Tecnologias para Linhas de Produtos de Software

Eduardo Figueiredo
http://www.dcc.ufmg.br/~figueiredo
reuso.software@gmail.com

26 Março 2012

Agenda da Aula

- Ferramentas para variabilidade
  - Feature Modeling Plug-in (FMP)
  - XFeature
  - Pure::Variants
  - SPLOT
- Técnicas para implementação de LPS
  - Compilação condicional
  - Programação orientada a aspectos
  - Programação orientada a características

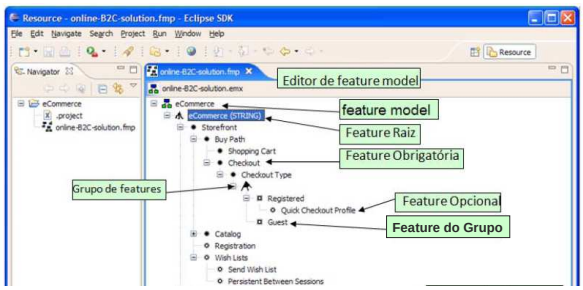
Feature Modeling Plug-in (FMP)

http://gsd.uwaterloo.ca/fmp

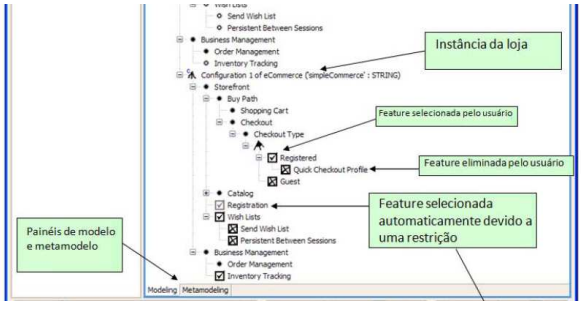
FMP

- Permite criar modelos de características para LPS
  - Plug-in para a plataforma Eclipse
  - Protótipo desenvolvido em 2005
- Criada na Universidade de Waterloo
  - Grupo do Krzysztof Czarnecki que propôs o modelo de características

Tela 1: Modelo de Características



Tela 2: Produto da LPS



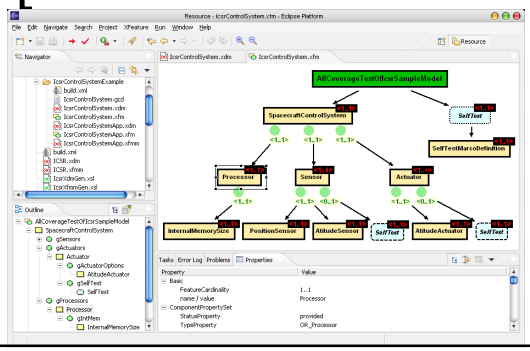
XFeature

http://www.pnp-software.com/XFeature/

XFeature

- Propósito semelhantes ao FMP
  - Permite criar modelos de características
  - Plug-in para a plataforma Eclipse
  - Protótipo desenvolvido em 2005
- Criada em dois centros de pesquisa em Zurique, Suíça
  - P&P Software GmbH
  - Automatic Control Laboratory

Tela do XFeature



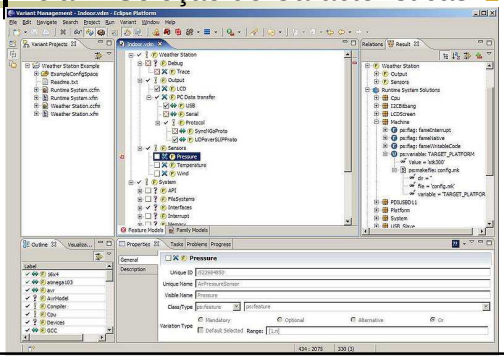
Pure::Variants

http://www.pure-systems.com/pure_variants.49.0.html

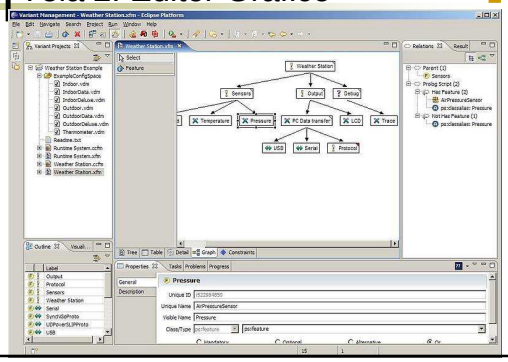
Pure::Variants

- Sistema comercial
  - Mais maduro
- Integrável ao Eclipse, mas também a outras IDEs (IBM)
  - Enterprise Architect, IBM Rational DOORS, IBM Rational Rhapsody, etc.

