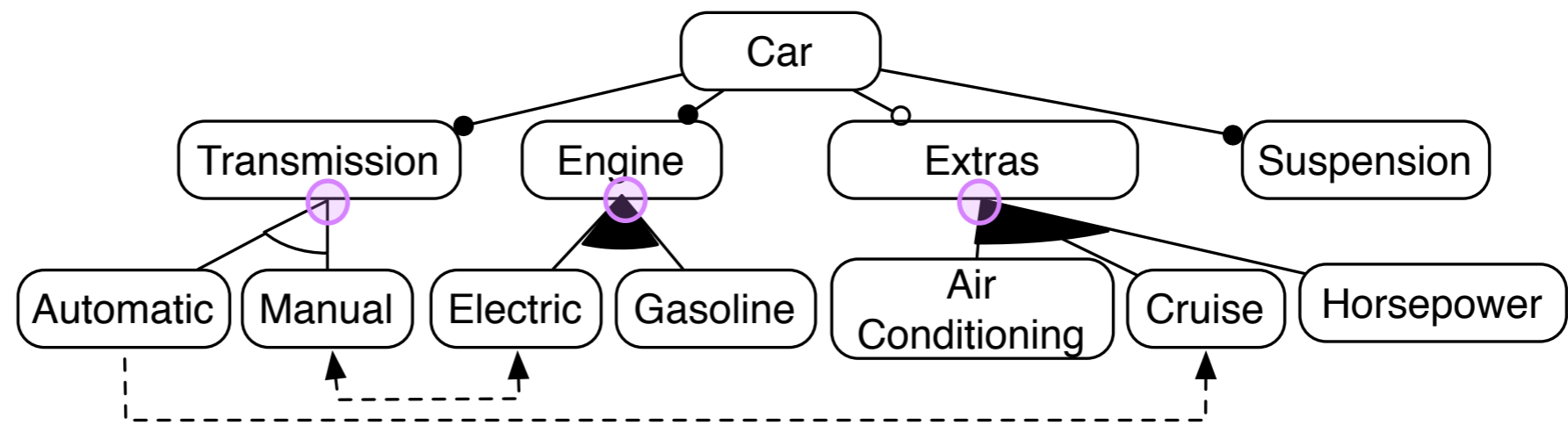
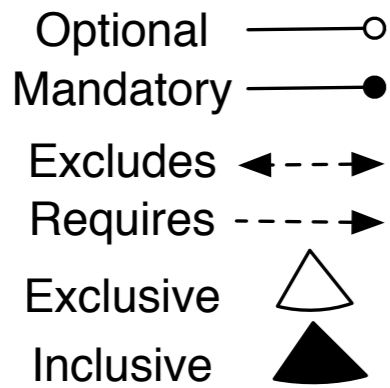
Variants

Optional vs. Alternative variants

Binding

Variation points

Feature dependencies



Feature: unit/increment of functionality

Mandatory: required in all products

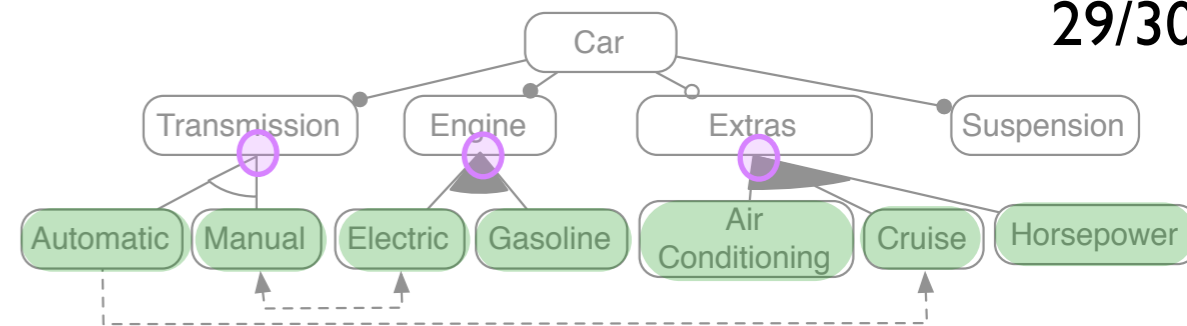
Variable: customization required

Variant: option available for a variable feature

Variation point: placeholder that stores the variant of a variable feature

Binding: assigning a variant to a variation point

Mining for variation points and variants



Several products using the same framework (i.e. of the same domain)

rank & classify SCEs in the framework based on metrics:

IN=instantiations
EX=extensions
OV=overrides
IM=implementations
UM=usages (+ above)

```
class C1 {
    /*IN = 10,EX=0,IM = 0*/

    C1 () { ... }
    /*IN = 10,OV=0,IM = 0*/

    m1_1 (C3 arg1) { ... }
    /*IN = 8,OV = 0,IM = 0*/

    m1_2 () { ... }
    /* IN = 3, OV=0,IM=0*/
```

How to extend a framework:

hotspots/hooks → variation points
cold-spots/templates → mandatory part of the variable feature
users of hooks → variants

[Thummalapenta MSR 08]

X: Requires several applications using the same framework.

Gaps to fill

- Current mining approaches depend on specific implementation techniques
 - E.g. variation points as configuration variables, variable features as framework usage
- The amount of information required sometimes outweigh the benefits
- No support for newcomers (mining to introduce variability to single apps)
 - No support for a-priori analysis of variability decisions (cost-benefit of a variant feature)
 - Implementation issues as business opportunities e.g. “compulsive branching”

A. Lozano

An overview of techniques for detecting software variability
concepts in source code

In Proc. Int'l Workshop on Software Variability Management

pp. 141-150

variability@ER, 2011.