CIDE+: A Semi-automatic Approach for Extracting Software Product Lines

Virgilio Borges, Rógel Garcia, Marco Túlio Valente

DCC – UFMG

CBSOFT 2010 - Tools Session
Motivation

- SPL extraction is a time-consuming task
  - Even starting from an existing codebase

- Two approaches for extracting SPL:
  - Compositional-based
  - Annotation-based
Compositional-based Approaches

- Example: aspects (AspectJ)

- Physical (“real”) modularization and separation of concerns
- Problem: costs do not outweigh the benefits

Fonte: Zhang, Jacobsen, OOPSLA 2004
Annotation-based Approaches

- Example: preprocessors

```java
boolean push(Object o) {
    Lock lk = new Lock();
    if (lk.lock() == null) {
        Log.log("lock failed");
        return false;
    }
    elements[top++]= o;
    size++;
    lk.unlock();
    if ((size % 10) == 0)
        snapshot("db");
    if ((size % 100) == 0)
        replicate("db","srv2");
    return true;
}
```

- It works. It is widely used.
- Problems: annotation hell; code pollution
Visual Annotations

- **CIDE: Colored IDE (Eclipse + background colors)**

```java
boolean push(Object o) {
    Lock lk = new Lock();
    if (lk.lock() == null) {
        Log.log("lock failed");
        return false;
    }
    elements[top++]= o;
    size++;
    lk.unlock();
    if ((size % 10) == 0)
        snapshot("db");
    if ((size % 100) == 0)
        replicate("db","srv2");
    return true;
}
```

- It works, generating less code pollution than #ifdefs
- Problem: colors assigned manually (repetitive, error-prone etc)
Our Tool: CIDE+

- Semi-automatic approach to assign colors to feature code

```java
boolean push(Object o) {
    Lock lk = new Lock();
    if (lk.lock() == null) {
        Log.log("lock failed");
        return false;
    }
    elements[top++] = o;
    size++;
    lk.unlock();
    if ((size % 10) == 0)
        snapshot("db");
    if ((size % 100) == 0)
        replicate("db","srv2");
    return true;
}
```
Input: Feature Seeds

- Program elements that implement an optional feature F
- When F is disabled, S can be removed from the code
- Example: logging

```
void log(String s) {
    ..... 
}
```

- Granularity: package, class, method, field

- Therefore, cannot dispense a meaningful knowledge about the internals of the target system
Annotation Algorithm

- Fixed point algorithm

- Two phases:
  - Color Propagation
  - Color Expansion
1st Phase: Propagation

- Marks all program elements that reference the seeds $S$ references to the seeds

```java
void log(String s) {
    ..... }
```

```
{ 
    log("stack overflow"); 
    ..... 
    ..... 
    log("lock failed"); 
    ..... 
    log("page delivered"); 
    ..... 
    ..... 

    log("new customer inserted"); 
    ..... 
    log("game over"); 
    ..... 
    ..... 
    log("user authenticated"); 
    ..... 
}
```
1st Phase: Propagation

- Marks all program elements that reference the seeds $S$

```java
void log(String s) {
    ......

    log("stack overflow");
    ......
    log("lock failed");
    ......
    log("page delivered");
    ......
    log("new customer inserted");
    ......
    log("game over");
    ......
    log("user authenticated");
    ......
}
```
Color Propagation Rules

1: \( \text{ColorPropagation(Package } p, \text{Color } c) = \)
2: \( \forall t \in \text{classes}(p) \rightarrow \text{ColorPropagation}(t, c); \)
3: \( \forall i \in \text{interfaces}(p) \rightarrow \text{ColorPropagation}(i, c); \)
4: \( \forall p = \text{import}(p.*) \rightarrow \text{cide}(p, c); \)

5: \( \text{ColorPropagation(Class } t, \text{Color } c) = \)
6: \( \forall m \in \text{meths}(t) \rightarrow \text{ColorPropagation}(m, c); \)
7: \( \forall f \in \text{fields}(t) \rightarrow \text{ColorPropagation}(f, c); \)
8: \( \forall s \in \text{extends}(t) \rightarrow \text{ColorPropagation}(s, c); \)
9: \( \forall v \in \text{hasType}(t) \rightarrow \text{ColorPropagation}(v, c); \)
10: \( \forall m \in \text{hasReturnType}(t) \rightarrow \text{ColorPropagation}(m, c); \)
11: \( \forall n = \text{new}(t) \rightarrow \text{cide}(n, c); \)
12: \( \forall p = \text{import}(t) \rightarrow \text{cide}(p, c); \)

