DCC / ICEx / UFMG

Evolving Software Product Lines with Aspects

Eduardo Figueiredo

http://www.dcc.ufmg.br/~figueiredo

Study Settings

- Two software product lines
  - MobileMedia
  - BestLap
- Development and evolution
  - Alignment of the AspectJ (AOP) and Java (CC) versions
  - Implementation of change scenarios (for both AOP and CC)
- Goal: assessment of design stability

Change Scenarios

MobileMedia

	Change	Feature
1	MobilePhoto core	
2	Exception handling	Mandatory
3	Label photo and count the number of photo views and sorting	Mandatory and Optional
4	Specify and view favourite photos	Optional
5	Multiple copies of photos	Optional
6	Send photo by SMS	Optional
7	Manage music	Alternative
8	Manage videos	Alternative

BestLap

	Change	Feature
1	Core features for Motorola V220	
2	Extended screen size for Motorola V300 and L6	Alternative
3	Extended sound for Nokia family	Alternative
4	Extended the shortcut keys for Siemens and Sony Ericsson	Alternative
5	Allow users to store lap times	Optional

Multi-Dimensional Analysis

- Change Impact Analysis
- Conventional Modularity Metrics
- Separation of Features
- Feature Dependency Analysis

Multi-Dimensional Analysis

- **Change Impact Analysis**
- Conventional Modularity Metrics
- Separation of Features
- Feature Dependency Analysis

Change Impact Analysis

- Metrics
  - Number of components added / removed / changed
  - Number of operations added / removed / changed
  - Number of lines of code added / removed / changed, etc.
- Classification of Changes
  - Targeting Mandatory Features
  - Targeting Optional Features
  - Targeting Alternative Features

Example of Results

- Metrics
  - Number of components added, removed, changed
  - Number of operations added, removed, changed
  - Number of lines of code added, removed, changed, etc.

Result 1
AspectJ fails when changes target mandatory features (details in the paper)

- Classification of Changes
  - **Targeting Mandatory Features**
  - Targeting Optional Features
  - Targeting Alternative Features

Result 2: Optional Features

AspectJ adds more elements

Components	R.4 (%)	R.6 (%)
Added	**+100.0**	**+12.5**
Removed	0.0	0.0
Changed	-60.0	-16.7

Classes (main behaviour) + Aspects (connection / glue)

Operations	R.4 (%)	R.6 (%)
Added	**+70.0**	**+21.3**
Removed	0.0	0.0
Changed	-85.7	-42.9

New methods (expose joinpoints) + advice

(Releases 4 and 6)

Splitting Methods

AspectJ may require the creation of methods

```
class PhotoController {
  ...
  public boolean savePhoto(...) {
     String photoName;
     ...
     // #ifdef ...
     if (imgByte.length > 0)
       addImageData(...);
     // #endif
     ...
     return true;
  }
}
```

```
class PhotoController {
  ...
  public boolean savePhoto(...) {
     String photoName;
     ...
     return proceedSave(...);
  }
  boolean proceedSave(...) {
     ...
     return true;
  }
}
```

CC

A new method to expose variability joinpoints

AOP

Result 2: Optional Features

... but, AspectJ changes less elements

Components	R.4 (%)	R.6 (%)
Added	+100.0	+12.5
Removed	0.0	0.0
Changed	**-60.0**	**-16.7**

Operations	R.4 (%)	R.6 (%)
Added	+70.0	+21.3
Removed	0.0	0.0
Changed	**-85.7**	**-42.9**

Changes are more localised into aspects

AspectJ conforms more closely the Open-Close Principle [1]

(Releases 4 and 6)

[1] Meyer, B. Object-Oriented Software Construction. st ed. Prentice-Hall, Englewood Cliffs, 1988.

