Emergent Feature Modularization

Márcio Ribeiro

Advisor: Paulo Borba
Co-advisor: Claus Brabrand
Software Dependencies

M1

M2

M3

M4

M5
String error = ... ;
...
...
...
...
#ifdef PDF
if (error.equals(“”)) {
   //enable PDF button
}
#endif
String[] error = ... ;
...
#ifdef PDF
if (error.equals("")) {
    //enable PDF button
}
#endif

Problem

- No mutual agreement

Behavioral Errors

- Increase Effort

Compilation Errors

- Incomplete fixes
Feature Dependency problem is critical in **Software Families** and **Product Lines**

How often?

To answer: Families/SPLs based on Preprocessors: *common* in *industrial* practice
Script tool

43 Families and SPLs

Results

<table>
<thead>
<tr>
<th>Metric</th>
<th>Average ± SD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Directives</td>
<td>11.26 ± 7.13</td>
</tr>
<tr>
<td>Dependencies</td>
<td>65.92 ± 18.54</td>
</tr>
</tbody>
</table>
## Script tool

### 43 Families and SPLs

## Results

<table>
<thead>
<tr>
<th>Metric</th>
<th>Average + SD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Directives</td>
<td>11.26 + 7.13</td>
</tr>
<tr>
<td>Dependencies</td>
<td>65.92 + 18.54</td>
</tr>
</tbody>
</table>
### Results

<table>
<thead>
<tr>
<th>Metric</th>
<th>Average ± SD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Directives</td>
<td>11.26% ± 7.13%</td>
</tr>
<tr>
<td>Dependencies</td>
<td>65.92% ± 18.54%</td>
</tr>
</tbody>
</table>
To address the problems discussed so far...

Emergent Interfaces!
String error = ... ;

...  

#ifdef PDF
if (error.equals("")) {
    //enable PDF button
}
#endif  

Be careful! You provide error for PDF feature  

Maintenance points  

Code analysis
Abstract feature code

Decrease effort

Now aware of dependencies

Minimize error introduction

Interfaces emerge!

Help to make VSoC realistic!

Be careful! You provide *error* for PDF feature
How to capture dependencies?!

Feature-oblivious dataflow analysis

void m() {
    int x = 0;
    #ifdef A
    x++;
    #endif
    #ifdef B
    x--;
    #endif
}

c = {A}

void m() {
    int x = 0;
    x++;
}

c = {B}

void m() {
    int x = 0;
    x--;
}

c = {A,B}

void m() {
    int x = 0;
    x++;
    x--;
}
Reducing costs...

Feature-sensitive dataflow analysis

Provides: *error* to d1

d1 Requires: *error*

String error = ...

if (error.equals("") ...

d1: {PDF}
d2: {}
Consecutive Feature-sensitive

CFG instrumented

Same lattice

TF: $\{\phi\} \subseteq c$

- $c = \{A\}$
  - $\bot$
  - $\{\text{: int } x = 0;\}$
  - $0$
  - $\{A\}: x++;$
  - $+$
  - $\{B\}: x--;$
  - $+$

- $c = \{B\}$
  - $\bot$
  - $\{\text{: int } x = 0;\}$
  - $0$
  - $\{A\}: x++;$
  - $0$
  - $\{B\}: x--;$
  - $-\$

- $c = \{A, B\}$
  - $\bot$
  - $\{\text{: int } x = 0;\}$
  - $0$
  - $\{A\}: x++;$
  - $+$
  - $\{B\}: x--;$
  - $0/+\$
Simultaneous Feature-sensitive

CFG instrumented

$\left[ \psi_{FM} \right] \rightarrow \mathcal{L}$

TF: point-wise

\[
\{A\} \mapsto 0, \{B\} \mapsto 0, \{A,B\} \mapsto 0
\]

\[
\{A\} : x++; \\
\]

\[
\{A\} \mapsto +, \{B\} \mapsto 0, \{A,B\} \mapsto +
\]
Emergo
Evaluation
Goal Question Metrics (GQM)

With Emergent Interfaces *versus* Without Emergent Interfaces

Q1: reduce effort?  
Q2: reduce errors?

- NoFa
- NoFe
- SLoC
- Time
- NE

**Empirical Study**  
**Controlled Experiments**
if (i < len && i >= 0) {
    TScreen *screen = TScreenOf(xw);
}

<table>
<thead>
<tr>
<th>Feature</th>
<th>SLoC</th>
</tr>
</thead>
<tbody>
<tr>
<td>2nd</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>48</td>
</tr>
<tr>
<td></td>
<td>21</td>
</tr>
</tbody>
</table>

NoFa = NoFe = 2
SLoC = 61

NoFa = NoFe = 3
SLoC = 82

NE = 1
NE = 2

Time = X seconds
Empirical Study

Raffle some methods with feature dependencies of each Family/SPL (3 replications)

Which methods?!
We force the selection of methods with few and many fragments

Group 1
1 or 2 fragments

Group 2
More than 2 fragments

300 100
Results
Question 1: Do Emergent Interfaces reduce effort?

String error = ... ;
...
#ifdef PDF
...
#elseif INTERRUPTION
...
#elseif WORD
...
#elseif XML
...
#elseif SIMILARITY_FILTER
...
#elseif SIMILARITY_FILTER
...
#endif
...
#endif
...
#endif
...
#endif
...
#endif

More gains in methods with many fragments

But they occur occasionally: 38% (Group 2)
So far we used proxy metrics...

