The New Feature Interaction Challenge

Sven Apel
University of Passau, Germany
The Promise of Components

„[…] software components are executable units of independent production, acquisition, and deployment that can be composed into a functioning system.“

— Clemens Szyperski
The Promise of Components
The Promise of Components
The Promise of Components
The Promise of Components
Crosscutting Features
Feature Modularity
So, what is the problem?
How about behavior?
Feature Interactions

“A feature is a prominent or distinctive user-visible aspect, quality, or characteristic of a software system or systems “ — Kyo-Chul Kang

“A feature interaction is some way in which a feature or features modify or influence another feature in defining overall system behavior.“ — Pamela Zave
Desired Feature Interactions

Database System

Transactions

Statistics
Feature Modularity

Transactions

Glue code
Lifter
Derivative
Connector

Coordination

Statistics
A matter of Scale

Pair-wise Interactions: $O(n^2)$
Including higher orders: $O(2^n)$

A matter of Scale
<table>
<thead>
<tr>
<th>ORACLE</th>
<th>BERKELEY DB</th>
<th>Insert</th>
<th>Memory mgmt.</th>
<th>Statistics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transactions</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fsync</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Logging</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(Kästner et al., SPLC’09)
BERKELEY DB

Transactions

Insert
Mem mgmt.
Statistics

(Fsyncl, Logging)

Insert
Memory mgmt.
Statistics

(Kästner et al., SPLC’09)
→ Optional-Feature Problem
(Kästner et al., SPLC'09)
Undesired Feature Interactions

(Hall, ASE J., 2005)
Date: Thu, 21 Aug 2014 23:34:12 -0400 (EDT)
From: cve-assgin@...re.org
To: henri@...v.fi
Cc: cve-assgin@...re.org, oss-security@...ts.openwall.com
Subject: Re: Enigmail warning

-----BEGIN PGP SIGNED MESSAGE-----
Hash: SHA1

> http://sourceforge.net/p/enigmail/forum/support/thread/3e7268a4/

This seems to discuss at least two non-identical issues.

http://sourceforge.net/p/enigmail/forum/support/thread/3e7268a4/#b315
and http://sourceforge.net/p/enigmail/bugs/294/ are about "an email
with only Bcc recipients is sent in plain text." This is assigned
CVE-2014-5369.

http://sourceforge.net/p/enigmail/forum/support/thread/3e7268a4/#10f1
and
http://sourceforge.net/p/enigmail/forum/support/thread/3e7268a4/#0a5a
are about one or more issues in which there is unexpected cleartext
email transmission unrelated to use of Bcc. This perhaps requires
a non-default configuration. It is conceivable -- although perhaps
unlikely -- that the problem is a UI bug (e.g., an encryption choice
is presented even when the product is configured to never use
encryption). In any case, none of this has a CVE assignment yet. There
Crosscutting  ⇔  Feature Interaction
Feature-Interaction Zoo (i)
Feature-Interaction Zoo (i)

- $\mathbb{E} \models \phi$
- $\mathbb{E} \models \phi$
- $\mathbb{E} \models \phi$
- $\prod (\text{ }} = 50s$
- $\prod (\text{ }} = 30s$
- $\prod (\text{ } \bullet \text{ } \text{ }) = 60s$

not 80s!
Feature-Interaction Zoo (ii)

...  
#if ENABLE_FEATURE_HUMAN_READABLE && ENABLE_FEATURE_DF_FANCY  
opt_complementary = "k-mB:m-Bk:B-km"; // coordination code  
#endif  
...
Feature-Interaction Zoo (ii)

```c
#if ENABLE_FEATURE_HUMAN_READABLE && ENABLE_FEATURE_DF_FANCY
opt_complementary = ”k-mB:m-Bk:B-km”; // coordination code
#endif
```

EmailMsg receiveMsg(MsgId id) {
    ...
    EmailMsg msg = server.getMsg(id);
    return msg;
}