13: \( \text{ColorPropagation(Interface } i, \text{Color } c) = \)
14: \( p = \text{declaration}(i) \rightarrow \text{cide}(p, c); \)
15: \( \forall t \in \text{impl}(i) \land \forall m \in \text{meths}(t) \land m \in i \rightarrow \text{ColorPropagation}(m, c); \)
16: \( \forall t \in \text{hasType}(i) \rightarrow \text{ColorPropagation}(t, c); \)
17: \( \forall m \in \text{hasReturnType}(t) \rightarrow \text{ColorPropagation}(m, c); \)
18: \( \forall p = \text{import}(i) \rightarrow \text{cide}(p, c); \)
Color Propagation Rules

19: \( \text{ColorPropagation}(Method \ m, \text{Color} \ c) = \)
20: \( p = \text{impl}(m) \rightarrow \text{cide}(p, c); \)
21: \( \forall s = \text{call}(m) \rightarrow \text{cide}(s, c); \)
22: \( \forall m' \in \text{overrides}(m) \rightarrow \text{ColorPropagation}(m', c). \)

23: \( \text{ColorPropagation}(Field \ f, \text{Color} \ c) = \)
24: \( d = \text{declaration}(f) \rightarrow \text{cide}(d, c); \)
25: \( \forall s = \text{access}(f) \rightarrow \text{cide}(s, c). \)

26: \( \text{ColorPropagation}(LocalVariable \ i, \text{Color} \ c) = \)
27: \( d = \text{declaration}(i) \rightarrow \text{cide}(d, c); \)
28: \( \forall s = \text{access}(i) \rightarrow \text{cide}(s, c). \)

29: \( \text{ColorPropagation}(FormalParam \ p, \text{Color} \ c) = \)
30: \( d = \text{declaration}(p) \rightarrow \text{cide}(d, c); \)
2nd Phase: Color Expansion

- Checks whether the enclosing context of the elements annotated in the previous phase can also be marked.

```java
void f(int x) {
    log("clicked" + x);
}
....
f(10);
....
f(z);
....
f(20);
```

after color propagation
2nd Phase: Color Expansion

- Checks whether the enclosing context of the elements annotated in the previous phase can also be marked.

```c
void f(int x) {
    log("clicked" + x);
}
....
f(10);
....
f(x);
....
f(20);
```

after color propagation

```c
void f(int x) {
    log("clicked" + x);
}
....
f(10);
....
f(x);
....
f(20);
```

color expansion
2nd Phase: Color Expansion

- Checks whether the enclosing context of the elements annotated in the previous phase can also be marked.

```java
void f(int x) {
    log("clicked" + x);
}
....
f(10);
....
f(z);
....
f(20);
```

- After color propagation

```java
void f(int x) {
    log("clicked" + x);
}
....
f(10);
....
f(z);
....
f(20);
```

- Color expansion

```java
void f(int x) {
    log("clicked" + x);
}
....
f(10);
....
f(z);
....
f(20);
```

- Expansion can trigger propagation
Color Expansion Rules

\[\text{BodyExpansion}() =\]
\[s = [\text{if(exp) stm}] \land \text{color(exp, c)} \rightarrow \text{cide}(s, c);\]
\[s = [\text{while exp stm}] \land \text{color(exp, c)} \rightarrow \text{cide}(s, c);\]
\[s = [\text{do stm while exp}] \land \text{color(exp, c)} \rightarrow \text{cide}(s, c);\]
\[s = [\text{case(exp) \{stm\}]} \land \text{color(exp, c)} \rightarrow \text{cide}(s, c);\]
\[s = [\text{switch(exp) \{stm\}]} \land \text{color(exp, c)} \rightarrow \text{cide}(s, c);\]
\[s = [\text{for(e} _1; e_2; e_3) \text{stm}] \land \text{color(e}_1, c) \land \text{color(e}_2, c) \land \text{color(e}_3, c) \rightarrow \text{cide}(s, c);\]