Result 3: Alternative Features

AspectJ fails when turning a mandatory feature into alternative

More (classes + aspects)

Components	R.7 (%)	R.8 (%)
Added	**+19.0**	+62.5
Removed	0.0	+100.0
Changed	**+25.0**	-59.1

More (methods + advices)

Operations	R.7 (%)	R.8 (%)
Added	**+6.8**	+36.6
Removed	-11.3	0.0
Changed	**+68.1**	-56.5

Changes in joinpoints may cause core->variability ripple effects

(Release 7)

Result 3: Alternative Features

AspectJ succeeds when including a new alternative

It still adds more elements

Components	R.7 (%)	R.8 (%)
Added	+19.0	**+62.5**
Removed	0.0	+100.0
Changed	+25.0	**-59.1**

Operations	R.7 (%)	R.8 (%)
Added	+6.8	**+36.6**
Removed	-11.3	0.0
Changed	+68.1	**-56.5**

But, with fewer changes since they do not target the core

(Release 8)

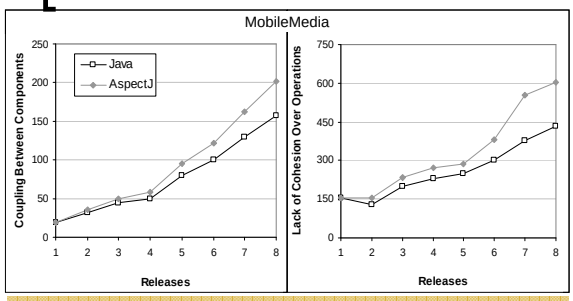
Multi-Dimensional Analysis

- Change Impact Analysis
- **Conventional Modularity Metrics**
- Separation of Features
- Feature Dependency Analysis

Modularity Analysis

- The main goal is to evaluate the stability
  - Variation in Coupling, cohesion, and size
- Traditional Modularity Metrics
  - Coupling between Components (CBC)
  - Lack of Cohesion in Operations (LCOM)
  - Number of Lines of Code (LOC)
  - Number of Components (VS), etc.

Absolute Values Favour Java

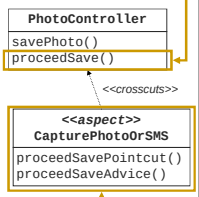


Key factor: overhead of AspectJ - heterogeneous aspects - little reuse

Higher Coupling and Lower Cohesion

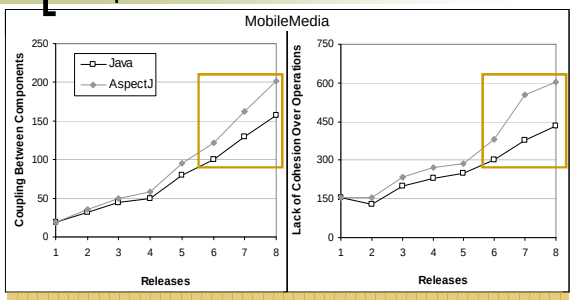
New "non-cohesive" methods to expose joinpoints

```
class PhotoController {
  ...
  public boolean savePhoto(...) {
    String photoName;
    ...
    // #ifdef includeSMS || capturePhoto
    if (imgByte.length > 0)
      addImageData(...);
    // #endif
    ...
    return true;
  }
}
```



CC Aspects coupled to the base code **AOP**

Complex Combinations of Features

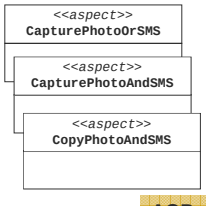


"IFDEF" uses AND / OR operators vs. (R7 and R8) **AspectJ requires several new aspects**

A New Aspect per Combination

Aspects modularise the shared code

```
...
  // #ifdef includeSMS || capturePhoto
  ...
  // #endif
...
  // #ifdef includeSMS && capturePhoto
  ...
  // #endif
...
  // #ifdef includeSMS && copyPhoto
  ...
  // #endif
...
```



CC Shared code between the SMS and Copy features **AOP**

Concluding Remarks

- Quantitative data answer
  - When the use of aspects is more adequate?
  - When the use of aspects is more challenging?
- Questions that may impact the aspectisation decision
  - What is the kind of the change?
  - Which are the target features?
  - Does the target feature share code with others?
  - Are there complex dependencies among features?

Concluding Remarks

- The use of aspects is more adequate
  - Include optional features
  - No shared code
  - No complex dependencies
- The use of aspects is more challenging
  - Turn mandatory into alternative
  - Shared code
  - Complex dependencies

Bibliography

- Figueiredo, E. et al. **Evolving Software Product Lines with Aspects: An Empirical Study on Design Stability**. Int'l Conference on Software Engineering (ICSE), 2008.