9. Feature Interactions – Handout

Software Product Lines | Thomas Thüm, Timo Kehrer, Elias Kuiter | June 21, 2023

universität ulm

UNIVERSITÄT BERN

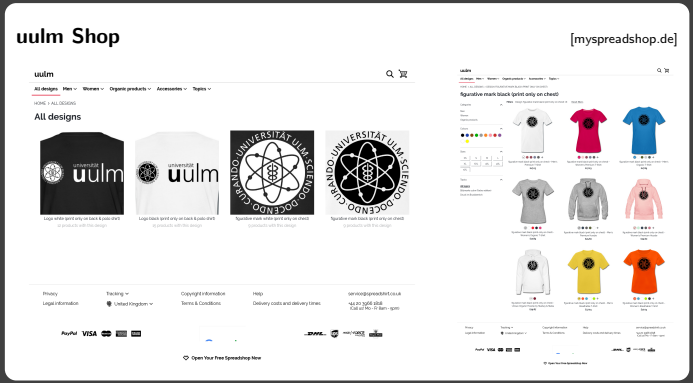
OTTO VON GUERICKE UNIVERSITÄT MAGDEBURG

9. Feature Interactions

An Interaction when Customizing Clothes

Customization of Clothes

- platforms to create and buy clothes
- creator preselects clothes with certain colors
- customization with pictures in different colors
- where is the problem?



An Interaction when Customizing Clothes

The Problem: Unwanted Interaction of Colors

spreadshirt

Wir brauchen eine Entscheidung von Dir

Hallo,

vielen Dank für Deine Bestellung! Bevor wir Deine Ware bedrucken können, brauchen wir eine Entscheidung von Dir.

Bei der Produktion sind wir auf ein kleines Problem gestoßen:

Die Farbe Deines Designs ist der Farbe Deines Produkts zu ähnlich. Wenn der Kontrast zwischen Druck- und Stofffarbe zu gering ausfällt, wird das Design erfahrungsgemäß schlecht zu erkennen sein.

The Solution: Choose Other Colors

Unser Lösungsvorschlag:

Wir können die Farbe des Produkts für Dich ändern. In einzelnen Fällen ist es auch möglich, die Farbe des Designs anzupassen. Oder wir belassen die Bestellung genauso wie sie ist, weil Du Dich bewusst für einen geringen Kontrast entschieden hast.

Bestellung einsehen

Bitte kontaktiere schnellstmöglich unseren Kundenservice, indem Du auf diese Mail antwortest oder anrufst unter 0341 59 400 5900 (Mo-Fr 9-18 Uhr)

Wenn wir bis zum 02.08.2021 keine Antwort erhalten, werden wir Deine Bestellung stornieren und gegebenenfalls den Rechnungsbetrag erstatten.

Wir freuen uns auf Deine Rückmeldung.

Viele Grüße
Dein Team von Spreadshirt

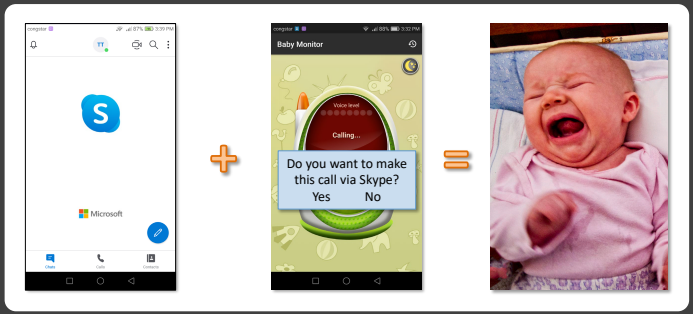
An Interaction when Customizing Clothes

The Problem: Unwanted Interaction of Colors

The Solution: Choose Other Colors

seems that contrast is checked for each order (cf. application engineering)
and not for each published design (cf. domain engineering)

An Interaction of Android Apps



which of those 3.5 million Android apps interact?
where to document?
whom to blame?

Skype vs BabyMonitor

- Skype app installed and used for years
- BabyMonitor installed, carefully tried and used for months
- BabyMonitor can call any other number (i.e., works without internet)
- automatic update of the Skype app
- update causes a question to be asked for every call
- what is the problem?

Feature Interactions

Feature Interaction [Apel et al. 2013, p. 214]

"A feature interaction between two or more features is an emergent behavior that cannot be easily deduced from the behaviors associated with the individual features involved."

for short: interaction

Inadvertent Interaction [Apel et al. 2013, p. 214]

"An inadvertent feature interaction occurs when a feature influences the behavior of another feature in an unexpected way (for example, regarding the expected control flow, program or data state, or visible behavior)."

here simplified as: wanted vs unwanted

Feature-Interaction Problem [Apel et al. 2013, p. 214]

"The feature-interaction problem is to detect, manage, and resolve (inadvertent) feature interactions among features."

Feature Interactions

- detection discussed in next two lectures [Lecture 10 and Lecture 11]
- resolving interactions (see Part II)
- managing interactions (see Part III)

What's Next?

