On the Use of Feature-Oriented Programming for Evolving Software Product Lines – A Comparative Study

Gabriel Coutinho Sousa Ferreira, Felipe Nunes Gaia, Eduardo Figueiredo and Marcelo de Almeida Maia

{gabriel, felipegaia}@mestrado.ufu.br, figueiredo@dcc.ufmg.br, marcmaia@facom.ufu.br

Resumo

Uma análise quantitativa e qualitativa da modularidade e estabilidade arquitetural de uma SPL (Linha de produtos de software).

O artigo avalia comparativamente três mecanismo de variabilidade durante a evolução de uma linha de produto de software chamada “WebStore”.

Introdução

- **Características de Linha de produto de software**
  - Features obrigatórias e opcionais.
  - Grupo de Features opcionais = novo produto.
  - Reutilização rápida.
  - Mudanças ocorrem por natureza e devem ser acomodadas.
- **Mecanismos de variabilidade “efetivos”**
  - Devem garantir estabilidade da arquitetura.
  - Facilitar futuras mudanças.

  Obs: Evitar degradação arquitetural

Os mecanismos

O principal objetivo: Avaliar os mecanismos de composição disponíveis em Programação Orientada a Característica. As outras duas técnicas de variabilidade são usadas como linha de base para comparação.

- Compilação condicional
  - Diretivas #define #if #else
- Padrão de projetos baseado em orientação a objeto
  - Decorator, FactoryMethod
- Programação orientada a características
  - AHEAD, refinamentos

Compilação condicional

- É uma técnica conhecida
- Baseada em diretivas e pré-processadores

```
1  private ControllerAction selectPaymentMethod(...) {
2    if (paymentType.equals("Default")) {
3      paymentAction = new GoToAction("payment.jsp");
4    }

5    //#if defined(Paypal)
6    if (paymentType.equals("Paypal")) {
7      paymentAction = new GoToAction("paypal.jsp");
8    }
9    //#endif

10   return paymentAction;
11 }
```

Padrões de projeto (OO)

- Padrões de projetos orientados a objetos amplamente utilizado com o Livro “Gang of Four”.

```
1  public class ControllerMapper implements IDecorator {
2    protected Map actions = new HashMap();

3    public ControllerMapper() {
4      init();
5    }

6    public void addAction(String an, ControllerAction ca) {
7      actions.put(an, ca);
8    }

9    public void init() {
10     addAction("goToHome", new GoToAction("home.jsp"));
11   }

12   public ControllerAction getAction(String an) {
13     return actions.containsKey(an) ? actions.get(an) : null;
14   }
15 }
```

Padrões de projeto (OO)

```
1  abstract class ControllerDecorator implements IDecorator {
2    protected IDecorator mapper;
3    protected Map controllerMap = new HashMap();

4    public ControllerDecorator(IDecorator m) {
5      this.mapper = m;
6      init();
7    }

8    public abstract void init();

9    public void addAction(String an, ControllerAction ca) {
10     controllerMap.put(an, ca);
11   }

12   public ControllerAction getAction(String an) {
13     return controllerMap.containsKey(an) ?
               controllerMap.get(an) : mapper.getAction(an);
14   }
15 }
```

FOP - Programação orientada a característica (AHEAD)

```
1  public class ProcessCheckoutFormAction {
2    private ControllerAction selectPayment ...) {
3      if (paymentType.equals("Default")) {
4        paymentAction = new GoToAction("payment.jsp");
5      }
6      return paymentAction;
7    }
8  }
```

```
1  layer paypal;

2  refines class ProcessCheckoutFormAction {

3    private ControllerAction selectPayment(...) {
4      Super(ControllerAction, String).selectPayment(...);

5      if (paymentType.equals("Paypal")) {
6        paymentAction = new GoToAction("paypal.jsp");
7      }
8    }
9  }
```

Fases do estudo

1. Desenho e implementação
2. Mapeamento das características (Feature source code shadowing)
3. Calculo de métricas
4. Avalição qualitativa e quantitativa dos resultados

A evolução da WebStore SPL

- Projetada com o proposito acadêmico.
- Possui as principais funcionalidade de uma loja web real.
- Cenários de mudanças foram projetados para as três implementações do estudo.
- Cenários foram analisados durante cinco versões.

A WebStore SPL(Principais funcionalidades)

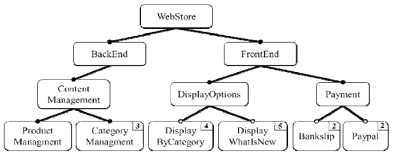
- Gerenciar produtos
- Gerenciar categoria dos produtos
- Mostrar o catalogo de produtos
- Controlar pagamentos

Algumas métricas das implementações

	CC	FOP	DP
Number of Components (avg)	25	50	45
LOC (aprox.)	1150	1250	1400

- Classes, interfaces e refinamentos foram contados como componentes
- FOP gerou mais componentes, no entanto DP gerou 20% de linhas de código a mais.

O modelo de características



Cenário de mudanças

Objetivos

Exercitar os recursos dos mecanismos de implementação e, assim, avaliar a estabilidade da arquitetura da linha de produto.

Release	Description	Type of Change
R1	WebStore core	
R2	Two types of payment included (Paypal and BankSlip)	Inclusion of optional feature
R3	New feature included to manage category	Inclusion of optional feature
R4	The management of category was changed to mandatory feature and new feature included to display products by category	Changing optional feature to mandatory and inclusion of optional feature
R5	New feature included to display products by nearest day of inclusion	Inclusion of optional feature

Analise das mudanças

Resumo

Como as mudanças realizadas com diferentes mecanismos de variabilidade afetam a evolução da WebStore.