EmailMsg receiveMsg(MsgId id) {
    EmailMsg msg = super.receiveMsg(id);
    server.forward(msg, addressList);
    return msg;
}
There is a whole zoo of different kinds of feature interactions!
How to efficiently detect and reason about feature interactions?
33 features

a unique configuration/variant for every person on this planet
320 features

more configurations/variants than estimated

atoms in the universe
Brute-Force Analysis
(a.k.a. Variant-Based or Product-Based Analysis)
Example:

**Big bang**

- Time line
  - Birth of Earth: $9 \times 10^9$ years
  - Now: $1.37 \times 10^{10}$ years
  - Measurement finished: $2.9 \times 10^{21}$ years

**TU/e**
Sampling:
(random, t-wise, code coverage, …)

Create Feature Combination

Analyze few, make statements about many

Standard Analysis
Variability-Aware Analysis
(a.k.a. Family-Based Analysis)
A Classification and Survey of Analysis Strategies for Software Product Lines

THOMAS THUM, University of Magdeburg, Germany
SVEN APEL, University of Passau, Germany
CHRISTIAN KASTNER, Carnegie Mellon University, USA
INA SCHAEFER, University of Braunschweig, Germany
GUNTER SAAKE, University of Magdeburg, Germany

Software-product-line engineering has gained considerable momentum in the recent years, both in industry and in academia. A software product line is a family of software products that share a common set of features. Software product lines challenge traditional analysis techniques, such as type checking, model checking, and theorem proving, in their quest of ensuring correctness and reliability of software. Simply creating and analyzing all products of a product line is usually not feasible, due to the potentially exponential number of valid feature combinations. Recently, researchers began to develop analysis techniques that take the distinguishing properties of software product lines into account, for example, by checking feature-related code in isolation or by exploiting variability information during analysis. The emerging field of product-line analyses is both broad and diverse, so it is difficult for researchers and practitioners to understand their similarities and differences. We propose a classification of product-line analyses to enable systematic research and application. Based on our insights with classifying and comparing a corpus of 123 research articles, we develop a research agenda to guide future research on product-line analyses.

Categories and Subject Descriptors: A.1 [General]: Introductory and Survey; D.2.2 [Software Engineering]: Design Tools and Techniques—Modules and interfaces; D.2.4 [Software Engineering]: Software/Program Verification—Correctness proofs, formal methods, model checking; D.2.9 [Software Engineering]: Management—Software configuration management; D.2.13 [Software Engineering]: Reusable Software—Domain engineering; D.3.4 [Software Engineering]: Processors—Code generation, compilers, incremental compilers, parsing; E.3.1 [Logics and Meanings of Programs]: Specifying and Verifying and Reasoning about Programs—Mechanical verification, specification techniques

General Terms: Design; Experimentation; Reliability; Theory; Verification

Additional Key Words and Phrases: Product-line analysis, software product line, program family, software analysis, type checking, static analysis, model checking, theorem proving

1. INTRODUCTION

Software-product-line engineering aims at providing techniques for efficient development of software product lines [Czarnecki and Eisenecker 2000; Clements and Northrop 2001; Pohl et al. 2005; Apel et al. 2013]. A software product line (or program family [Parnas 1976]) consists of a set of similar software products that rely on a common code base. The software products of a product line are distinguished in terms of the features they provide. A feature is a prominent or distinctive user-visible behavior.

Apel’s work is supported by the German Research Foundation (DFG – AP 2062, AP 2064, AP 2065, AP 2066, and AP 2067). Kastner’s work is supported by the German Research Foundation (DFG – SCHA 1635/2). Saake’s work is supported by the German Research Foundation (DFG – SA 465/34).