Tela 1: Seleção de Características



Tela 2: Editor Gráfico



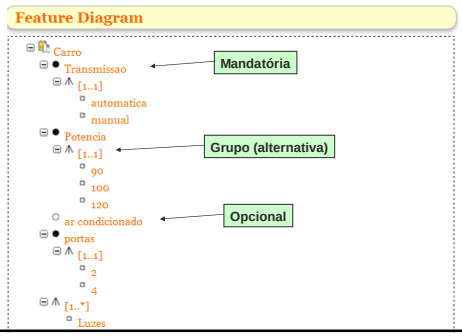
S.P.L.O.T.

http://www.splot-research.org/

Histórico do SPLOT

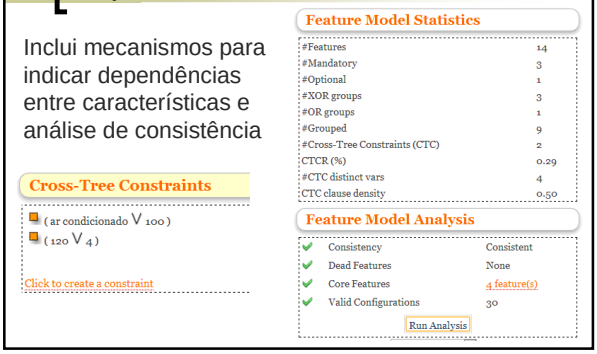
- Lançada em maio de 2009
  - Universidade de Waterloo
  - Mesmo grupo do Czarnecki (FMP)
- Ferramenta Web *open source*
- Objetivo é incentivar o compartilhamento de informação
  - Possui um repositório com cerca de 180 modelos de características

Modelo de Características



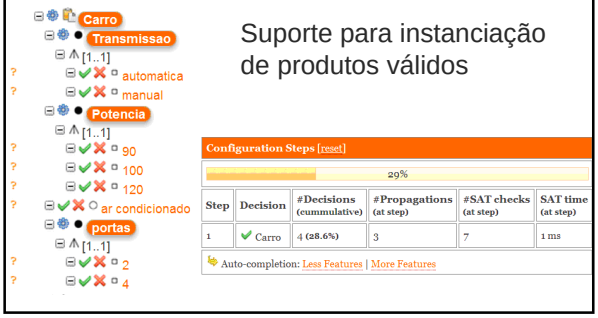
Dependências e Análise

Inclui mecanismos para indicar dependências entre características e análise de consistência



Configuração do Produto

Suporte para instanciação de produtos válidos



Compilação Condicional

Compilação Condicional

- Consiste na anotação de trechos de código associados a uma determinada característica
  - Estas anotações são interpretadas por um pré-processador que decide sobre a inclusão do código no produto
- Exemplos comuns
  - *#ifdef*, *#else*, *#endif*

Exemplo de Uso (MobileMedia)

```
public class PhotoListScreen extends List {

  public static final Command viewCommand;
  public static final Command addCommand;
  public static final Command deleteCommand;
  public static final Command editLabelCommand;
  // #ifdef includeSorting
  public static final Command sortCommand;
  // #endif
  // #ifdef includeFavourites
  public static final Command favoriteCommand;
  public static final Command viewFavoritesCommand;
  // #endif
...
}
```

O código que implementa as características opcionais delimitado por compilação condicional.

Exemplo de Uso (MobileMedia)

```
public class PhotoListScreen extends List {

  public static final Command viewCommand;
  public static final Command addCommand;
  public static final Command deleteCommand;
  public static final Command editLabelCommand;
```

S

```
  // #ifdef includeSorting
  public static final Command sortCommand;
  // #endif
```

F

```
  // #ifdef includeFavourites
  public static final Command favoriteCommand;
  public static final Command viewFavoritesCommand;
  // #endif
...
}
```

Exemplo de Uso (MobileMedia)

```
public class PhotoListScreen extends List {
  ...
  public void initMenu() {
    this.addCommand(viewCommand);
    this.addCommand(addCommand);
    this.addCommand(deleteCommand);
    this.addCommand(editLabelCommand);
    // #ifdef includeSorting
    this.addCommand(sortCommand);
    // #endif
    // #ifdef includeFavourites
    this.addCommand(favoriteCommand);
    this.addCommand(viewFavoritesCommand);
    // #endif
  }
}
```

O código que implementa as características pode estar dentro de métodos.

