Safe evolution of software product lines

Paulo Borba
Informatics Center
Federal University of Pernambuco

phmb@cin.ufpe.br
Product families are everywhere...

No Clouds  |  Static Clouds  |  Moving Clouds

Tuesday, February 14, 12
Software product lines come to the rescue...

reusable assets

related products
But what do we know about software product line (SPL) evolution?
Supporting new products

Rain of fire

Image loading

On demand  Startup

Rain of fire

Clouds

Static  Dynamic  On demand  Startup

Rain of fire

Image loading

On Demand  Start Up

Rain.java, CC.java  Main 2, On demand  Main 1, Startup.aj

Clouds.java

Dynamic  Static
Refactoring or fixing artifacts

Rain of fire

Clouds
- Static
- Dynamic

Image loading
- On demand
- Startup

Rain.java, CC.java

On Demand
Rain of fire

Start Up
Rain of fire

Clouds
- Static
- Dynamic

Image loading
- On demand
- Startup

Rain.java, CC.java

Main 1, Startup.aj

Clouds
- Static
- Dynamic

Image loading
- On demand
- Startup

Rain.java, CC.java

Main 2, On demand

Tuesday, February 14, 12
Extracting a SPL

Rain of fire
Clouds

<table>
<thead>
<tr>
<th>Rain of fire</th>
<th>Rain.java, CC.java</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clouds</td>
<td>Clouds.java</td>
</tr>
</tbody>
</table>
Is the evolution safe?
Does it impact users?
Safety during extension

No Clouds

Moving Clouds

Static Clouds

Rain of fire

Image loading

On demand

Startup

Clouds

Static

Dynamic

On demand

Startup

Rain of fire

Rain.java, CC.java

Main 2, On demand

Main 1, Startup.aj

Rain of fire

Rain.java, CC.java

Main 2, On demand

Main 1, Startup.aj

Clouds

Clouds.java

Main 1, Startup.aj

Startup

On demand
Safety during refactoring

- No Clouds
- Static Clouds
- Moving Clouds

Rain of fire

Image loading

Static
Dynamic
On demand
Startup

Rain java, CC.java
Main 2, On demand
Main 1, Startup.aj
Clouds.java

Startup
On demand
Main 2, On demand
Main 1, Startup.aj
Clouds.java

No Clouds
Moving Clouds
Static Clouds
Moving Clouds

Tuesday, February 14, 12
Safety during extraction

No Clouds

Static Clouds

Rain of fire

Clouds

<table>
<thead>
<tr>
<th>Rain of fire</th>
<th>Rain.java, CC.java</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clouds</td>
<td>Clouds.java</td>
</tr>
</tbody>
</table>

No Clouds

Static Clouds

Tuesday, February 14, 12
Safety during progressive porting
Focus on the generated products!

PL products

PL' products

1

2

3

4

5

6

1

2

3

4

5

6

Tuesday, February 14, 12
Beyond program refinement and refactoring

- Rain of fire
  - Clouds
    - Static
    - Dynamic
  - Image loading
    - On demand
    - Startup

<table>
<thead>
<tr>
<th>Rain of fire</th>
<th>Rain.java, CC.java</th>
</tr>
</thead>
<tbody>
<tr>
<td>On Demand</td>
<td>Main 2, On demand</td>
</tr>
<tr>
<td>Start Up</td>
<td>Main 1, Startup.aj</td>
</tr>
<tr>
<td>Clouds</td>
<td>Clouds.java</td>
</tr>
</tbody>
</table>
Eclipse refactoring...

AlternativeC
...c.m()...

Common
m() {...a.n()...}

AlternativeS
n() {...}

AlternativeC
...c.m()...

AlternativeS
n() {...}
might break a SPL

AlternativeC
...c.m()...

AlternativeC
...c.m()...

Common
m() {...a.n()...}

AlternativeS
n() {...}

AlternativeS
n() {...}
Not SPL aware

AlternativeC
...c.m()...

