Exploring Feature Interactions in the Wild

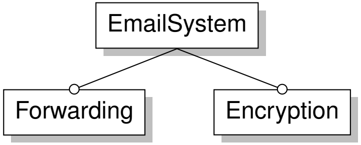
Sergiy Kolesnikov[1] Judith Roth[1] Sven Apel[1]
Norbert Siegmund[1] Christian Kästner[2] Brady Garvin[3]

[1]University of Passau, Germany [2]Carnegie Mellon University, USA

[3]University of Nebraska–Lincoln, USA

FOSD Meeting 2014, Dagstuhl

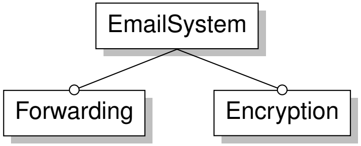
Example of a Feature Interaction



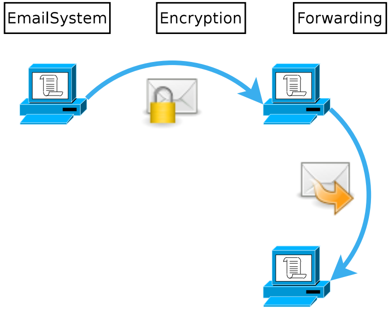
○ optional feature

R. Hall. Fundamental nonmodularity in electronic mail. Automated Soft. Eng. '05

Example of a Feature Interaction

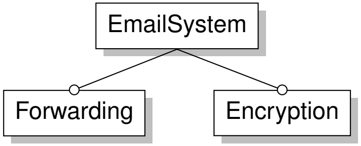


○ optional feature

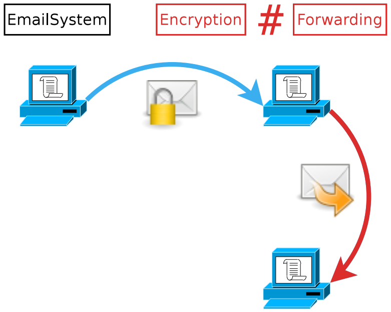


R. Hall. Fundamental nonmodularity in electronic mail. Automated Soft. Eng. '05

Example of a Feature Interaction



○ optional feature



R. Hall. Fundamental nonmodularity in electronic mail. Automated Soft. Eng. ’05

Specification Violation and Minimality

- Specification violation

$$\phi := \begin{array}{c} \textbf{AG}\ \big(\text{recv}(\text{msg } m) \wedge m.\text{isEncrypted}\big) \Rightarrow \\ \big((\text{send}(\text{msg } m) \Rightarrow m.\text{isEncrypted})\ \textbf{R}\ \text{send}(\text{msg } m)\big) \end{array}$$

Specification Violation and Minimality

- Specification violation

$$\phi := \begin{array}{c} \mathbf{AG}\ \big(\text{recv}(\text{msg } m) \wedge m.\text{isEncrypted}\big) \Rightarrow \\ \big((\text{send}(\text{msg } m) \Rightarrow m.\text{isEncrypted})\ \mathbf{R}\ \text{send}(\text{msg } m)\big) \end{array}$$

$$\textit{Encryption} \wedge \textit{Forwarding} \not\models \phi$$

Specification Violation and Minimality

- Specification violation

$$\phi := \begin{array}{c}\textbf{AG}\ \big(\text{recv}(\text{msg } m) \wedge m.\text{isEncrypted}\big) \Rightarrow \\ \big((\text{send}(\text{msg } m) \Rightarrow m.\text{isEncrypted})\ \textbf{R}\ \text{send}(\text{msg } m)\big)\end{array}$$

$$\textit{Encryption} \wedge \textit{Forwarding} \not\models \phi$$

- Minimality

$$\textit{Encryption} \models \phi$$

$$\textit{Forwarding} \models \phi$$

Specification Violation and Minimality

- Specification violation

$$\phi := \begin{array}{c}\mathbf{AG}\ \left(\text{recv}(\text{msg } m) \wedge m.\text{isEncrypted}\right) \Rightarrow \\ \left((\text{send}(\text{msg } m) \Rightarrow m.\text{isEncrypted})\ \mathbf{R}\ \text{send}(\text{msg } m)\right)\end{array}$$

$$\textit{Encryption} \wedge \textit{Forwarding} \not\models \phi$$

- Minimality

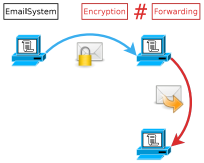
$$\textit{Encryption} \models \phi$$

$$\textit{Forwarding} \models \phi$$

$$\textit{Encryption} \# \textit{Forwarding}$$

External Interactions

- External Interactions:
  - Functional (e.g., mail is forwarded unencrypted)



