Implementing Software Product Line Adoption Strategies

Candidate: Vander Alves
Supervisors: Paulo Borba and Geber Ramalho
March 23, 2007
Different devices, 15 to 60 different applications...

With plenty of common functionality, but also with variability

<table>
<thead>
<tr>
<th></th>
<th>64kb, flip</th>
<th>4Mb, flip</th>
<th>100Kb, no flip</th>
</tr>
</thead>
</table>
Little reuse and high cost
Goal: automate product line

Strategic reuse of common artifacts and handling of variability
Software Product Lines (SPL)

Source: SEI
GO SPL!

- SPL benefits
  - Higher productivity
  - Higher quality
  - Low cost

- But higher complexity
  - Variability management
  - Adoption strategies
SPL challenges

- Variability management
- Adoption strategy
Variability management

- Highly variant domains
  - Realm of SPL
  - Different artifacts
  - Often crosscutting implementation
  - high instantiation cost
Mobile games domain

- Screen Sizes
- Keyboard
- Specific APIs

- Memory
- Processing Power
- Number of colors

Porting

Porting
Massive variability
Aspect-Oriented Programming

- Extension to OO
- Modularizing crosscutting concerns
  - Tangling
  - Scattering
- Language and tool support
- Different levels of abstraction (AOSD, Early Aspects,...)
Adoption Strategies

- Proactive
- Extractive
- Reactive
Proactive Adoption Strategy

Source: Krueger, PFE’01
Reactive Adoption Strategy

Source: Krueger, PFE’01
Extractive Adoption Strategy

Source: Krueger, PFE’01
Problem

- Reactive and Extractive very relevant
- Lack of guidelines
  - Implementation
  - Feature model
- Essential requirements
  - Concrete guidelines
  - Safety
Key idea

- Method to create and evolve product lines
  - Combine adoption strategies
  - Refactorings to guide the process
  
Concrete guidelines
Safety
Our Method

1. Identify Variability
2. Extract SPL
3. Migrate SPL
4. React SPL
5. Refactor Feature Model
6. Update Configuration Knowledge
Our Method

Identify Variability

Extract SPL

Migrate SPL

React SPL

Refactor Feature Model

Update Configuration Knowledge
Extract SPL

Product 1 → refactoring → Aspect P1 → Core
Product 2 → Classes P1 → Aspect P2

Product1 = \{\text{Core, Classes P1}\} \bullet \text{Aspect P1}
Product2 = \{\text{Core, Classes P2}\} \bullet \text{Aspect P2}
Our Method

- Identify Variability
- Extract SPL
- Migrate SPL
- Refactor Feature Model
- React SPL
- Update Configuration Knowledge
React SPL

Product 1 = \{ \text{Core}', \text{Classes P1} \} \bullet \text{Aspect P1}'

Product 2 = \{ \text{Core}', \text{Classes P2} \} \bullet \text{Aspect P2}'

Product 3 = \{ \text{Core}', \text{Classes P3} \} \bullet \text{Aspect P3}
Further React SPL

Product 1 = \{Core', Classes P1' \} \bullet \{Aspect P1'', AspectFlip\}
Product 2 = \{Core', Classes P2' \} \bullet \{Aspect P2'', AspectFlip\}
Product 3 = \{Core', Classes P3 \} \bullet Aspect P3
Our Method

- Identify Variability
- Extract SPL
- Migrate SPL
- React SPL
- Refactor Feature Model
- Update Configuration Knowledge
Migrate SPL

Product1 = {Core', Classes P1 } • Aspect P1'
Product2 = {Core', Classes P2 } • Aspect P2'
Product3 = {Core', Classes P3 } • Aspect P3
Method properties

- Extractive + reactive approaches
- AOP-based variation management
  - isolate crosscutting variations from the core
  - reuse opportunity
- Refactorings to guide the process
  - Concrete and declarative guidelines
  - Safety
  - Derived from elementary AOP laws
Example - extractive refactoring

- Load images variation
  - Images are loaded on demand
  - Images loaded only once
Extractive Refactoring

