Code-based Variability Model Extraction for Software Product Line Improvement

Bo Zhang
University of Kaiserslautern
Kaiserslautern, Germany

Martin Becker
Fraunhofer IESE
Kaiserslautern, Germany
## Variability Problems in SPLE

<table>
<thead>
<tr>
<th>Variability Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>• The variability model becomes inconsistent with its realization [Sincero+ 10]</td>
</tr>
<tr>
<td>- Dead features that can never be selected [Tartler+ 09]</td>
</tr>
<tr>
<td>• The variability model tends to be missing or incomplete in practice [Patzke+ 12]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Variability Realization</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Variability code becomes overly complex during SPL evolution [Krueger 10]</td>
</tr>
<tr>
<td>- Nested, tangled and scattered #ifdef blocks [Liebig+ 10]</td>
</tr>
<tr>
<td>• Variability code is difficult to understand and maintain</td>
</tr>
<tr>
<td>- Ambiguous variation points and inexplicit variant elements [Patzke+ 12]</td>
</tr>
</tbody>
</table>
Context: Conditional Compilation (CC)

- Conditional Compilation (CC) is widely used to implement variability, but
- Preprocessor code is difficult to understand and maintain
  - Nesting, tangling and scattering in #ifdef code [Liebig+ 10]
  - Undisciplined preprocessor annotations [Liebig+ 11]

Using CC to implement variability in a SPL

#ifdef code coloring by Feature Commander [FC]
SPL Improvement Process

• Based on the Reflexion Model approach [Murphy+ 01]
Defining Goals, Questions, and Metrics for SPL Improvement

GQM Model of SPL Improvement

Idea: Extracting Variability Model from Preprocessor Code
Variability Model Extraction

- Extracting Product Configuration → macro constants
- Extracting Variation Points → #ifdef blocks
- Extracting Variability Tree

Variability Code Using CC

1  #define A
2  #define B  30
3  #define C  -2
...  P1

1  #ifdef A
2  #if B > 20 && C < 0
3   Func_1();
4   #else
5   Func_2();
6   #endif
7  #else
8   Func_3();
9   #endif
10  ...

Variability Tree

- Variation Point
- Variant Element

Core

#ifdef A
node

#ifdef A
if B > 20 && C < 0

node

else
node

if B > 20 && C < 0

node

node

node

Func_3

Func_1

Func_2
Tool Support

- Implemented in Python scripts

- `cppVP.py`: Extracting variation points from CPP code
- `cppTree.py`: Building the variability tree and storing it in an XML file.
- `cppConst.py`: Parsing C code and extracting the definition and usage of macro constants
- `cppParsing.py`: Parsing C preprocessor statements
Extracting Product Configuration

- Parsing `#defines`
  - Assuming all constant names are unique in a SPL
  - Extracting macro constant names and their values if given

Variability Code Using CC
Industrial Case Study

- FreeRTOS: An embedded SPL from industry with 23K LOC
- Extracted product configuration (#defines)

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Cardinality</th>
<th>#References</th>
</tr>
</thead>
<tbody>
<tr>
<td>portSTACK_</td>
<td>1</td>
<td>-1</td>
<td>2</td>
</tr>
<tr>
<td>portUSING_</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>portBYTE_A</td>
<td>1</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>configKERN</td>
<td>1</td>
<td>0</td>
<td>255</td>
</tr>
<tr>
<td>portLARGE</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>portCRITIC</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>portCOMPAC</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>portSMALL</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>portPRESCA</td>
<td>0x00</td>
<td>16</td>
<td></td>
</tr>
</tbody>
</table>

Measurement of Macro Constants
Extracting Variation Points

- Parsing `#ifdefs`
  - **Type**: optional | alternative
  - **Branch**: positive | negative

```
1 #define A 2
2 #define B 30
3 #define C -2
...
```

```
1 #ifdef A
2   #if B > 20 && C < 0
3     Func_1();
4 #elif
5     Func_2();
6 #endif
7 #define A 2
8 #define B 10
9 #define C -5
...
```