  - Non-functional (e.g., unexpected performance drop/gain)

DBMS with Encryption and Compression features:

📦 = -10% (rps)
🔒 = -30% (rps)

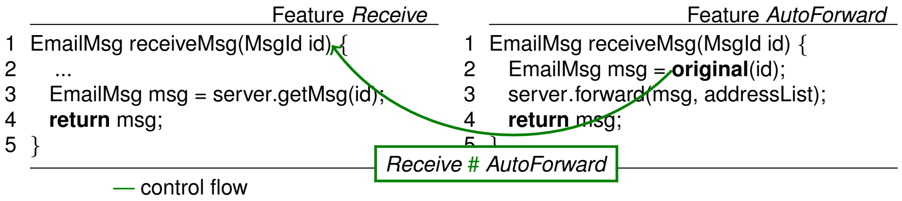
Expected: 📦 + 🔒 = -40%
Measured: 📦 + 🔒 = -20%
Interaction: 📦 # 🔒 = +20%

Internal Interactions

- Internal Interactions:
  - Structural (e.g., code nesting)

```
// from BusyBox
#if ENABLE_FEATURE_HUMAN_READABLE &&
      ENABLE_FEATURE_DF_FANCY
  opt_complementary = "k−mB:m−Bk:B−km"; // coordination code
#endif
```

  - Operational (e.g., variable control flow)



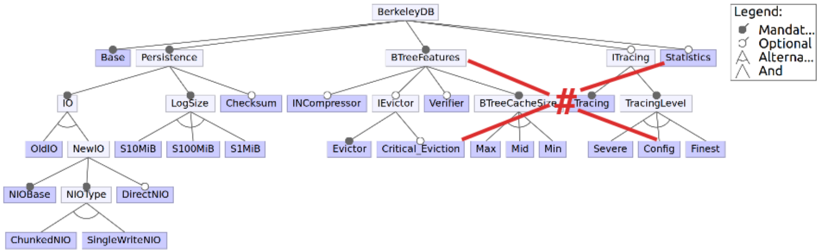
Feature *Receive*

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

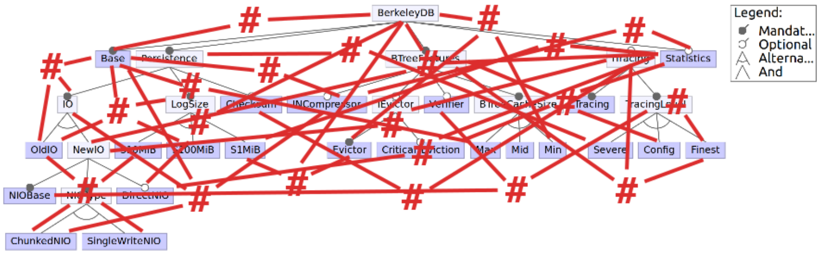
Feature *AutoForward*

```
EmailMsg receiveMsg(MsgId id) {
  EmailMsg msg = original(id);
  server.forward(msg, addressList);
  return msg;
}
```

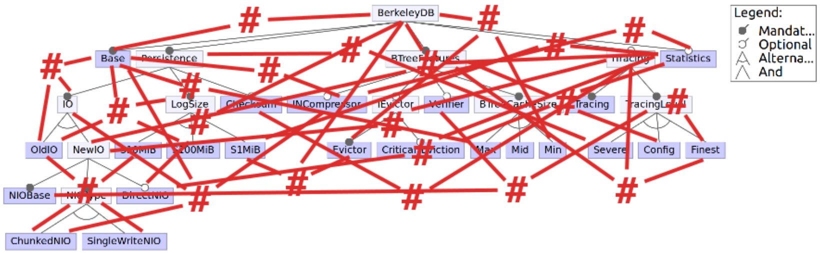
Interactions in Real-World Systems



Interactions in Real-World Systems

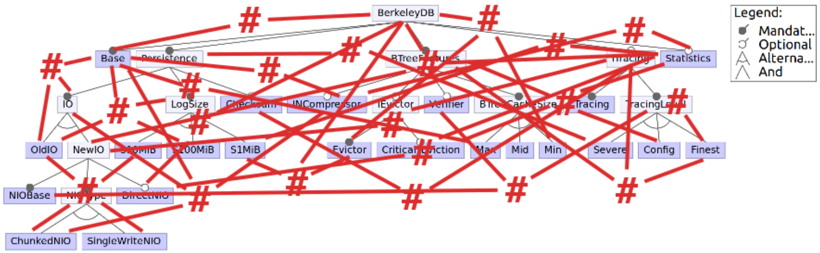


Interactions in Real-World Systems



- Questions
  - ***How many*** interactions?
  - Which ***order***?
  - Which ***visibility class***?
  - What is the ***frequency distribution*** of order and visibility?