- interactions due to the absence of features
- interactions in source code
- interactions with the base code

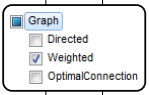
A Common Interaction of Toasters

no interaction for two toasts (i.e., $T_1 \wedge T_2$ shown)
and for no toasts (i.e., $\neg T_1 \wedge \neg T_2$ not shown)

unwanted interaction for one toast
(i.e., $T_1 \wedge \neg T_2$ shown and $\neg T_1 \wedge T_2$ not shown)

Example Interactions with Preprocessors

```
class Edge {
  Node first, second;
//#ifdef Weighted
  int weight;
//#endif
  Edge(Node first, Node second) {...}
  boolean equals(Edge e) {
    return (first == e.first
      && second == e.first
//#ifndef Directed
      || first == e.second
      && second == e.first
      ) && weight == e.weight;
//#endif
  }
  void testEquality() {
    Node a = new Node();
    Node b = new Node();
    Edge e = new Edge(a, b);
    Assert.assertTrue(e.equals(e));
} }
```



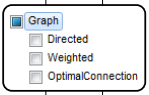

```
class Edge {
  Node first, second;

  int weight;

  Edge(Node first, Node second) {...}
  boolean equals(Edge e) {
    return (first == e.first
      && second == e.first

      || first == e.second
      && second == e.first
      ) && weight == e.weight;

  }
  void testEquality() {
    Node a = new Node();
    Node b = new Node();
    Edge e = new Edge(a, b);
    Assert.assertTrue(e.equals(e));
} }
```

No Interaction?

- configuration for undirected, weighted edges
- product can be compiled
- what is the problem?

Example Interactions with Preprocessors

```
class Edge {
  Node first, second;
//#ifdef Weighted
  int weight;
//#endif
  Edge(Node first, Node second) {...}
  boolean equals(Edge e) {
    return (first == e.first
      && second == e.first
//#ifndef Directed
      || first == e.second
      && second == e.first
      ) && weight == e.weight;
//#endif
  }
  void testEquality() {
    Node a = new Node();
    Node b = new Node();
    Edge e = new Edge(a, b);
    Assert.assertTrue(e.equals(e));
} }
```



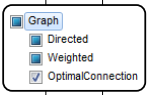

```
class Edge {
  Node first, second;


  Edge(Node first, Node second) {...}
  boolean equals(Edge e) {
    return (first == e.first
      && second == e.first

      || first == e.second
      && second == e.first
      ) && weight == e.weight  ;

  }
  void testEquality() {
    Node a = new Node();
    Node b = new Node();
    Edge e = new Edge(a, b);
    Assert.assertTrue(e.equals(e));
} }
```

Static Interaction

- configuration for undirected, unweighted edges
- product cannot be compiled due to static feature interaction
- field weight used for undirected edges but defined in feature Weighted
- occurs whenever features Directed and Weighted are both not selected

Example Interactions with Preprocessors

```
class Edge {
  Node first, second;
//#ifdef Weighted
  int weight;
//#endif
  Edge(Node first, Node second) {...}
  boolean equals(Edge e) {
    return (first == e.first
      && second == e.first
//#ifndef Directed
      || first == e.second
      && second == e.first
      ) && weight == e.weight;
//#endif
  }
  void testEquality() {
    Node a = new Node();
    Node b = new Node();
    Edge e = new Edge(a, b);
    Assert.assertTrue(e.equals(e));
} }
```



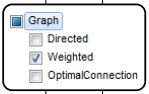

```
class Edge {
  Node first, second;

  int weight;

  Edge(Node first, Node second) {...}
  boolean equals(Edge e) {
    return (first == e.first
      && second == e.first

  }
  void testEquality() {
    Node a = new Node();
    Node b = new Node();
    Edge e = new Edge(a, b);
    Assert.assertTrue(e.equals(e));
} }
```

Other Static Interaction

- configuration for directed, weighted edges
- product cannot be compiled due to static feature interaction
- semicolon and bracket missing for every configuration with feature Directed
- feature Directed has inadvertent interaction with base code

Example Interactions with Preprocessors

```
class Edge {
  Node first, second;
//#ifdef Weighted
  int weight;
//#endif
  Edge(Node first, Node second) {...}
  boolean equals(Edge e) {
    return (first == e.first
      && second == e.first
//#ifndef Directed
      || first == e.second
      && second == e.first
      ) && weight == e.weight;
//#endif
  }
  void testEquality() {
    Node a = new Node();
    Node b = new Node();
    Edge e = new Edge(a, b);
    Assert.assertTrue(e.equals(e));
} }
```




```
class Edge {
  Node first, second;

  int weight;

  Edge(Node first, Node second) {...}
  boolean equals(Edge e) {
    return (first == e.first
      && second == e.first

      || first == e.second
      && second == e.first
      ) && weight == e.weight;

  }
  void testEquality() {
    Node a = new Node();
    Node b = new Node();
    Edge e = new Edge(a, b);
    Assert.assertTrue(e.equals(e));
} }
```

Dynamic Interaction?

- again: configuration for directed, weighted edges
- product can be compiled but test fails
- not a static interaction
- defect in the base code
- no interaction at all

Detection of Interactions

- static interactions [Lecture 10]
- dynamic interactions [Lecture 11]
- next: interactions of more than two features

Higher-Order Interactions

Kinds of Interactions

- wanted and unwanted interactions
- static and dynamic interactions
- one-wise interactions (interaction of features with the base code and faulty features)
- pair-wise interactions (between two features)
- higher-order interactions (between more than two features)

Variability Bug Database [VBDb]

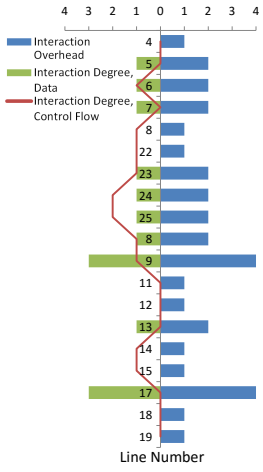
- database of known feature interactions
- operating system Linux (43 interactions)
- web server Apache (23)
- system tool Busybox (18)
- 3D printer firmware Marlin (14)

Patterns of Feature Interactions [VBDb]