Common
m() {...a.n()...}

AlternativeS
n() {...}

AlternativeC
...c.m()...

Common
m() {...a.n()...}

AlternativeS
n() {...}

AlternativeC
...c.m()...

Common
m() {...a.n()...}

AlternativeS
n() {...}
Agenda

- SPL refinement theory
- SPL refinement templates
- SPL refinement checking
Refinement theory

• Captures our notion of safe evolution
• Independent of SPL languages and semantic encodings
• Axioms and assumptions define the interface between the theory and such languages
‘Feature model’ interface

\[ F \cong F' \text{ whenever } \llbracket F \rrbracket = \llbracket F' \rrbracket \]

\[ \llbracket : \text{FeatureModel} \rightarrow P[\text{Configuration}] \]

\[ F \cong F' \text{ whenever } \llbracket F \rrbracket = \llbracket F' \rrbracket \]

\[ \cong \text{ is an equivalence} \]
FM language independence

\[
F \simeq F' \land A \subseteq A' \land K \simeq K' \\
(F, A, K) \subseteq (F', A', K')
\]

\{Clouds, Static, Startup\}

\{(Add, 1), (ValueType, 'Int')\}

Tuesday, February 14, 12
Assets

Any kind of asset with a compositional notion of refinement

(preservation of observable behavior)
Asset language interface

\[ \subseteq : P[\text{Asset}], P[\text{Asset}] \rightarrow \text{bool} \]

\[ \text{wf} : P[\text{Asset}] \rightarrow \text{bool} \]
Independence of asset language and refinement notion

\[ F \simeq F' \land A \subseteq A' \land K \simeq K' \Rightarrow (F, A, K) \subseteq (F', A', K') \]

\[ w_f(as) \]

\[ as \subseteq as' \]
As long as it satisfies a compositionality axiom

\[ \forall as, as', s : \mathbb{P}[\text{Asset}] \cdot \]

\[ as \sqsubseteq as' \land \text{wf}(as \cup s) \implies \]

\[ \text{wf}(as' \cup s) \land as \cup s \sqsubseteq as' \cup s \]
Assets mapping

Main ↦ class Main {
  ...new StartUp(...);
}

StartUp

Main ↦ class Main {
  ...new OnDemand(...);
}

On demand
Asset mapping refinement

\[ A \sqsubseteq A' \]

when

\[ \text{dom}(A) = \text{dom}(A') \]
\[ \forall n \in \text{dom}(A) \cdot A(n) \sqsubseteq A'(n) \]
CK evaluation

<table>
<thead>
<tr>
<th>{Start Up}</th>
<th>{Main 1, Startup, Common.java, ...}</th>
</tr>
</thead>
<tbody>
<tr>
<td>On Demand</td>
<td>Main 2, On demand</td>
</tr>
<tr>
<td>Start Up</td>
<td>Main 1, Startup</td>
</tr>
<tr>
<td>On Demand v Start Up</td>
<td>Common.java</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

```
class Main {
    ...new StartUp(...);...
}
```

```
class Main {
    ...new OnDemand(...);...
}
```
CK interface...

\[ \mathcal{F}[\text{Asset}] : \mathcal{C} \mathcal{K} \rightarrow \mathcal{A} \mathcal{M} \rightarrow \text{Configuration} \rightarrow \]

\[ K \cong K' \text{ whenever } [K] = [K'] \]

\[ \cong \text{ is an equivalence} \]
with compositionality axiom

\[ A \subseteq A' \]

implica

\[ \forall K, c . \quad \text{wf}(\llbracket K \rrbracket_c^A) \Rightarrow \]

\[ \text{wf}(\llbracket K \rrbracket_c^{A'}) \land \llbracket K \rrbracket_c^A \subseteq \llbracket K \rrbracket_c^{A'} \]
CK language independence

\[
[K]_C^A
\]

\[
F \cong F' \land A \sqsubseteq A' \land K \cong K' \\
\Rightarrow \\
(F, A, K) \sqsubseteq (F', A', K')
\]