Exemplo de Uso (MobileMedia)

```
public class PhotoListScreen extends List {
  ...
  public void initMenu() {
    this.addCommand(viewCommand);
    this.addCommand(addCommand);
    this.addCommand(deleteCommand);
    this.addCommand(editLabelCommand);
```

S

```
    // #ifdef includeSorting
    this.addCommand(sortCommand);
    // #endif
```

F

```
    // #ifdef includeFavourites
    this.addCommand(favoriteCommand);
    this.addCommand(viewFavoritesCommand);
    // #endif
  }
}
```

Configuração

- Considerando apenas
  - Uma característica mandatória (Central)
  - Duas características opcionais (Ordenação e Favoritos)

Instâncias	Central	Ordenação	Favoritos
Produto 1	Sim	Sim	Sim
Produto 2	Sim	Sim	Não
Produto 3	Sim	Não	Sim
Produto 4	Sim	Não	Não

Como é feita a configuração

- Várias formas...
- Exemplo: é dito ao pré-processador quais características (símbolos) devem ser incluídos no produto final

S F preprocessor.symbols = central, includeSorting, includeFavourites

S preprocessor.symbols = central, includeSorting

F preprocessor.symbols = central, includeFavourites

preprocessor.symbols = central

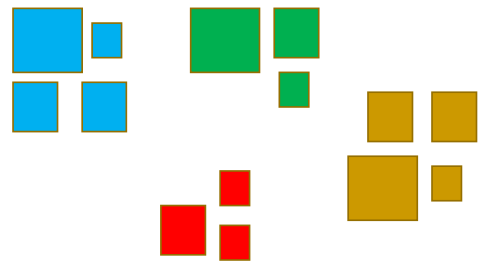
DSOA

Aspectos

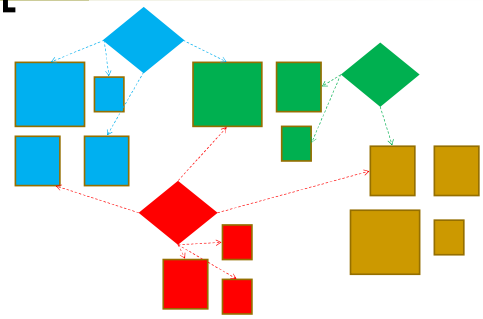
DSOA para LPS

- Aspectos podem ser usados para separar características que são transversais na LPS
  - Cada característica variável pode ser implementada por um conjunto de classes e aspectos
- A extração de características para aspectos exige identificação do código que implementa esta característica

Características Modularizadas

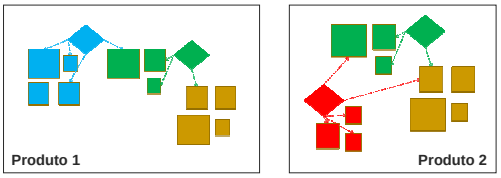


Aspectos fazem a ligação



Configuração

- Feito pela inclusão ou exclusão das classes e aspectos que implementam uma característica



Exemplo de Uso (MobileMedia)

```
public class PhotoListScreen extends List {
  public static final Command viewCommand;
  public static final Command addCommand;
  public static final Command deleteCommand;
  public static final Command editLabelCommand;
  ...
}
```

```
public aspect SortingAspect {
  public static final Command sortCommand;
  pointcut initMenu(PhotoListScreen screen):
    execution(public void PhotoListScreen.initMenu()) && this(screen);
  after(PhotoListScreen screen) : initMenu(screen) {
    screen.addCommand(sortCommand);
  }
  ...
}
```

FOP

Características

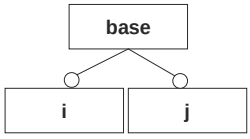
Definições

- Programação Orientada a Características (FOP) é um paradigma de programação para desenvolvimento de linha de produtos de software
- Uma característica é um incremento funcional no desenvolvimento do software