L	precondition	M	B	A	Σ
21	**some enabled:**	**9**	**7**	**14**	**49**
5	a	6	3	7	21
10	$a \wedge b$	3	3	5	21
5	$a \wedge b \wedge c$		1		6
1	$a \wedge b \wedge c \wedge d \wedge e$				1
20	**some-enabled-one-disabled:**	**4**	**11**	**10**	**45**
3	$\neg a$	1	6	10	20
13	$a \wedge \neg b$	3	4		20
3	$a \wedge b \wedge \neg c$		1		4
1	$a \wedge b \wedge c \wedge d \wedge \neg e$				1
2	**other configurations:**	**1**		**1**	**4**
1	$\neg a \wedge \neg b$				1
	$a \wedge \neg b \wedge \neg c$	1			1
1	$a \wedge \neg b \wedge \neg c \wedge \neg d \wedge \neg e$			1	2
43	**TOTAL**	**14**	**18**	**23**	**98**

Interaction on Data and Control Flow [Meinicke et al. 2016]

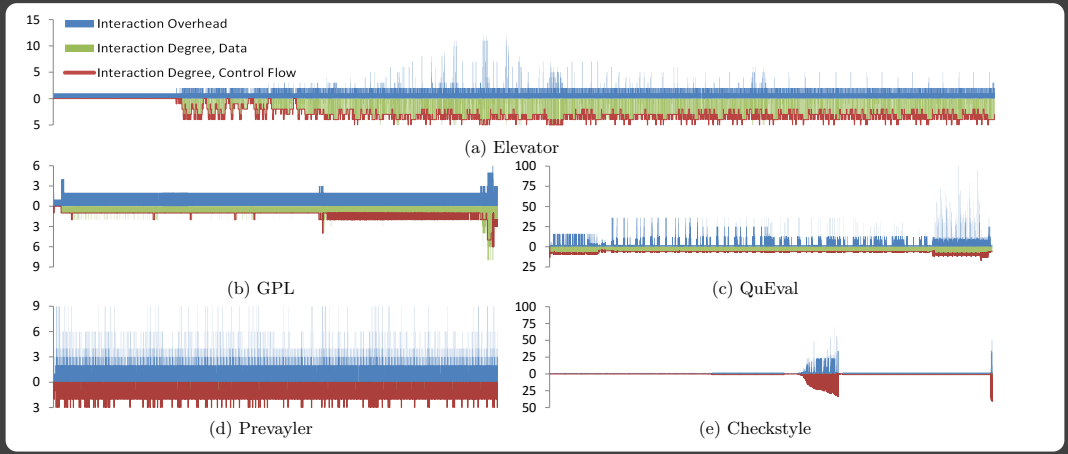
```
1  boolean STATISTICS, SMILEY, WEATHER,
2          FAHRENHEIT, SECURE_LOGIN;
3  void createHtml() {
4      String c = wpGetContent();
5      if (SMILEY)
6          c = c.replace(":]", getSmiley(":]"));
7      if (WEATHER) {
8          String weather = getWeather();
9          c = c.replace("[:weather:]", weather);
10     }
11     String head = initHeader();
12     print("<html><head>" + head + "</head><body>")
13     if (STATISTICS) {
14         long time = getCurrentTime();
15         printStatistics(time);
16     }
17     print("<div>" + c + "</div>");
18     String foot = wpGenFooter();
19     print("<hr/>" + foot + "</body></html>");
20 }
21 String getWeather() {
22     float temparature = getCelsius();
23     if (FAHRENHEIT)
24         return (temparature * 1.8 + 32) + "°F";
25     return temparature + "°C";
26 }
```



How do Features Interact?

- given a program with runtime variability
- given one test case (i.e., concrete inputs)
- how much does the execution depend on the configuration?
- how many values for each variable? (green)
- how many different control flows? (red)
- blue color not relevant here (minimal overhead during simultaneous execution)

Execution Traces in Configurable Systems [Meinicke et al. 2016]



insights: not all features interact. some statements may lead to higher interactions than others.

What is a Feature Interaction? – Summary

Lessons Learned

- feature interaction and feature-interaction problem
- wanted/unwanted, static/dynamic, one-wise/pair-wise/higher-order
- examples: customization of clothes, Android apps, toaster, preprocessor code, runtime variability, Variability Bug Database

Practice

Do you know further examples of feature interactions?

Further Reading

- Apel et al. 2013, Chapter 9, pp. 213–217

9. Feature Interactions

9a. What is a Feature Interaction?

9b. How to Handle Feature Interactions?

- Motivation and Goals
- Strategy S1: Adapt Feature Model
- Strategy S2: Orthogonal Implementation
- Strategy S3: Duplicate Implementations
- Strategy S4: Move Source Code
- Strategy S5: Conditional Compilation
- Strategy S6: Derivative Modules
- Overview and Discussion
- Summary

9c. How to Avoid Feature Interactions?

Handling/Implementing Feature Interactions

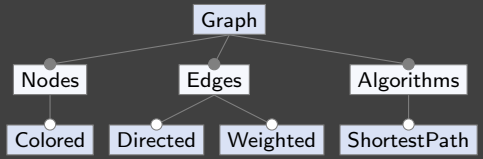
Assumptions

- Interacting features have been already identified
- Interaction is a pairwise interaction (i.e., two features)

Problem Description

- Find a strategy to handle the interaction
- Strategy does not introduce the optional-feature problem

Running Example: A Feature Interaction in our Graph Library



Optional-Feature Problem

- Domain view: *Weighted* and *ShortestPath* can be deliberately selected independent of each other.
- Implementation view: *ShortestPath* requires *Weighted* due to an implementation dependency.