\[\text{ExpExpansion}() =\]
\[s = [\text{if(exp) stm}] \land \text{color(stm, c)} \land \text{free(exp)} \rightarrow \text{cide}(s, c);\]
\[s = [\text{while exp stm}] \land \text{color(stm, c)} \land \text{free(exp)} \rightarrow \text{cide}(s, c);\]
\[s = [\text{do stm while exp}] \land \text{color(stm, c)} \land \text{free(exp)} \rightarrow \text{cide}(s, c);\]
\[s = [\text{if(exp) s}_1 \text{ else s}_2] \land \text{color(s}_1, c) \land \text{color(s}_2, c) \land \text{free(exp)} \rightarrow \text{cide}(s, c);\]
\[s = [\text{case(exp) \{stm\}]} \land \text{color(stm, c)} \land \text{free(exp)} \rightarrow \text{cide}(s, c);\]
\[s = [\text{switch(exp) \{stm\}]} \land \text{color(stm, c)} \land \text{free(exp)} \rightarrow \text{cide}(s, c);\]
\[s = [\text{for(e} _1; e_2; e_3) \text{stm}] \land \text{color(stm, c)} \land \text{free(e}_1, e_2, e_3) \rightarrow \text{cide}(s, c);\]

\[\text{StmExpansion}() =\]
\[s = [\text{else stm}] \land \text{color(stm, c)} \rightarrow \text{cide}(s, c);\]
\[s = [\text{return exp}] \land \text{color(exp, c)} \rightarrow \text{cide}(s, c);\]
Color Expansion Rules

\[ StmExpansion() = \\
  s = \text{[else stm]} \land color(stm, c) \rightarrow cide(s, c); \\
  s = \text{[return exp]} \land color(exp, c) \rightarrow cide(s, c); \]

\[ MethExpansion() = \\
  s = \text{[t m(...)} \{ \text{stm}\}] \land color(stm, c) \rightarrow ColorPropagation(m, c); \]

\[ ClassExpansion() = \\
  s = \text{[class t { members }]}} \land color(members, c) \rightarrow ColorPropagation(t, c); \]

\[ AssignExpansion() = \\
  s = \text{[i = exp;]} \land color(i, c) \rightarrow cide(s, c); \]
Semi-automatic Expansions

- In some situations, the algorithm can lead to type/syntactic errors

```plaintext
if (option == k) {
    ....
}
```

- In other situations, it is complex to infer if expansion is safe

```plaintext
if (bar()) {
    stm;
}

x = y;
```

```plaintext
T foo() {
    ... // no returns
    return t;
}
```
Semi-automatic Expansions

- In some situations, the algorithm can lead to type/syntactic errors

```plaintext
if (option == k) {
    ...
}
```

- In other situations, it is complex to infer if expansion is safe

```plaintext
if (bar()) {
   stm;
}
x = y;
```

```
T foo() {
    ...
    // no returns
    return t;
}
```

```
if (bar()) {
   stm;
}
x = y;
```
Semi-automatic Expansions

- In some situations, the algorithm can lead to type/syntactic errors

```java
if (option == k) {
    ....
}
```

Syntax error

```java
T foo() {
    ... // no returns
    return t;
}
```

Type errors

- In other situations, it is complex to infer if expansion is safe

```java
if (bar()) {
    stm;
}
x = y; // bar() updates y
```

After propagation

```java
if (bar()) {
    stm;
}
x = y; // bar() updates y
```

Unsafe expansion
Semi-Automatic Expansions

- In the previous cases, we apply a default expansion (and generate a warning)

```java
if (option == k) {
    ....
}
```

**Syntax error**

- `if (option == k) {
    ....
}
```

**Type error**

- `T foo() {
    ...
    return t;
}
```