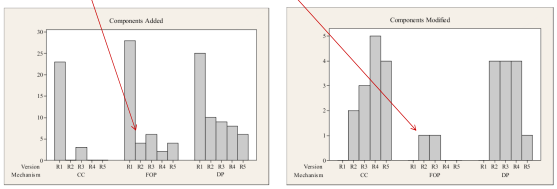
Métricas utilizadas são baseadas em:

- componentes(Classes, Interfaces, Refinamentos)
- métodos
- linhas de código

Obs.: O mecanismo de variabilidade é considerado mais estável, o quanto menor for o impacto das mudanças.

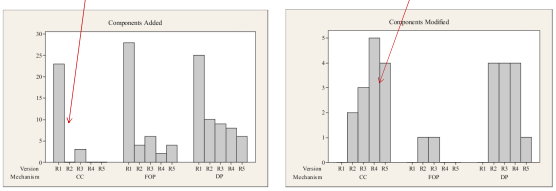
Analise das mudanças

- FOP requer menos modificações.
- Desesseis adicionados e dois componentes modificados.
- Aderente ao principio “software should be open for extension, but closed for modification”.



Analise das mudanças

- CC adiciona poucos componentes, mas gera muitas mudanças.
- Não aderente ao principio “software should be open for extension, but closed for modification”.



Analise das mudanças

- A implementação baseada em “design patters/OO” gera os piores resultados.

			Releases			
			R.2	R.3	R.4	R.5
Components	Added	CC	0	3	0	0
		FOP	4	6	2	4
		DP	10	9	8	6
	Removed	CC	0	0	0	0
		FOP	0	0	0	0
		DP	0	0	11	0
	Changed	CC	2	3	5	4
		FOP	1	1	0	0
		DP	4	4	4	1
Lines of Code	Added	CC	15	148	7	14
		FOP	35	160	14	28
		DP	132	181	179	59
	Removed	CC	0	3	0	0
		FOP	0	3	0	0
		DP	0	1	192	0
	Changed	CC	1	2	0	0
		FOP	1	1	1	1
		DP	9	2	3	0

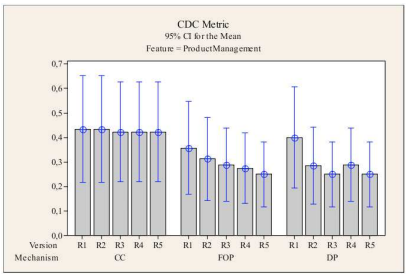
Analise da modularidade

Obs.: Concern = feature

- Um conjunto de métricas foi utilizada para medir a modularidade durante a evolução da SPL WebStore.
  - **CDC -** Concern Diffusion over Components(Classes, refinamentos)
    - scattering
  - **CDO -** Concern Diffusion over Operations(Métodos)
    - scattering
  - **NOCA -** Number Of Concern Attributes (Atributos)
    - scattering.
  - **CDLOC -** Concern Diffusion over Lines of Code(Linhas)
    - tangling
  - **LOCC -** Number Of Lines of Concern Code(Linhas)

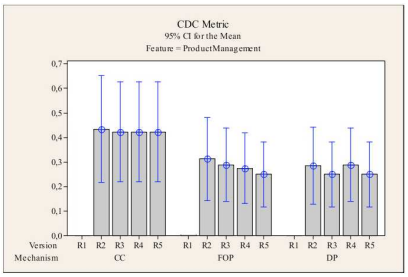
Analise da modularidade(CDC)

- Valores mais baixos representam melhores resultados.



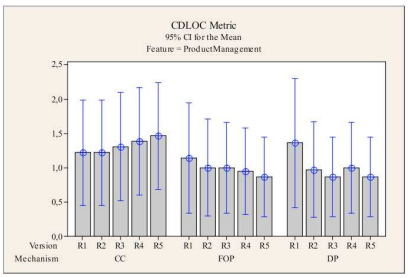
Analise da modularidade(CDC)

- Sem R1



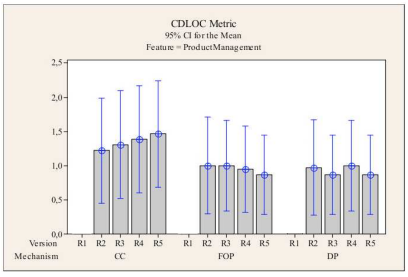
Analise da modularidade(CDLOC)

- Valores mais baixos representam melhores resultados.



Analise da modularidade(CDLOC)

- Valores mais baixos representam melhores resultados.



Trabalhos relacionados

- Aspect
- CaesarJ
- Aspectual Mixin Laye

Restrições

- Devido à natureza exploratória do estudo, há algumas restrições e os resultados podem não se repetir em outros contextos mais complexos.
- Não forma incorporados todos os tipos de mudanças durante a evolução da SPL.
- O uso de AHEAD também poderia ser apontada como uma restrição, uma vez que não é a única existente.

Conclusão

- FOP tende a ser mais estavel que as outras soluções
- FOP adere ao principio “Fechado para altração e aberto para extensão”.
- CC não é adequado para implenteção de LPS quando o foco é a madularidade das caracteristicas.
- DP é mais rigido para acomadar mudanças
- DP requer mais componentes e mudanças

Meus questionamentos

- Dúvidas sobre o modelo de implementação em DP/OO. Não tem pré-compilação
- As métricas favorecem estruturas modulares
- Apenas métricas relacionadas a acoplamento e coesão foram considerados.
- Outros aspectos importantes a se considerar em arquiteturas evolutivas:
  - Produtividade
  - Manutenibilidade
  - Compressibilidade
  - Distribuição