Feature Module BasicGraph

```
class Edge {
  Node a, b; ...
}
```

Feature Module Weighted

```
refines class Edge {
  double weight; ...
}
```

Feature Module ShortestPath

```
refines class Graph {
  List shortestPath(Node a, Node b){
    Edge e1, e2;
    ...
    if (e1.weight > e2.weight)
    ...
  }
}
```

Handling Feature Interactions: General Goals

Question

What makes a good strategy to implement coordination code for feature interactions (while solving/avoiding the optional-feature problem)?

1. Variability

For every valid configuration (according to feature model), we can generate a product that implements this configuration.

2. Implementation Effort

Should not require overwhelming implementation effort (would not be attractive in practice).

3. Binary Size and Performance

Should not increase binary size or decrease performance of products compared to an individual implementation of each product.

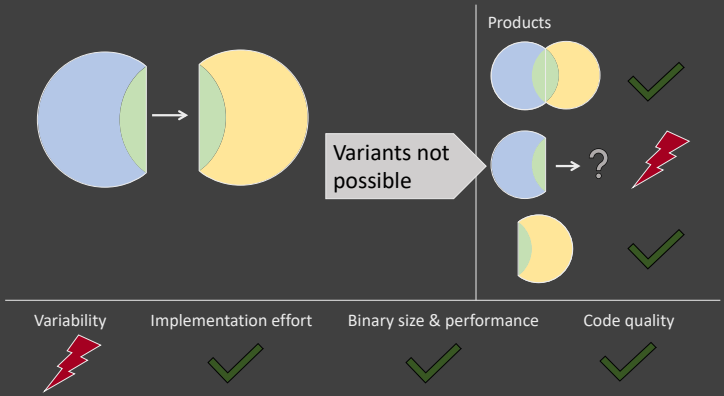
4. Code Quality

Should not reduce code quality, which would make the product line harder to maintain.

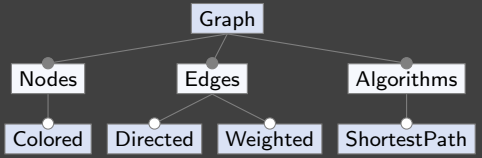
Strategy S1: Adapt Feature Model – (a) Add Domain Dependency

Strategy S1a

Declare implementation dependency as domain dependency in the feature model.



Strategy S1: Adapt Feature Model – (a) Add Domain Dependency



ShortestPath → *Weighted*

Same implementation as before, but we make implementation dependency explicit: *ShortestPath* requires *Weighted*.

Feature Module BasicGraph

```
class Edge {
  Node a, b; ...
}
```

Feature Module Weighted

```
refines class Edge {
  double weight; ...
}
```

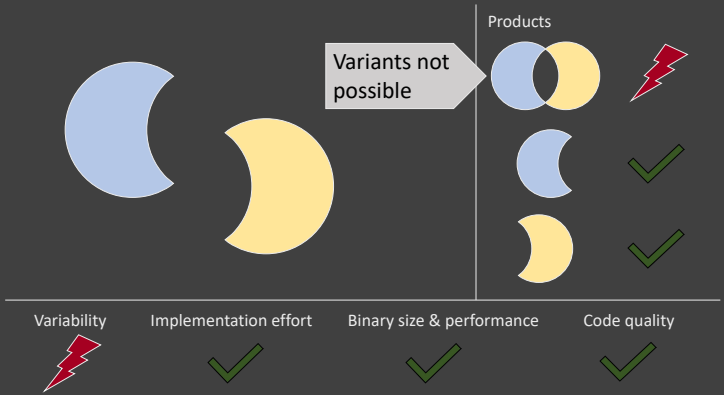
Feature Module ShortestPath

```
refines class Graph {
  List<Node> shortestPath(Node a, Node b){
    Edge e1, e2;
    ...
    if (e1.weight > e2.weight)
    ...
  }
}
```

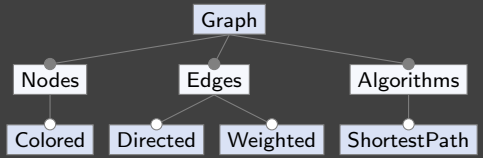
Strategy S1: Adapt Feature Model – (b) Exclude Feature Combination

Strategy S1b

Declare problematic feature combinations as mutually exclusive in the feature model.



Strategy S1: Adapt Feature Model – (b) Exclude Feature Combination



$\neg(\mathit{ShortestPath} \wedge \mathit{Weighted})$

We may safely assume any uniform weight because *ShortestPath* and *Weighted* are mutually exclusive.

Feature Module BasicGraph

```
class Edge {
  Node a, b; ...
}
```

Feature Module Weighted

```
refines class Edge {
  double weight; ...
}
```

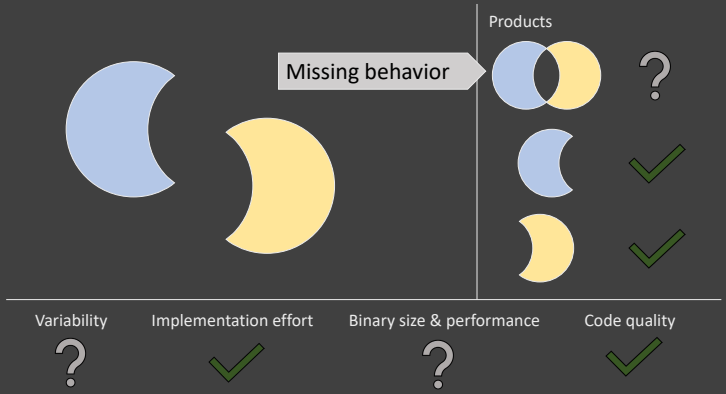
Feature Module ShortestPath

```
refines class Graph {
  List shortestPath(Node a, Node b){
    double w1 = 1.0;
    double w2 = 1.0;
    ...
    if (w1 > w2)
    ...
  }
}
```

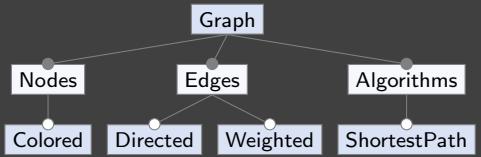
Strategy S2: Orthogonal Implementation

Strategy S2

Orthogonal implementation with no dedicated coordination of interaction.