- Load images variation point

```java
public void changeScreen(int screenId) {
    if (screenId == Resources.SPLASH_SCREEN) {
        Resources.loadSplashImages();
    } else if (screenId == Resources.GAME_SCREEN) {
        Resources.loadGameScreenImages();
    } else if (screenId == Resources PRESENTATION_SCREEN) {
        Resources.loadPresentationImages();
    } else if (screenId == Resources.HI_SCORE_SCREEN) {
        Resources.loadMenuHScoresImages();
    } else {
        Resources.loadMenuImages();
    }
    this.currentScreenId = screenId;
}
```
public void changeScreen(int screenId) {
    if (screenId == Resources.SPLASH_SCREEN) {
        Resources.loadSplashImages();
    } else if (screenId == Resources.GAME_SCREEN) {
        Resources.loadGameScreenImages();
    } else if (screenId == Resources.PRESENTATION_SCREEN) {
        Resources.loadPresentationImages();
    } else if (screenId == Resources.HI_SCORE_SCREEN) {
        Resources.loadMenuHScoresImages();
    } else {
        Resources.loadMenuImages();
    }
    this.currentScreenId = screenId;
}

public void changeScreen(int screenId) {
    this.currentScreenId = screenId;
}

before(int screenId): execution(void MainCanvas.changeScreen(int)) & args(screenId) {
    if (screenId == Resources.SPLASH_SCREEN) {
        Resources.loadSplashImages();
    } else if (screenId == Resources.GAME_SCREEN) {
        Resources.loadGameScreenImages();
    } else if (screenId == Resources.PRESENTATION_SCREEN) {
        Resources.loadPresentationImages();
    } else if (screenId == Resources.HI_SCORE_SCREEN) {
        Resources.loadMenuHScoresImages();
    } else {
        Resources.loadMenuImages();
    }
}
Extractive Refactoring example

- `body` does not use local variables declared in `body'`;
- there is no designator `within` or `withincode` capturing join points inside `body'`; ...
## Refactorings

<table>
<thead>
<tr>
<th>Refactoring</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Extract Method to Aspect</td>
</tr>
<tr>
<td>2</td>
<td>Extract Resource to Aspect - after</td>
</tr>
<tr>
<td>3</td>
<td>Extract Context</td>
</tr>
<tr>
<td>4</td>
<td>Extract Before Block</td>
</tr>
<tr>
<td>5</td>
<td>Extract After Block</td>
</tr>
<tr>
<td>6</td>
<td>Extract Argument Function</td>
</tr>
<tr>
<td>7</td>
<td>Change Class Hierarchy</td>
</tr>
<tr>
<td>8</td>
<td>Extract Aspect Context Commonality</td>
</tr>
</tbody>
</table>
Our Method

- Identify Variability
- Extract SPL
- Migrate SPL
- React SPL
- Refactor Feature Model
- Update Configuration Knowledge
Product line derivation and evolution benefit from refactoring
But PL refactoring should go beyond code...
and deal with populations and families too!
Feature model refactorings as improved configurability

\[
\begin{align*}
\{A,B\} & \quad \subseteq \quad \{A,B\} \\
\{A,C\} & \quad \subseteq \quad \{A,C\} \\
\ldots & \quad \subseteq \quad \ldots \\
\{A,B,C\} & \quad \subseteq \quad \{A,B,C\}
\end{align*}
\]
Refactoring catalog
Refactoring populations and families: feature models

\[ \text{fm1} \sqsubseteq \text{fm} = \text{fm1} \sqsubseteq \text{fm} \land \text{fm2} \sqsubseteq \text{fm} \]
Case Studies

- Evaluate method in industrial-strength domain
- Consider analytical and quantitative data
- Identify enhancements to method
Rain of Fire

- Extractive/Reactive
- 5KLOC
Rain of Fire

- Variability identification with concern graphs
- Platform variation fully isolated
  - Load images on demand
  - Flip feature reused
- Refactoring set was useful guide
- Coarse-grained configuration knowledge
Best Lap

- 15 KLOC
- 16 families, 65 devices
- Migrate step
- Variability identification by locating tags with tool
- Migration strategies
  - Arena feature
  - Mostly static variability
Configuration knowledge

- Varied granularity
- Non-compositional pieces
- Reflect feature dependency
  - Arena feature
Aspect reusability in Best Lap

![Bar chart showing the number of aspects in different number of instances categories: 1 with 29 aspects, [2, 9] with 20 aspects, [10, 16] with 17 aspects, and [1, 16] with 66 aspects.]