Feature Expression | Assets
--- | ---
"int" in ValueType | task involving int
"float" in ValueType | task involving float
"int" \(\cup\) "float" not in ValueType | task involving anything else than int or float
some ValueType | foreach \(v\) : ValueType do something with \(v\)
\#ValueTypes > 1 | do something when multiple types in the EPL
A PL is a tuple...

$$(F, A, K)$$

such that

$$\forall c \in \llbracket F \rrbracket \cdot wf(\llbracket K \rrbracket_A^c)$$
Refinement beyond source code...

SPL renaming refinement
and allowing extension without impacting user base...

add optional feature refinement
Product line refinement

\[(F, A, K) \sqsubseteq (F', A', K')\]

whenever

\[\forall c \in \llbracket F \rrbracket \cdot \exists c' \in \llbracket F' \rrbracket \cdot \llbracket K \rrbracket^A_c \sqsubseteq \llbracket K' \rrbracket^A_{c'}\]
Focus on products, not feature names

{Music, 240x320}
{Common.aj, Music.java, ...}
...class Music {...}...

{Audio, 240x320}
{Common.aj, Audio.java, ...}
...class Audio {...}...
Safety for existing users

{Music, 240x320}
{Common.aj, Music.java, ...}
class Music {...}

{Music, 240x320}
{Common.aj, Music.java, ...}
class Music {...}
No guarantees for new users

{Music, 240x320}

{Common.aj, Music.java, ...}

class Music {...}
Useful evolution, but not refactoring
Refinement applies when the intention is to...

• improve internal structure
• increase configurability
• preserve observable behavior
Reducing configurability, not refinement

{Music, 128x149}
Not preserving behavior, not refinement

{Music, 240x320}  {Music, 240x320, Copy}
FM compositionality

\[ F \cong F' \]

implies

\[ (F, A, K) \sqsubseteq (F', A, K) \]
CK compositionality

\[ K \cong K' \]

implies

\[ (F, A, K) \sqsubseteq (F, A, K') \]
AM compositionality

\[ A \subseteq A' \]

implies

\[ (F, A, K) \subseteq (F, A', K) \]
Compositionality theorem

\[ F \equiv F' \land A \sqsubseteq A' \land K \equiv K' \]

\[ \Rightarrow \]

\[ (F, A, K) \sqsubseteq (F', A', K') \]
SPL refinement theory

\[ F \cong F' \land A \subseteq A' \land K \cong K' \]

\[ \Rightarrow \]

\[ (F, A, K) \subseteq (F', A', K') \]
Instantiating the theory

\[ \begin{align*}
[F] &\quad [K]_c^A \\
\text{Rain of fire} &\quad \text{Rain.java, CC.java} \\
\text{On Demand} &\quad \text{Main 2, On demand} \\
\text{Start Up} &\quad \text{Main 1, Startup.aj} \\
\text{Clouds} &\quad \text{Clouds.java} \\
\end{align*} \]

\[ F \cong F' \land A \subseteq A' \land K \cong K' \quad \Rightarrow \quad (F, A, K) \subseteq (F', A', K') \]

\[ \forall f (as) \quad as \subseteq as' \]
Refinement catalogue...

allows us to forget the formal definitions of refinement and semantics when performing refinement activities
SPL refinement laws: add optional feature

\[ O \notin \text{features}(F) \]

resulting PL is well-formed

\[ m \text{ does not map names from } A \]

each feature expressions in its implies \( O \)
Proving soundness

∀ F, A, K, F', A', K'... ·

wfPL(F, A, K) ∧ syntax(...) ∧ conditions(...)