Strategy S2: Orthogonal Implementation



$\neg(\mathit{ShortestPath} \wedge \mathit{Weighted})$

Calculation of shortest path ignores weights but merely counts the number of edges on a path.

Feature Module BasicGraph

```
class Edge {
  Node a, b; ...
}
```

Feature Module Weighted

```
refines class Edge {
  double weight; ...
}
```

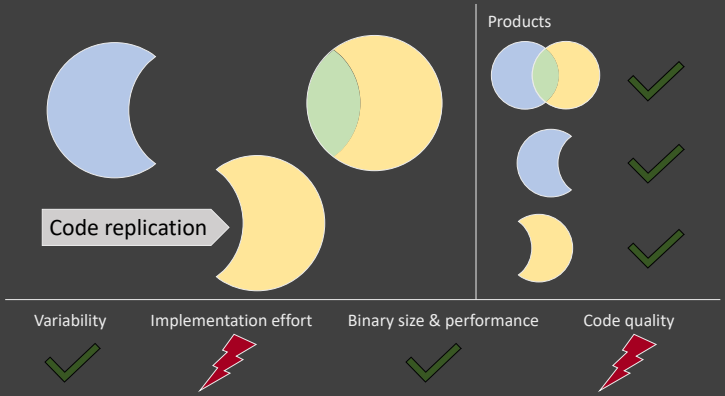
Feature Module ShortestPath

```
refines class Graph {
  List shortestPath(Node a, Node b){
    // ignore weights
    ...
  }
}
```

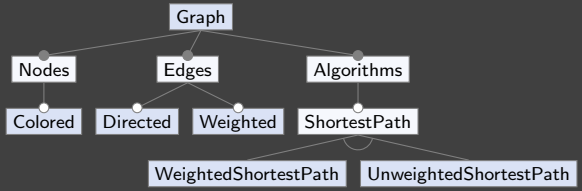
Strategy S3: Duplicate Implementations

Strategy S3

Multiple implementations of a feature, with and without coordination code.



Strategy S3: Duplicate Implementations



$WeightedShortestPath \leftrightarrow ShortestPath \wedge Weighted$

$UnweightedShortestPath \leftrightarrow ShortestPath \wedge \neg Weighted$

One of two different implementations chosen based on selection of interacting features.

Feature Module UnweightedShortestPath

```
refines class Graph {
  List shortestPath(Node a, Node b){
    ...
    ...
    // ignore weights
    ...
  }
}
```

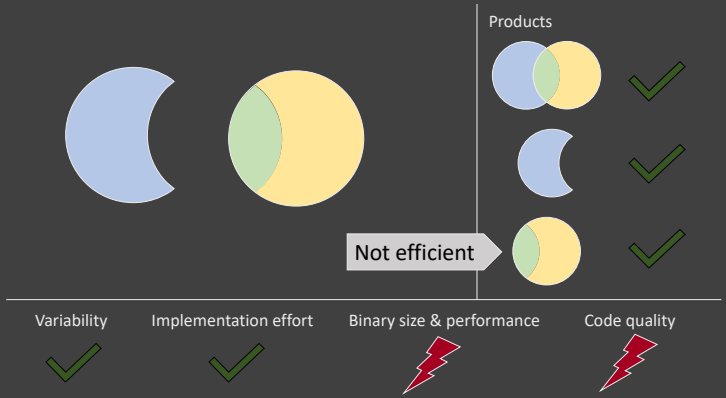
Feature Module WeightedShortestPath

```
refines class Graph {
  List shortestPath(Node a, Node b){
    Edge e1, e2;
    ...
    if (e1.weight > e2.weight)
    ...
  }
}
```

Strategy S4: Move Source Code

Strategy S4

Move all the coordination code to one of the features (or to a third one all interacting features depend on).



Strategy S4: Move Source Code

Feature Module BasicGraph

```
class Edge {
  Node a, b;
  double weight = 1.0;
  ...
}
```

Feature Module Weighted

```
refines class Edge {
  ...
}
```

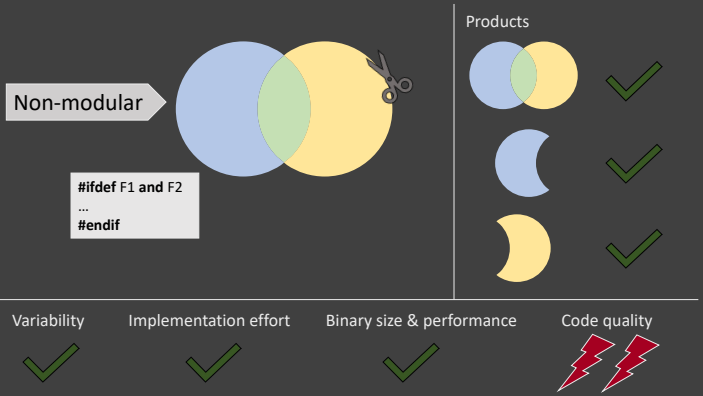
Feature Module ShortestPath

```
refines class Graph {
  List shortestPath(Node a, Node b){
    Edge e1, e2;
    ...
    if (e1.weight > e2.weight)
    ...
  }
}
```

Strategy S5: Conditional Compilation

Strategy S5

Coordination code is only executed if both features are selected.



