WIFI - What is a Feature Interaction?

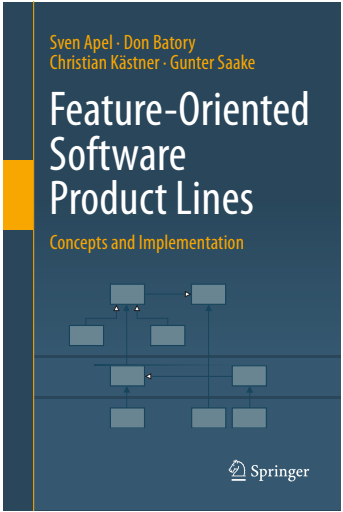
FOSD'26 | Sabrina Böhm, Tim Schmidt, Sebastian Krieter, Thomas Thüm, Malte Lochau | March 23–27, 2026

universität ulm

Technische Universität Braunschweig

Universität Siegen

Definition of Feature Interactions [Apel et al. 2013]



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

WIFI - What is a Feature Interaction?

FOSD'26 | Sabrina Böhm, Tim Schmidt, Sebastian Krieter, Thomas Thüm, Malte Lochau | March 23–27, 2026

universität ulm

Technische Universität Braunschweig

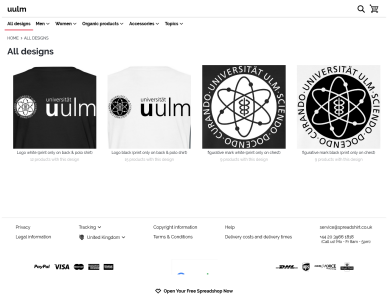
Universität Siegen

Example Interactions

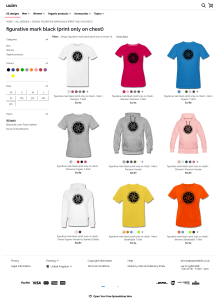
Customization of Clothes

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

uulm Shop



[myspreadshop.de]



Example Interactions

The Problem: Unwanted Interaction of Colors

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

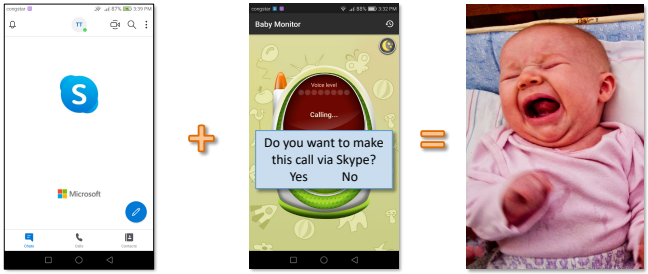
Example Interactions

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)

Example Interactions



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?

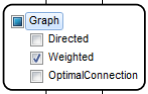
Example Interactions

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));
} }
```



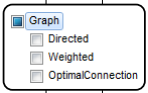

```
class Edge {
  Node first, second;

  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));
} }
```



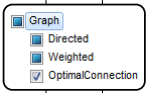

```
class Edge {
  Node first, second;


  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));
} }
```

Graph
- Directed
- Weighted
- OptimalConnection

```
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

Interaction Patterns [Abal et al. 2018]

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 Patterns [Abal et al. 2018]

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

Patterns Beyond Conjunction of Literals

- Apache: `CROSS_COMPILE && (WIN32 || OS2)`
- Busybox: `LFS && (FEATURE_HTTPD_BASIC_AUTH || FEATURE_HTTPD_CGI)`
- Linux: `NUMA && (SMP || DEBUG_SPINLOCK || DEBUG_LOCK_ALLOC)`
- Linux: `(SND_FSI_AK4642 || SND_FSI_DA7210) && !I2C`
- JHipster: `(Microservice ||Gateway) && UAA && Docker`
- JHipster: `(Microservice ||Gateway) && UAA && !Docker`

WIFI - What is a Feature Interaction?

FOSD'26 | Sabrina Böhm, Tim Schmidt, Sebastian Krieter, Thomas Thüm, Malte Lochau | March 23–27, 2026

universität ulm

Technische Universität Braunschweig

Universität Siegen

WIFI - What is a Feature Interaction?

1. **Definition of Feature Interactions**
2. **Example Interactions**
3. **Example Interactions with Preprocessors**
4. **Interaction Patterns**
5. **Our Nomenclature**

References I


Abal, Iago et al. (Jan. 2018). "Variability Bugs in Highly Configurable Systems: A Qualitative Analysis". In: *Trans. on Software Engineering and Methodology (TOSEM)* 26.3, 10:1–10:34. ISSN: 1049-331X. DOI: 10.1145/3149119.


Apel, Sven et al. (2013). *Feature-Oriented Software Product Lines*. Berlin, Heidelberg: Springer. ISBN: 978-3-642-37520-0. DOI: 10.1007/978-3-642-37521-7.


Rhein, Alexander von et al. (2015). "Presence-Condition Simplification in Highly Configurable Systems". In: *Proc. Int'l Conf. on Software Engineering (ICSE)*. Florence, Italy: IEEE, pp. 178–188. ISBN: 978-1-4799-1934-5.

Our Nomenclature

- The *feature interaction cardinality* is the number of features interacting.
- A *feature interaction formula* is a logical formula with features as variables that specifies in which valid configurations a feature interaction is present. Ideally, the feature interaction formula is minimized [Rhein et al. 2015] with respect to the feature model.
- One strategy to *resolve a feature interaction* is to restrict the set of valid configurations such that the interacting feature combinations are excluded from the feature model.
- A *feature dependency* refers to the fact that one or more features depend on each other and is typically documented in a feature model. In problem space, the feature model may contain an implication from one feature to another. In solution space, one feature may call a method defined in another feature.