Indirect measure of effort!
**Controlled Experiments**

\[ H_{10}: \mu_{\text{TimeNoEIs}} = \mu_{\text{TimeEIs}} \]

\[ H_{11}: \mu_{\text{TimeNoEIs}} \neq \mu_{\text{TimeEIs}} \]

\[ H_{20}: \mu_{\text{ErrorsNoEIs}} = \mu_{\text{ErrorsEIs}} \]

\[ H_{21}: \mu_{\text{ErrorsNoEIs}} \neq \mu_{\text{ErrorsEIs}} \]

Blocks two factors: student and task
Four Eclipses to avoid problems!
Collecting *Time* and *NE* automatically

Finish the task

Save all; Build automatically; Script execution

```
java -cp ... TestCase
```

Play / Pause time

- dry-run.properties
- e1-ei-bestlap.properties
To understand the effects in different types of tasks...

**Interprocedural**
- New requirement

**Intraprocedural**
- Unused variable

```c
f
  tS = ... ;
  ...
  #ifdef A
  g(tS);
  #endif
```

```c
g
  if (s < 0) {
    ...
  }
```

```c
h
  aM = ... ;
  ...
  #ifdef A
  new Al(aM);
  #endif
```
**Round 1:** Marburg, Germany (6 MSc/PhD students)
**Round 2:** Recife, Brazil (10 MSc/PhD students)
**Round 3:** Maceió, Brazil (14 UROP students)
Results:
New requirement task Interprocedural
New requirement task

Reject $H_0$ in both rounds!

3 times faster!

3.1 times faster!
Number of Errors (NE) per Subject

Subjects

Number of Errors (NE)

Round 2

Round 3
Results:
Unused variable task

Intraprocedural

To fix... #ifdefs!
No test cases!
Accomplished correctly?!

Time penalty = \[
\begin{align*}
\text{sd} & : \ NE = 2 \\
\text{sd}/2 & : \ NE = 1 \\
0 & : \ NE = 0
\end{align*}
\]
Unused variable task

<table>
<thead>
<tr>
<th>Technique</th>
<th>Time (seconds)</th>
<th>Round 1</th>
<th>Round 2</th>
<th>Round 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Els</td>
<td>1.68</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2mes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>faster!</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>No-Els</td>
<td>1.5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2mes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>faster!</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Reject $H_0$ in Round 3!

1.5 times faster!

1.68 times faster!
### Number of Errors (wrong feature expressions)

<table>
<thead>
<tr>
<th>Feature Expression</th>
<th>Els</th>
<th>No-Els</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>tires (Bestlap)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Els</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>No-Els</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td><strong>xP (Bestlap)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Els</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>No-Els</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td><strong>numOfViews (MMedia)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Els</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>No-Els</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td><strong>albumMusic (MMedia)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Els</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>No-Els</td>
<td>2</td>
<td>5</td>
</tr>
</tbody>
</table>

**Round 2**
- 75% without Els
- 25% with Els

**Round 3**
- 78% without Els
- 22% with Els
Question 1: Do Emergent Interfaces reduce effort?

Developers concentrate on the task

Despite different student levels, very similar results in both rounds

New requirement
Reject $H_0$ in both rounds!

Unused variable
Reject $H_0$ in Round 3!
Greater effort reduction for *interprocedural* dependencies...

≠

*Intra*procedural

≠

*Inter*procedural

Optimizations...

Caching algorithms...

Precision  Performance
Question 2: Do Emergent Interfaces reduce error introduction?

“If all such potential influenced code (either through control- or data-dependency) is clearly presented to developers, they may have better chances to detect the errors.”

How do fixes become bugs? Z. Yin et Al, FSE, 2011

In both rounds there is a tendency...

\[ N_{E_{\text{NoEIs}}} > N_{E_{\text{EIs}}} \]
Threats to validity

Time penalty / SD

Students

Missing dependencies (step by step?)

Maintenance tasks

MobileMedia

Precision versus Performance
Performance
Intraprocedural Compilation Fixed point

(a) Graph PL (b) MobileMedia08 (c) Lampaio (d) BerkeleyDB

(a) Graph PL (b) MobileMedia08 (c) Lampaio (d) BerkeleyDB

(a) Graph PL (b) MobileMedia08 (c) Lampaio (d) BerkeleyDB
Limitations

- Notion of feature dependency
- Performance Precision
- Dataflow analysis
- Experiment threats

Future Work

- Software development
- Improve expressiveness with JML
- Other mechanisms
- More experiments
Concluding Remark

Emergent Interfaces

Feature Modularization

No Feature Modularization
Emergent Feature Modularization

Márcio Ribeiro
Advisor: Paulo Borba
Co-advisor: Claus Brabrand
Related work

**Conceptual Module Querying for Software Reengineering**

**Granularity in Software Product Lines**

**Reducing Combinatorics in Testing Product Lines**

**Do Background Colors Improve Program Comprehension in the #ifdef Hell?**

**An Analysis of the Variability in Forty Preprocessor-Based Software Product Lines**
Algorithm complexity (First idea)

```c
method() {
  stmt1;
  stmt2;
  stmt3;
  stmt4;
  +#ifdef A
  stmt5;
  +#endif
  +#ifdef B
  stmt6;
  +#endif
}
```

Assumption:
- nested if statements = $O(1)$
- statements $\geq$ assignments
- $c =$ number of configurations

Worst case: $O(n^2 \cdot c)$

Best case: $O(n \cdot c)$
Nested if statements

FEATURES(stmt) ⊆ configuration

\{A,B\} ⊆ \{A,B,C\}

FEATURES(stmt) ∪ FEATURES(assign) ∈ FM

\{A,B\} ∈ \{{{A}, \{B\}, \{A,B\}}}

#define A && B
stmt;
#undef

#define A
assignment;
#undef

#define B
stmt;
#undef