Strategy S5: Conditional Compilation

Feature Module BasicGraph

```
class Edge {
   Node a, b; ...
}
```

Feature Module Weighted

```
refines class Edge {
   double weight; ...
}
```

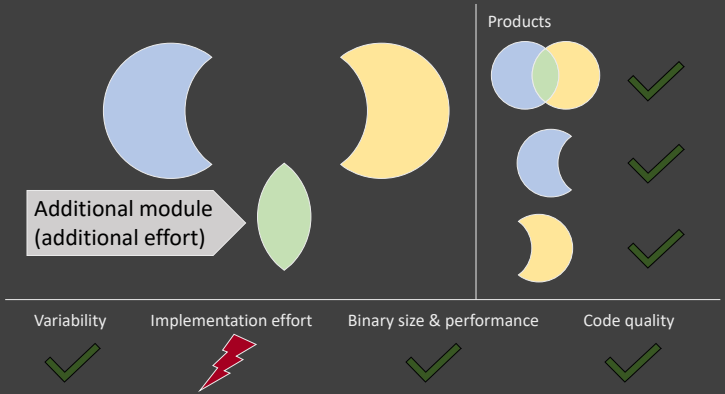
Feature Module ShortestPath

```
refines class Graph {
   List shortestPath(Node a, Node b){
      Edge e1, e2;
      ...
#ifdef WEIGHTED
      if (e1.weight > e2.weight) ...
#endif
      ...
   }
}
```

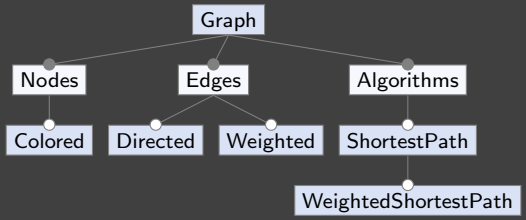
Strategy S6: Derivative Modules

Strategy S6

Create a dedicated module for code that coordinates features.



Strategy S6: Derivative Modules



$WeightedShortestPath \leftrightarrow ShortestPath \wedge Weighted$

Derivative module is selected only if both interacting features are selected.

Feature Module ShortestPath

```
refines class Graph {
  List shortestPath(Node a, Node b){
    Edge e1, e2;
    ...
    if (isLonger(e1,e2))
    ...
  }
  boolean isLonger(Edge e1, Edge e2){
    return false;
  }
}
```

Feature Module WeightedShortestPath

```
refines class Graph {
  boolean isLonger(Edge e1, Edge e2){
    return e1.weight > e2.weight;
  }
}
```

Overview and Discussion

Solution	Variability	Effort	Size & performance	Quality
S1: Adapt Feature Model		✓	✓	✓
S2: Orthogonal implementation	?	✓	?	✓
S3: Duplicate implementations	✓		✓	
S4: Move source code	✓	✓		
S5: Conditional compilation	✓	✓	✓	
S6: Derivative modules	✓		✓	✓

How to Handle Feature Interactions? – Summary

Lessons Learned

- Adaptation of feature model to avoid (undesired) feature interactions.
- Strategies to implement coordination code for known feature interactions.
- Discussion of the strenghts and weaknesses of each of the strategies.

Further Reading

- Kästner et al.: On the impact of the optional feature problem: Analysis and case studies. SPLC 2009.
- Apel et al. 2013, Chapter 9

Practice

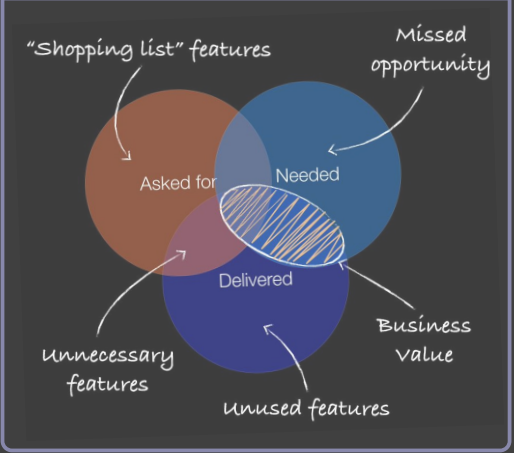
Looking back at our graph library and the feature interaction between *ShortestPath* and *Weighted*. Which strategy would you choose to handle this interaction? Why?

Can you think of other feature interactions for the graph library (you may also add additional features)? Again, how would you handle them?

9. Feature Interactions

The Choice of Features

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."

The Choice of Features

Henry Ford, 1909

"Any customer can have a car painted any color that he wants so long as it is black."

Why only black? [Apel et al. 2013]

- black color dried faster
- faster production
- more products and cheaper production

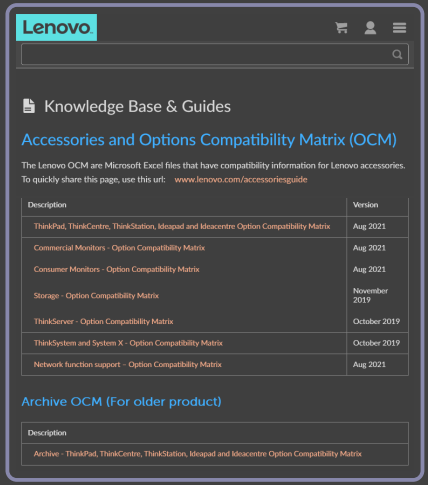
Documentation of Interactions – Incompatibilities of Lenovo Hardware

Documentation of Remaining Interactions

- not all interactions can be prevented/fixed
- how to apply strategy S1 (see Part II) if there is no feature model?
- what to document?