⇒ wfPL(F', A', K') ∧

(F, A, K) ⊑ (F', A', K')

relies on specific languages and semantic encodings
Mining evolution scenarios

<table>
<thead>
<tr>
<th></th>
<th>TaRGeT</th>
<th>Mobile Media</th>
<th>RGMSs</th>
</tr>
</thead>
<tbody>
<tr>
<td>scenarios</td>
<td>20</td>
<td>8</td>
<td>15</td>
</tr>
<tr>
<td>releases</td>
<td>15</td>
<td>4</td>
<td>15</td>
</tr>
<tr>
<td>features</td>
<td>42</td>
<td>12</td>
<td>20</td>
</tr>
<tr>
<td>KLOC</td>
<td>32</td>
<td>3</td>
<td>3.5</td>
</tr>
</tbody>
</table>
Split asset

\[\{n \rightarrow a\} \oplus m \subseteq F \quad \{n \rightarrow a', n' \rightarrow a''\} \oplus m \]

\[
\begin{array}{c}
\text{e} \\
\hline
\text{n} \\
\hline
\text{its}
\end{array}
\]

\[
\begin{array}{c}
\text{e} \\
\hline
\text{n, n'} \\
\hline
\text{its}
\end{array}
\]

\[a \sqsubseteq a' a''\]

\(n\) and \(n'\) do not appear in \(\text{its}\)
Adding void mandatory feature

\[ F \subseteq features(F) \]
**CK transformations**

<table>
<thead>
<tr>
<th>e</th>
<th>n</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[ \equiv \]

<table>
<thead>
<tr>
<th>( e' )</th>
<th>n</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[ \equiv \]

<table>
<thead>
<tr>
<th>e</th>
<th>n</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[ e \Leftrightarrow e' \]

<table>
<thead>
<tr>
<th>( e' )</th>
<th>( n' )</th>
</tr>
</thead>
<tbody>
<tr>
<td>e</td>
<td>n</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[ \equiv \]

<table>
<thead>
<tr>
<th>e</th>
<th>n</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[ \equiv \]

<table>
<thead>
<tr>
<th>( e' )</th>
<th>( n' )</th>
</tr>
</thead>
<tbody>
<tr>
<td>e</td>
<td>n</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[ e \Leftrightarrow e' \]
Feature model transformations

\[ P \implies \neg R \land R \implies \neg Q \]
\[ Q \implies \neg R \land R \implies \neg Q \]
\[ P \implies (Q \lor R) \]
\[ Q \implies P \]
 Asset transformations

\[ \text{eds} \]
\[
\text{class C} \{ \\
\text{fds} \\
\text{mds} \\
T \text{m(pds)} \{ \\
\text{body'};} \\
\text{body} \\
\} \\
\} \equiv \\
\text{eds} \]
\[
\text{class C} \{ \\
\text{fds} \\
\text{mds} \\
T \text{m(pds)} \{ \\
\text{body} \\
\} \\
\} \equiv \\
\text{priv aspect A} \{ \\
\text{before(context)} : \\
\text{exec(C.m) \&\&} \\
\text{bind(context)} \{ \\
\text{body'}[cthis/this] \\
\} \\
\} \]

\(\rightarrow\) body’ does not declare or use local variables; body’ does not call super;...
Automatic SPL refinement checking

\[
[F] \\ [K] \\
\frac{F \equiv F' \land A \subseteq A' \land K \equiv K'}{(F, A, K) \subseteq (F', A', K')}
\]

\[
f(a) = a' \\
\text{as} \sqsubseteq \text{as}'
\]

\[
F \subseteq A \subseteq K \\
M \notin \text{features}(F)
\]
When changes are on FM and CK...

FM and CK equivalence checking
SPL refinement approximation

testing individual assets or products
Conclusions

• Language independent theory of SPL refinement
• Safe, stepwise and compositional SPL development and evolution
• Sound refinement laws
• Tools for refinement checking
Thank you!
Safe evolution of software product lines

Paulo Borba
Informatics Center
Federal University of Pernambuco

phmb@cin.ufpe.br