**Side effects**

- `if (bar()) {
    stm;
}  
x = y; // bar updates y
```

+ warning
# Semi-Automatic Expansion Rules

<table>
<thead>
<tr>
<th>Definition</th>
<th>Default Expansion</th>
</tr>
</thead>
<tbody>
<tr>
<td>SE1 Only parts of an expression have been annotated with a color c</td>
<td>Annotate the whole expression with c</td>
</tr>
<tr>
<td>SE2 The <strong>return</strong> statements of a method have been annotated with a color c; but the method has other statements that have not been annotated with c</td>
<td>Annotate the whole method body with c</td>
</tr>
<tr>
<td>SE3 In a call to a method ( m ), an actual parameter has been annotated with a color c; but the associated formal parameter has not</td>
<td>Annotate the whole method call with c</td>
</tr>
<tr>
<td>SE4 The right-hand side of an expression has been annotated with a color c; but the left-hand side has not</td>
<td>Annotate the left-hand side and its references with the color c</td>
</tr>
<tr>
<td>SE5 Color expansion E2 (Figure 6) has not been applied (using a color c) because it was not possible to infer whether the expression exp is side effect free</td>
<td>Annotate ( \text{exp} ) with c</td>
</tr>
</tbody>
</table>
Extraction Process

- **Seeds**
  - Define (or refine) feature seeds

- **Annotation Algorithm**
  - Execute annotation algorithm

- **Evaluate Results**
  - Fix errors due to non-accepted default expansions
  - Evaluate annotation coverage

- **Semi-automatic Expansions**
  - Authorize (or not) default expansions
Evaluation

- Three systems:
  - ArgoUML
  - JFreeChart
  - Prevayler
ArgoUML Example

- Manual extraction:
  - Developer with no knowledge about our algorithm
  - Using conditional compilation; then imported to CIDE
  - Public at: http://argouml-spl.tigris.org

<table>
<thead>
<tr>
<th>Features</th>
<th>KLOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>State Diagrams</td>
<td>3.9</td>
</tr>
<tr>
<td>Activity Diagrams</td>
<td>2.2</td>
</tr>
<tr>
<td>Design Critics</td>
<td>16.3</td>
</tr>
<tr>
<td>Logging</td>
<td>2.5</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>24.9</strong></td>
</tr>
</tbody>
</table>
ArgoUML

- Semi-automatic extraction using CIDE+:

<table>
<thead>
<tr>
<th>Feature</th>
<th>Seeds</th>
<th># Iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>State Diagram</td>
<td>Two packages and two classes</td>
<td>2</td>
</tr>
<tr>
<td>Activity Diagram</td>
<td>Three packages and one class</td>
<td>1</td>
</tr>
<tr>
<td>Design Critics</td>
<td>Ten packages and two classes</td>
<td>2</td>
</tr>
<tr>
<td>Logging</td>
<td>One package</td>
<td>2</td>
</tr>
</tbody>
</table>
ArgoUML Results

<table>
<thead>
<tr>
<th>Feature</th>
<th>KB</th>
<th></th>
<th>Precision</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$M - A$</td>
<td>$M \cap A$</td>
<td>$A - M$</td>
<td></td>
</tr>
<tr>
<td>State Diagram</td>
<td>38.4</td>
<td>290.8</td>
<td>42.1</td>
<td>0.87</td>
</tr>
<tr>
<td>Activity Diagram</td>
<td>6.3</td>
<td>142.3</td>
<td>9.4</td>
<td>0.94</td>
</tr>
<tr>
<td>Design Critics</td>
<td>54.5</td>
<td>1,211.7</td>
<td>8.8</td>
<td>0.99</td>
</tr>
<tr>
<td>Logging</td>
<td>3.7</td>
<td>106.8</td>
<td>12.6</td>
<td>0.89</td>
</tr>
</tbody>
</table>

(M = Manual extraction; A = semi-automatic extraction)

- **recal ≤ 100%**:  
  - Limitations of the defined seeds in reaching all manual marked code  
  - Example: XML parser that process a ToDo list
- **precision ≤ 100%**:  
  - In many parts of the code the developer in charge of the manual extraction has not expanded an annotation to its enclosing context
ArgoUML: Semi-automatic Exp.

<table>
<thead>
<tr>
<th>Rules</th>
<th>State</th>
<th>Activity</th>
<th>Critics</th>
<th>Logging</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>SE1</td>
<td>20(1)</td>
<td>18(1)</td>
<td>33</td>
<td>2</td>
<td>73</td>
</tr>
<tr>
<td>SE2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SE3</td>
<td>23</td>
<td>7</td>
<td>35</td>
<td>1</td>
<td>66</td>
</tr>
<tr>
<td>SE4</td>
<td>49</td>
<td>14</td>
<td>44</td>
<td>1(1)</td>
<td>108</td>
</tr>
<tr>
<td>SE5</td>
<td>8</td>
<td>1</td>
<td>24</td>
<td>15</td>
<td>48</td>
</tr>
<tr>
<td>Total</td>
<td>100</td>
<td>40</td>
<td>136</td>
<td>19</td>
<td>295</td>
</tr>
</tbody>
</table>

- In just three cases the developer has not accepted the default actions associated to the proposed semi-automatic expansions
Non-accepted Default Actions