![Pie chart showing the distribution of types of aspect: 68% Product Specific Aspects, 15% Core Aspects [2, 9], 11% Core Aspects [10, 16], and 6% Non-Aspect Code.]
Analysis

- **Minor increase on SPL LOC**
  - Aspect declarations

- **But improved modularity**
  - Reusability
  - Locality
  - Adaptability
  - Pluggability
  - Independent development

- **14 % increase on Jar size**
Conclusions

- Relevance of extractive and reactive adoption strategies
- Orthogonal relevance of variability managements
- Motivation in highly variant domain
Conclusions

- Method for SPL extraction and reaction
- Refactorings for variability management
  - Guidelines and safety
  - Novel notion for SPLs
- Validation in case studies
- Domain independent method
Assumptions and limitations

- AOP does not handle all variability
  - But most variabilities and comparative analysis
- Pointcut languages still too syntactic
  - But tool support alleviates this
- Inefficient AOP implementations
  - But optimizers may mitigate this
Future work

- Traceability with tests and architecture
- Variability in other artefacts
- More case studies
- Case studies in other domains
Future work

- Definition of metric suite for SPL
- Extend tool support
- Handle additional variability
  - Import variation
  - Super class constructor call
  - Adding an else-if Block
Implementing Software Product Line Adoption Strategies

Candidate: Vander Alves
Supervisors: Paulo Borba and Geber Ramalho
March 23, 2007
Backup slides
Tool Support

- Specially critical for PLs
- Prototype
- AspectJ
- Eclipse (JDT & AJDT)
Extract Before Block

```java
public void changeScreen(int screenId) throws IOException {
    if (screenId == SPLASH_SCREEN) {
        loadSplashImages();
        new ArrayList();
    } else if (screenId == GAME_SCREEN) {
        loadGameScreenImages();
    } else if (screenId == PRESENTATION_SCREEN) {
        loadPresentationImages();
    } else if (screenId == HI_SCORE_SCREEN) {
        loadMenuHScoresImages();
    } else {
        loadMenuImages();
    }

    this.currentScreenId = screenId;
}
```
public void changeScreen(int screenId) throws IOException {

    if (screenId == SPLASH_SCREEN) {
        loadSplashImages();
        new Actions();
    } else if (screenId == GAME_SCREEN) {
        loadGame();
    } else if (screenId == MENU_SCREEN) {
        loadMenu();
    } else {
        loadMenuImages();
    }

    this.currentScreenId = screenId;
}
Extract Before Block

```java
public void changeScreen(int screenId) throws IOException {
    this.currentScreenId = screenId;
}
```
before (MainCanvas cthis, int screenId) :
    execution (public void MainCanvas.changeScreen(int))
    && this (cthis) && args (screenId) {
    if (screenId == cthis.SPLASH_SCREEN) {
        cthis.loadSplashImages();
        new ArrayList();
    } else if (screenId == cthis.GAME_SCREEN) {
        cthis.loadGameScreenImages();
    } else if (screenId == cthis.PRESENTATION_SCREEN) {
        cthis.loadPresentationImages();
    } else if (screenId == cthis.HI_SCORE_SCREEN) {
        cthis.loadMenuHScoresImages();
    } else {
        cthis.loadMenuImages();
    }
}
```java
public class teste {
    public void affe(int y) {
        int x;
        int k;
        k = 2;
        k = 3;
        int z;
        x = 5 + k;
    }
}
```