Interactions in Real-World Systems



- Questions
  - ***How many*** interactions?
  - Which ***order***?
  - Which ***visibility class***?
  - What is the ***frequency distribution*** of order and visibility?
- Goals
  - ***Best strategies*** for interaction detection, management, and resolution
  - ***Relationships*** between interactions of different kind
  - ***Prediction*** of interactions

Preliminary Study

cppstats

TypeChef

SPL Conqueror

B. Garvin et al. ISSRE'11

2010 IEEE 21st International Symposium on Software Reliability Engineering

Detecting Dependences and Interactions in Feature-Oriented Design

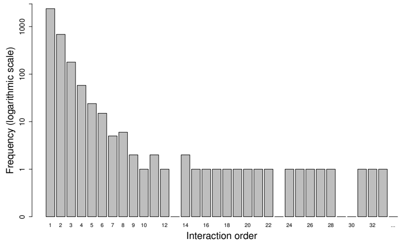
Subject Systems

	Visibility	$\lvert\mathcal{F}\rvert$	LOC	Description
LINUX	***Structural*** (code nesting)	9 102	5 986 427	OS kernel
BUSYBOX	***Operational*** (control flow)	792	191 615	UNIX utilities
GCC	***Functional*** (configuration faults)	171	2 648 177	Compiler collection
APACHE	***Non-functional*** (performance)	9	230 277	Web server

$\lvert\mathcal{F}\rvert$ – number of features

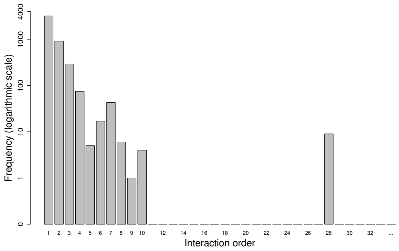
Distribution of Different Kinds of Feature Interactions

Structural



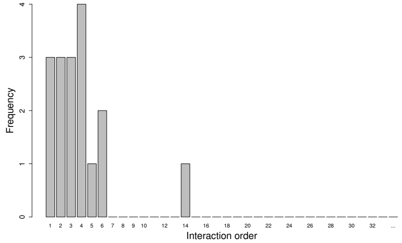
LINUX (code nesting)

Operational



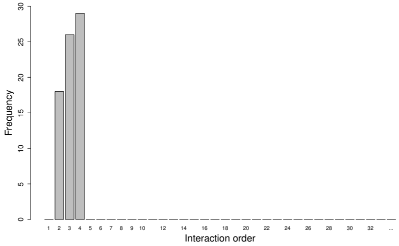
BUSYBOX (control flow)

Functional



GCC (configuration faults)

Non-functional



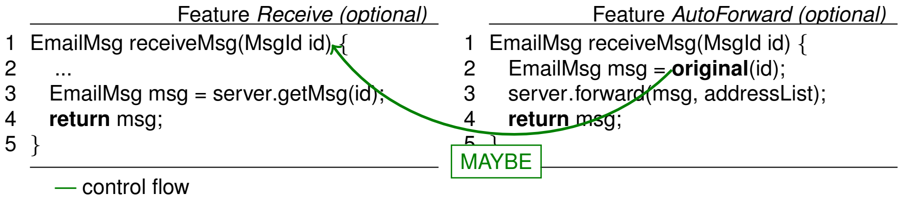
APACHE (performance)

Further Case Studies

- ***Functional interactions:***
  - ***14 systems***. Interaction faults up to the ***order of 7.***
    GCC, Apache, MySQL, OpenLDAP, RAX Planner . . .
- ***Non-Functional interactions:***
  - ***6 systems***. Performance interactions up to the ***order of 4.***
    Apache, BerkeleyDB-C, BerkeleyDB-Java, LLVM, SQLite, x264-codec.
- ***Structural interactions:***
  - ***3 systems***. Line coverage interactions up to the ***order of 6.***
    FTP-server, IRC-server, grep.
  - ***39 systems***. Code nesting up to the ***order of 33.***
    Apache, GCC, SQLite, BerkeleyDB . . .
- ***Operational interactions:***
  - ***5 systems***. Control-flow interactions up to the ***order of 28.***
    Apache, BerkeleyDB, Busybox, OpenSSL, SQLite.