```java
cls = org.apache.log4j.Logger.class;
....
cls = Class.forName("...");
```

The RHS of an expression has been annotated with a color c; but the LHS has not
Non-accepted Default Actions

```
cls = org.apache.log4j.Logger.class;
....
cls = Class.forName("...");
```

The RHS of an expression has been annotated with a color c; but the LHS has not

**default action:** annotate the LHS and its references

```
cls = org.apache.log4j.Logger.class;
....
cls = Class.forName("other concern");
```

+ warning
Non-accepted Default Actions

The RHS of an expression has been annotated with a color c; but the LHS has not.

default action: annotate the LHS and its references

Manual action: undo default action; annotate only the first assignment
Reason: local variable is reused to store other Class Values

The RHS of an expression has been annotated with a color c; but the LHS has not.

default action: annotate the LHS and its references

Manual action: undo default action; annotate only the first assignment
Reason: local variable is reused to store other Class Values

The RHS of an expression has been annotated with a color c; but the LHS has not.

default action: annotate the LHS and its references

Manual action: undo default action; annotate only the first assignment
Reason: local variable is reused to store other Class Values

The RHS of an expression has been annotated with a color c; but the LHS has not.

default action: annotate the LHS and its references

Manual action: undo default action; annotate only the first assignment
Reason: local variable is reused to store other Class Values
## Similar results: recall, precision, semi-automatic expansions

<table>
<thead>
<tr>
<th>Feature</th>
<th>$M - A$</th>
<th>$M \cap A$</th>
<th>$A - M$</th>
<th>Precision</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monitor</td>
<td>0</td>
<td>6,725</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Censorship</td>
<td>0</td>
<td>4,393</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Replication</td>
<td>715</td>
<td>11,175</td>
<td>0</td>
<td>1</td>
<td>0.94</td>
</tr>
</tbody>
</table>

(M= Manual extraction; A= semi-automatic extraction)
JFreeChart

<table>
<thead>
<tr>
<th>Feature</th>
<th>Bytes</th>
<th>Precision</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$M - A$</td>
<td>$M \cap A$</td>
<td>$A - M$</td>
</tr>
<tr>
<td>Pie Charts</td>
<td>1,005</td>
<td>383,165</td>
<td>0</td>
</tr>
<tr>
<td>3D Charts</td>
<td>382</td>
<td>172,444</td>
<td>0</td>
</tr>
</tbody>
</table>

(M= Manual extraction; A= semi-automatic extraction)

<table>
<thead>
<tr>
<th>Rules</th>
<th>Pie Charts</th>
<th>3D Charts</th>
</tr>
</thead>
<tbody>
<tr>
<td>SE1</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>SE2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SE3</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>SE4</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>SE5</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Total</td>
<td>21</td>
<td>7</td>
</tr>
</tbody>
</table>

Semi-automatic expansions (all default actions have been accepted)
Lessons Learned

- SPL extraction is a time-consuming and complex task

- CIDE: promising tool to extract “real” SPLs

- CIDE+: accelerates SPL extraction using CIDE

- However:
  - Developers should be familiar with the target system
  - Developers should carefully select the feature seeds
  - Number of semi-automatic expansions is significant
Related Tools

- Compositional-based approaches (e.g. aspects)
  - Example: AOP-Migrator
  - Do not scale to real, complex SPLs

- Annotation-based approaches (e.g. preprocessors)
  - No tools!
Conclusions

- CIDE+ has been successfully applied in three non-trivial systems
- CIDE+ provides degrees of automation well above existent tools
  - Particularly, when compared with tools based on aspects
- More details, including source code at:
  - www.dcc.ufmg.br/~mtov/cideplus
Thanks