Lenovo's Option Compatibility Matrices

- 7 Excel files for current products + archive for old products
- Excel file for computers contains 32 tables (series)
- table for ThinkPad X has 28 columns (models) and > 500 rows (accessories)
- 14k cells contain > 400 different footnotes
- a footnote explains one incompatibility


Lenovo

Knowledge Base & Guides

Accessories and Options Compatibility Matrix (OCM)

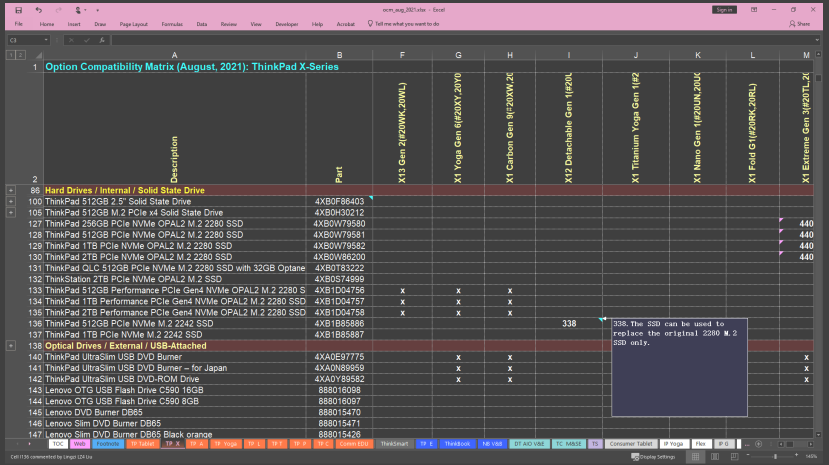
The Lenovo OCM are Microsoft Excel files that have compatibility information for Lenovo accessories.
To quickly share this page, use this url: www.lenovo.com/accessoriesguide

Description	Version
ThinkPad, ThinkCentre, ThinkStation, Ideapad and Ideacentre Option Compatibility Matrix	Aug 2021
Commercial Monitors - Option Compatibility Matrix	Aug 2021
Consumer Monitors - Option Compatibility Matrix	Aug 2021
Storage - Option Compatibility Matrix	November 2019
ThinkServer - Option Compatibility Matrix	October 2019
ThinkSystem and System X - Option Compatibility Matrix	October 2019
Network function support - Option Compatibility Matrix	Aug 2021

Archive OCM (For older product)

Description
Archive - ThinkPad, ThinkCentre, ThinkStation, Ideapad and Ideacentre Option Compatibility Matrix