SPL Type Errors vs. Static Attributes

- Product-line specific type errors



Feature *Receive (optional)*

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

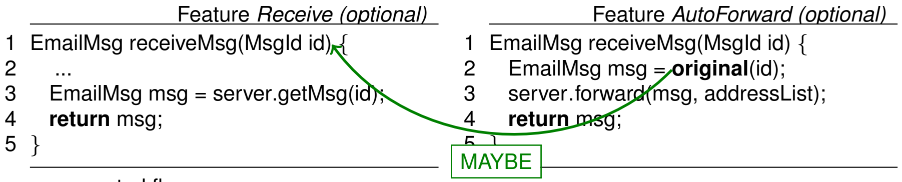
Feature *AutoForward (optional)*

```
EmailMsg receiveMsg(MsgId id) {
  EmailMsg msg = original(id);
  server.forward(msg, addressList);
  return msg;
}
```

— control flow

SPL Type Errors vs. Static Attributes

- Product-line specific type errors



Feature *Receive (optional)*

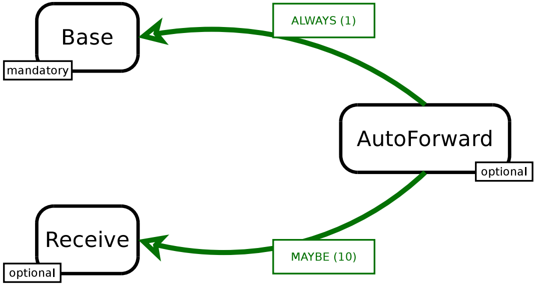
```
EmailMsg receiveMsg(MsgId id) {
   ...
  EmailMsg msg = server.getMsg(id);
  return msg;
}
```

Feature *AutoForward (optional)*

```
EmailMsg receiveMsg(MsgId id) {
  EmailMsg msg = original(id);
  server.forward(msg, addressList);
  return msg;
}
```

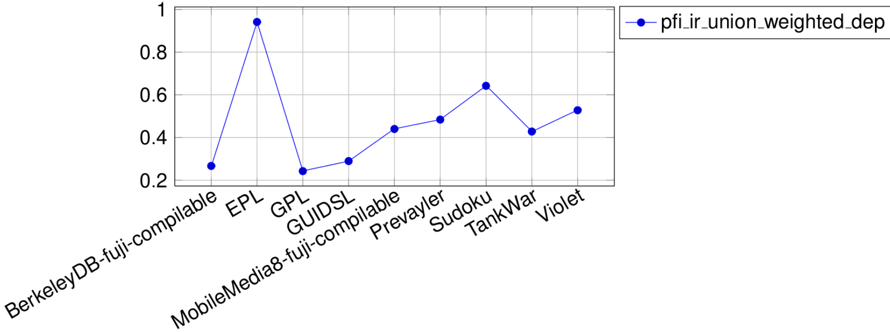
— control flow

- Predictor: Out degree measure on feature-call-graph



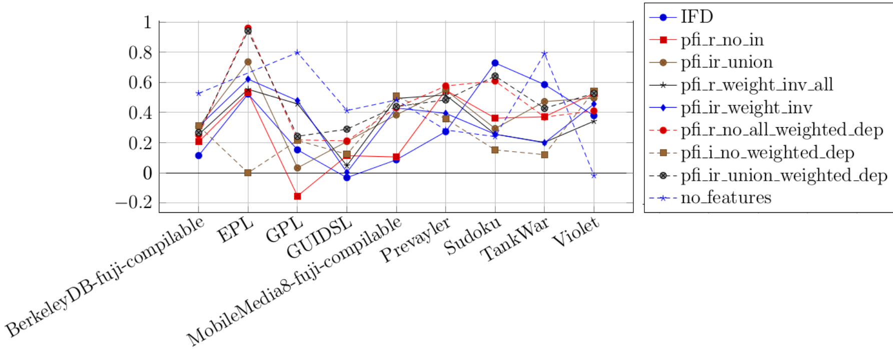
SPL Type Errors vs. Static Attributes (Evaluation)

- Correlation between number of type errors and measures:



SPL Type Errors vs. Static Attributes (Evaluation)

- Correlation between number of type errors and measures:



- 71 measures
- 29 Java product-lines

Conclusion

- Empirical study on interactions in real-world systems:
  - Questions
    - ***How many*** interactions?
    - Which ***order***?
    - Which ***visibility class***?
    - What is the ***frequency distribution*** of order and visibility?
  - Goals
    - ***Best strategies*** for interaction detection, management, and resolution
    - ***Relationships*** between interactions of different kind
    - ***Prediction*** of interactions