Refinamento Sucessivo

- O processo que enfatiza a simplicidade e facilidade de compreensão
- Considere o produto p1 = j • i • h
  1. O desenvolvimento começa com o característica **h**
  2. A característica **i** é adicionada refinando **h**
  3. A característica **j** é adicionada refinando **i**

Notação GenVoca

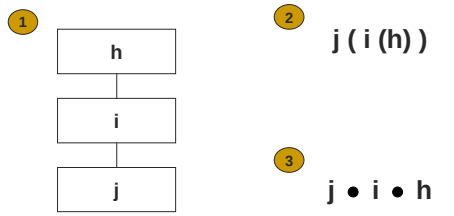


- Um produto da linha é chamado expressão
- O símbolo • denota composição

Expressões

- i : o programa base com a característica i
- j : o programa base com a característica j
- i • j : adicionar i ao programa j (acima)

Exemplos da Notação



Exemplo de Uso (MobileMedia)

```
public class PhotoListScreen extends List {
  public static final Command backCommand = new Command(…);
  …
  public void initMenu() {
    this.addCommand(backCommand);
  }
}
```

F

```
public refines class PhotoListScreen {
  public static final Command viewFavoritesCommand = new Command(…);
  public void initMenu() {
    Super().initMenu();
    this.addCommand(viewFavoritesCommand);
  }
}
```

S

```
public refines class PhotoListScreen {
  public static final Command sortCommand = new Command(…);
  public void initMenu() {
    Super().initMenu();
    this.addCommand(sortCommand);
  }
}
```

…

Estudo Comparativo

Compilação Condicional
vs. Aspectos

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

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

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

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

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)

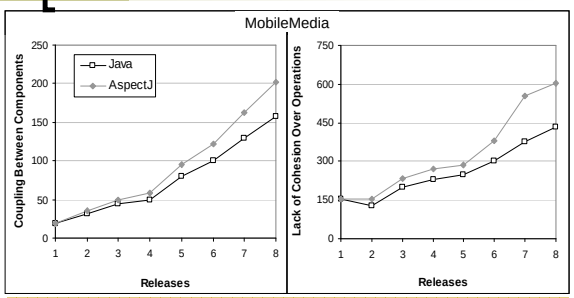
Multi-Dimensional Analysis

- 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

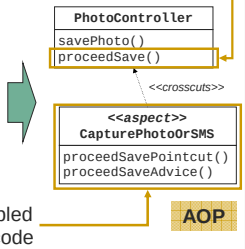


Key factor: overhead of AspectJ - heterogeneous aspects - little reuse

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;
  }
}
```

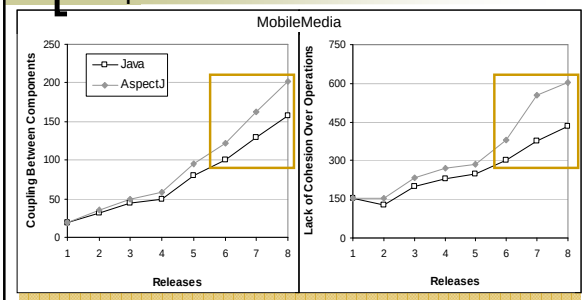


CC

Aspects coupled to the base code

AOP

Complex Combinations of Features

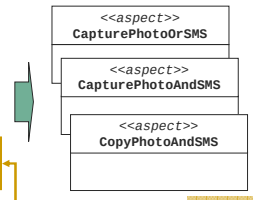


"IFDEF" uses AND / OR operators vs. **AspectJ requires several new aspects** (R7 and R8)

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

Bibliografia Principal

- Website das ferramentas
- Figueiredo, E. et al. **Evolving Software Product Lines with Aspects: An Empirical Study on Design Stability**. Int'l Conference on Software Engineering (ICSE), 2008.
- Leitura adicional
  - G. Ferreira, F. Gaia, E. Figueiredo e M. Maia. **On the Use of Feature-Oriented Programming for Evolving Software Product Lines - A Comparative Study**. Simpósio Brasileiro de Linguagens de Programação (SBLP), 2011.

Próxima Aula

- Laboratório 2011
- Assunto
  - Linha de produtos de software