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other work requires permission of the owner.
Which kinds of feature interactions occur in practice?
What are their properties?
What can we learn from them?
Case Study #1
int foo(int a, int b) {
    int c = a;
    if (c) {
        #ifdef A
            c += a;
        #ifdef B
            c += b;
        #endif
        #endif
    }
    return c;
}
int foo(int a, int b) {
    int c = a;
    if (c) {
#ifdef A
        c += a;
#ifdef B
        c += b;
#endif
#endif
    }
    return c;
}
Interactions in Control Flow

```
2  int c = a;
3  if (c)
4      A
5          c += a;
6      A \land B
7          c += b;
8      A \land B
9  return c;
```

```
Linux
```

```
Busybox
```

```
Linux
```

```
OpenSSL
```

```
SQLite
```

```
uClibc
```
Interaction Bugs

Double freeing
Freeing static memory
Uninitialized variables
Terminating cases
Missing returns
Dangling switch code
Interaction Bugs
(Medeiros et al., ICSE’16)
Feature interactions occur in practice, and not only between pairs of features!
Case Study #2
Performance Prediction
(Siegmund et al., ICSE'12)

Performance Influence Model

What is the performance of configuration X?

What is the fastest configuration in a given setting?

Partial feature selection

Workload

Objective function

(PageSize_1k, 15s),
(PageSize_2k, 0s),
(PageSize_4k, -10s), (CacheSize_8k, -5s),
(Encryption, 20s), (HashIndex, -5s),
(Encryption#PageSize_4k, 15s),
...
Determining a Feature’s Influence

(Siegmund et al., ICSE’12)

\[ \Pi (\text{Core}) = 100s \]
\[ \Pi (\text{Compression}, \text{Encryption}) = 120s \]
\[ \Pi (\text{Core}, \text{Encryption}) = 130s \]
\[ \Pi (\text{Compression}, \text{Transactions}) = 110s \]
\[ \Delta (\text{Compression}) = 20s \]
\[ \Delta (\text{Encryption}) = 30s \]
\[ \Delta (\text{Transactions}) = 10s \]

\[ \Phi (\text{Core}, \text{Compression}, \text{Encryption}, \text{Transactions}) = \Pi (\text{Core}) + \Delta (\text{Compression}) + \Delta (\text{Encryption}) + \Delta (\text{Transactions}) \]
\[ = 160s \]
Dude... wait....

What???
Feature Interactions!

(Siegmund et al., ICSE’12)

\[ \Phi \]

Predicted performance

\[ \Phi (\text{\ding{53}}, \text{\ding{54}}, \text{\ding{55}}) = \prod (\text{\ding{53}}) + \Delta (\text{\ding{54}}) + \Delta (\text{\ding{55}}) \]

= 100s + 20s + 30s

= \textbf{150s}

Measured performance

\[ \prod (\text{\ding{53}}, \text{\ding{54}}, \text{\ding{55}}) = 140s \]

Feature interaction: \text{\ding{53}} \# \text{\ding{55}} due to encrypting compressed data
Which Features interact at all?

(Siegmund et al., ICSE’12)
Which Features are central?
(Siegmund et al., ICSE’12)
Performance-Influence Models

(Siegmund et al., ESEC/FSE’15)

Performance-Influence Model:

$$\Pi(C) = 100 \times C + 20 \times \text{Compression} + 30 \times \text{Encryption} + 10 \times \text{Transactions} - 10 \times \text{Interaction} \ldots$$

Feature Interaction!
### System Domain, Language, LOC, Features, Variants

<table>
<thead>
<tr>
<th>System</th>
<th>Domain</th>
<th>Lang.</th>
<th>LOC</th>
<th>Features</th>
<th>Variants</th>
</tr>
</thead>
<tbody>
<tr>
<td>Berkeley DB</td>
<td>Database</td>
<td>C</td>
<td>219811</td>
<td>18</td>
<td>2560</td>
</tr>
<tr>
<td>Berkeley DB</td>
<td>Database</td>
<td>Java</td>
<td>42596</td>
<td>32</td>
<td>400</td>
</tr>
<tr>
<td>Apache</td>
<td>Web server</td>
<td>C</td>
<td>230277</td>
<td>9</td>
<td>192</td>
</tr>
<tr>
<td>SQLite</td>
<td>Database</td>
<td>C</td>
<td>312625</td>
<td>39</td>
<td>3932160</td>
</tr>
<tr>
<td>LLVM</td>
<td>Compiler</td>
<td>C++</td>
<td>47549</td>
<td>11</td>
<td>1024</td>
</tr>
<tr>
<td>x264</td>
<td>Video Enc.</td>
<td>C</td>
<td>45743</td>
<td>16</td>
<td>1152</td>
</tr>
</tbody>
</table>