Variability Code Using CC

```
1 ParseCode(code)
2 {
3     refList, tempRefList;
4     for each line in code {
5         ... // parsing #defines
6         if(line.match("#ifdef | ifndef | if | elif")) {
7             ref = new ConstRef(line);
8             ref.type = OPTIONAL;
9             if(line.match("#elif"))
10                tempRefList[-1].type = ALTERNATIVE;
11                ref.parent = tempRefList[-1]; // last element
12             if(ref.parent.type == OPTIONAL)
13                 ref.branch = POSITIVE;
14             else
15                 ref.branch = NEGATIVE;
16             tempRefList.append(ref);
17         }
18         else if(line.match("#else")) {
19             tempRefList[-1].type = ALTERNATIVE;
20             if(line.match("#endif")) {
21                 ref = tempRefList[-1];
22                 do {
23                     ref.end = line.end;
24                     refList.append(ref);
25                     tempRefList.delete(ref);
26                     ref = tempRefList[-1];
27                 } while(ref.prefix == "elif");
28             } // elseif
29         } // for each line in code
30     } // ParseCode
```
Industrial Case Study

- Extracted Variation Points (#ifdef blocks)

<table>
<thead>
<tr>
<th>Statement</th>
<th>refCount</th>
<th>fileCount</th>
<th>avgLevel</th>
<th>size (LOC)</th>
</tr>
</thead>
<tbody>
<tr>
<td>#ifdef__cp</td>
<td>112</td>
<td>56</td>
<td>2</td>
<td>112</td>
</tr>
<tr>
<td>ifndef PO</td>
<td>63</td>
<td>63</td>
<td>1</td>
<td>7734</td>
</tr>
<tr>
<td>#if conf~1</td>
<td>63</td>
<td>63</td>
<td>2</td>
<td>315</td>
</tr>
<tr>
<td>#if conf~2</td>
<td>42</td>
<td>37</td>
<td>1.07</td>
<td>434</td>
</tr>
<tr>
<td>#if conf~3</td>
<td>16</td>
<td>3</td>
<td>1.06</td>
<td>172</td>
</tr>
<tr>
<td>#if conf~4</td>
<td>15</td>
<td>11</td>
<td>1.33</td>
<td>829</td>
</tr>
<tr>
<td>#ifdef THU</td>
<td>13</td>
<td>13</td>
<td>1.38</td>
<td>245</td>
</tr>
</tbody>
</table>

Measurement of Variation Points
Extracting Variability Tree

- The tree structure is derived from `#ifdef nesting`
- Identical `#ifdef` statements are integrated into the same tree node

Variability Code Using CC:
```
1 #ifdef A
2    #if B > 20 && C < 0
3        Func_1();
4    #else
5        Func_2();
6    #endif
7 #endif
```

Variability Tree:
```
#ifdef A
   #if B > 20 && C < 0
      Func_1()
   #else
      Func_2()
   #endif
#endif
```

Variation Point
Variant Element

Variation Point
Variant Element
Extracting Variability Tree

• However, direct mapping from `#ifdef` to a tree node may cause circles
• Therefore, we distinguish identical `#ifdef` statement at different nesting levels
  – may cause redundancy

```
1 #ifdef A
2 #ifdef B
3 ...
4 #endif
5 #endif
```

```
1 #ifdef B
2 #ifdef A
3 ...
4 #endif
5 #endif
```

```
1 #ifdef A
2 #ifdef B
```

```
1 #ifdef B
2 #ifdef A
```

```
Root

#ifdef A
#ifdef B
```

```
Root

#ifdef B
#ifdef A
```

```
Root

#ifdef A
#ifdef B
```

```
Root

#ifdef B
#ifdef A
```

Industrial Case Study

- Extracted Variability Tree

XML Schema of the Variability Tree

Visualization of the Variability Tree using Treeviz [TV]
**Conclusion**

- SPL Improvement Process
- Variability Model Extraction
  - Extracting Product Configuration
  - Extracting Variation Points
  - Extracting Variability Tree

```
1 #define A
2 #define B  30
3 #define C -2
...
```

```
1 #ifdef A
2    #if B > 20 && C < 0
3        Func_1();
4    #else
5        Func_2();
6    #endif
7 #else
8      Func_3();
9 #endif
10 ...
```

Variability Code Using CC

---

Variability Tree

- Variation Point
- Variant Element
Future Work

• Variability Extraction
  – Identifying interdependencies between variation points
  – Extracting more abstract information (probably manually)
  – Investigating extraction from other mechanisms (e.g., conditional execution)

• SPL improvement
  – Finding concrete SPL improvement/refactoring approaches
  – Conducting evaluation with controlled experiments
References


Appendix: Reflexion Model approach [Murphy+ 01]

• Matching software design and implementation
  – Can be applied for checking design conformance, change assessment, etc.