Documentation of Interactions – Incompatibilities of Lenovo Hardware

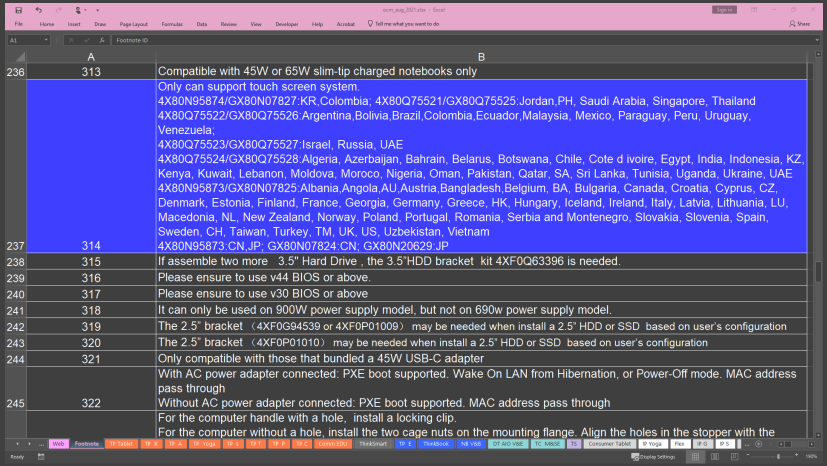


	A	B	F	G	H	I	J	K	L	M
1	Option Compatibility Matrix (August, 2021): ThinkPad X-Series									
2	Description	Part	X13 Gen 2i(#20WK,20WL)	X1 Yoga Gen 6i(#20XY,20Y0	X1 Carbon Gen 9i(#20XW,20	X12 Detachable Gen 1(#20L	X1 Titanium Yoga Gen 1(#2	X1 Nano Gen 1(#20UN,20UC	X1 Fold G1(#20RK,20RL)	X1 Extreme Gen 3i(#20TL,2C
86	Hard Drives / Internal / Solid State Drive									
100	ThinkPad 512GB 2.5" Solid State Drive	4XB0F86403								
105	ThinkPad 512GB M.2 PCIe x4 Solid State Drive	4XB0H30212								
127	ThinkPad 256GB PCIe NVMe OPAL2 M.2 2280 SSD	4XB0W79580								440
128	ThinkPad 512GB PCIe NVMe OPAL2 M.2 2280 SSD	4XB0W79581								440
129	ThinkPad 1TB PCIe NVMe OPAL2 M.2 2280 SSD	4XB0W79582								440
130	ThinkPad 2TB PCIe NVMe OPAL2 M.2 2280 SSD	4XB0W86200								440
131	ThinkPad QLC 512GB PCIe NVMe M.2 2280 SSD with 32GB Optane	4XB0T83222								
132	ThinkStation 2TB PCIe NVMe OPAL2 M.2 SSD	4XB0S74999								
133	ThinkPad 512GB Performance PCIe Gen4 NVMe OPAL2 M.2 2280 S	4XB1D04756	x	x	x					
134	ThinkPad 1TB Performance PCIe Gen4 NVMe OPAL2 M.2 2280 SSD	4XB1D04757	x	x	x					
135	ThinkPad 2TB Performance PCIe Gen4 NVMe OPAL2 M.2 2280 SSD	4XB1D04758	x	x	x					
136	ThinkPad 512GB PCIe NVMe M.2 2242 SSD	4XB1B85886				338				
137	ThinkPad 1TB PCIe NVMe M.2 2242 SSD	4XB1B85887								
138	Optical Drives / External / USB-Attached									
140	ThinkPad UltraSlim USB DVD Burner	4XA0E97775		x	x					x
141	ThinkPad UltraSlim USB DVD Burner – for Japan	4XA0N89959		x	x					x
142	ThinkPad UltraSlim USB DVD-ROM Drive	4XA0Y89582		x	x					x
143	Lenovo OTG USB Flash Drive C590 16GB	888016098								
144	Lenovo OTG USB Flash Drive C590 8GB	888016097								
145	Lenovo DVD Burner DB65	888015470								
146	Lenovo Slim DVD Burner DB65	888015471								
147	Lenovo Slim DVD Burner DB65 Black orange	888015426								

338: The SSD can be used to replace the original 2280 M.2 SSD only.

- columns contain notebooks
- rows contain accessories
- X indicates compatibility
- numbers indicate a known incompatibility

Documentation of Interactions – Incompatibilities of Lenovo Hardware



	A	B
236	313	Compatible with 45W or 65W slim-tip charged notebooks only
237	314	Only can support touch screen system. 4X80N95874/GX80N07827:KR,Colombia; 4X80Q75521/GX80Q75525:Jordan,PH, Saudi Arabia, Singapore, Thailand 4X80Q75522/GX80Q75526:Argentina,Bolivia,Brazil,Colombia,Ecuador,Malaysia, Mexico, Paraguay, Peru, Uruguay, Venezuela; 4X80Q75523/GX80Q75527:Israel, Russia, UAE 4X80Q75524/GX80Q75528:Algeria, Azerbaijan, Bahrain, Belarus, Botswana, Chile, Cote d ivoire, Egypt, India, Indonesia, KZ, Kenya, Kuwait, Lebanon, Moldova, Moroco, Nigeria, Oman, Pakistan, Qatar, SA, Sri Lanka, Tunisia, Uganda, Ukraine, UAE 4X80N95873/GX80N07825:Albania,Angola,AU,Austria,Bangladesh,Belgium, BA, Bulgaria, Canada, Croatia, Cyprus, CZ, Denmark, Estonia, Finland, France, Georgia, Germany, Greece, HK, Hungary, Iceland, Ireland, Italy, Latvia, Lithuania, LU, Macedonia, NL, New Zealand, Norway, Poland, Portugal, Romania, Serbia and Montenegro, Slovakia, Slovenia, Spain, Sweden, CH, Taiwan, Turkey, TM, UK, US, Uzbekistan, Vietnam 4X80N95873:CN,JP; GX80N07824:CN; GX80N20629:JP
238	315	If assemble two more 3.5" Hard Drive , the 3.5"HDD bracket kit 4XF0Q63396 is needed.
239	316	Please ensure to use v44 BIOS or above.
240	317	Please ensure to use v30 BIOS or above
241	318	It can only be used on 900W power supply model, but not on 690w power supply model.
242	319	The 2.5" bracket （4XF0G94539 or 4XF0P01009） may be needed when install a 2.5" HDD or SSD based on user's configuration
243	320	The 2.5" bracket （4XF0P01010） may be needed when install a 2.5" HDD or SSD based on user's configuration
244	321	Only compatible with those that bundled a 45W USB-C adapter
245	322	With AC power adapter connected: PXE boot supported. Wake On LAN from Hibernation, or Power-Off mode. MAC address pass through Without AC power adapter connected: PXE boot supported. MAC address pass through
		For the computer handle with a hole, install a locking clip. For the computer without a hole, install the two cage nuts on the mounting flange. Align the holes in the stopper with the

- 314: pen requires touch screen (cf. S1)
- 315/319/320: extra module needed (cf. S6)
- 316/317: fixed in newer BIOS versions
- 318/321: two modules with a different power supply (cf. S3)

How to Avoid Feature Interactions? – Summary

Lessons Learned

- reduction of variability
- which features are actually needed?
- documentation of interactions that cannot be avoided

Further Reading

Apel et al. 2013

Practice

Who checks the compatibility of Lenovo products?

FAQ – 9. Feature Interactions

Lecture 9a

- What are (inadvertent) feature interactions? Give examples!
- Are feature interactions limited to product lines?
- What is the feature-interaction problem? Why is it critical for product lines?
- What is the difference of static/dynamic, one-wise/pair-wise/higher-order interactions?
- What are typical patterns of interactions?
- How can features interact on control flow and data?

Lecture 9b

- How to resolve feature interactions?
- What is the optional feature problem?
- What are typical goals when resolving feature interactions?
- Name/explain strategies to resolve feature interactions!
- What are (dis)advantages of those strategies?

Lecture 9c

- How to cope with feature interactions?
- How to reduce variability?
- What are unused, unnecessary, and shopping-list-features?
- How to document feature interactions?