*Two days of measurement with 93% accuracy (Siegmund et al., ICSE'12)*

---

**Time line**

- **Big bang**: $9 \times 10^9$ years
- **Birth of Earth**: $9 \times 10^9$ years
- **Now**: $1.37 \times 10^{10}$ years
- **Measurement finished**: $2.9 \times 10^{21}$ years

---

[http://fosd.net/SPLConqueror]
Performance Interactions
JUQUEEN — Jülich Blue Gene/Q
(458.752 Cores, 5.9 Petaflops)

\[
\frac{\partial u}{\partial t} - \alpha \left( \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2} \right) = 0
\]

Matthias Bolten  Harald Köstler  Christian Engwer
Configuration predicted to be optimal

\[
\frac{\partial u}{\partial t} - \alpha \left( \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2} \right) = 0
\]
Also for numeric parameters! (Siegmund et al., ESEC/FSE'15)
Feature interactions occur in practice, at the level of non-functional properties!
Two Roads Diverged In A Yellow Wood, 
And Sorry I Could Not Travel Both.
The New Feature-Interaction Challenge

(Apel et al., FOSD’13)

Properties & Distribution?

Correlation?

Prediction?

\[ \Pi (\text{\large box} \bullet \text{\large lock}) = 60s \]
Different kinds of feature interactions are potentially related—take advantage of that!
How about behavior?
How about behavior?
How about behavior?

Interactions in Control Flow

```
int foo(int a, int b) {
    int c = a;
    if (c) {
        #ifdef A
        c += a;
        #ifdef B
        c += b;
        #endif
    } #endif
    return c;
}
```

Variability-Aware Analysis with TypeChef

CFG with Variability Information
How about behavior?

Interactions in Control Flow

Feature Interactions!

(Liebig et al., ESEC/FSE'13)

(int foo(int a, int b) {
    int c = a;
    if (c) {
        #ifdef A
            c += a;
        #ifdef B
            c += b;
        #endif
        #endif
    }
    return c;
})

Variability-Aware Analysis with **TypeChef**

CFG with Variability Information

Feature interaction: 🍭 # 🍭 due to encrypting compressed data

$$\Phi = \Phi_{\text{predicted}} + \Delta_{\text{actual}} + \Delta_{\text{expected}}$$

$$= 100s + 20s + 30s = 150s$$

$$\Phi_{\text{measured}} = 140s$$
The New Feature-Interaction Challenge

(Apel et al., FOSD'13)

Properties & Distribution?

Correlation?

Prediction?

$\prod (\text{key} \cdot \text{lock}) = 60s$
Sven Apel, Prof. Dr.

Chair of Software Engineering
University of Passau
Germany
apel@uni-passau.de

A comparison of 10 sampling algorithms for configurable systems.

ESEC/FSE’15  N. Siegmund, A. Grebhahn, S. Apel, C. Kästner.
Performance-influence models for highly configurable systems.

A classification and survey of analysis strategies for software product lines.

Scalable analysis of variable software.


Predicting performance via automated feature-interaction detection.

DKE’09  M. Rosenmüller, S. Apel, T. Leich, G. Saake.
Tailor-made data management for embedded systems: A case study on Berkeley DB.

On the impact of the optional feature problem: analysis and case studies.

SPLC’07  C. Kästner, S. Apel, D. Batory.
A case study implementing features using AspectJ.