The Impact of Static-Dynamic Coupling on Remodularization

Rick Chern
Kris De Volder

Vitor Madureira Sales

OOPSLA - October, 2008– **Nashville**, **EUA**

Introdução

Introdução

- Foi considerado dois tipos de estrutura do programa:
  - Estática
  - Dinâmica

- Acoplamento estático-dinâmico:
  - Alguns projetos de linguagens tornam mais difícil mover código de estrutura estática sem modificar a estrutura dinâmica do programa.

- *Como e quanto o acoplamento estático-dinâmico em uma linguagem impacta na complexidade da remodularização do programa?*

- *Foi feito uma série de remodularizações usando Java e SubjectJ.*

Acoplamento estático-dinâmico

Acoplamento estático-dinâmico

```
1  public class Counter extends JPanel {
2      private int count = 0;
3      private JButton button;
4      private JLabel label;
5
6      public Counter() {
7          label = new JLabel("" + getCount());
8          add(label);
9          button = new JButton("Increment");
10         add(button);
11         button.addActionListener(new ActionListener() {
12             public void actionPerformed(ActionEvent e) {
13                 increment();
14             }
15         });
16     }
17
18     public int getCount() {
19         return count;
20     }
21
22     public void increment() {
23         count++;
24         updateDisplay();
25     }
26
27     private void updateDisplay() {
28         label.setText("" + getCount());
29     }
30 }
```

Acoplamento estático-dinâmico

```
1  public interface CounterListener {
2      public void valueChanged();
3  }
4
5  public class CounterModel extends JPanel {
6      List<CounterListener> listeners = new ArrayList<CounterListener>();
7      private int count = 0;
8
9      public CounterModel() {
10     }
11
12     public void increment() {
13         count++;
14         notifyListeners();
15     }
16
17     private void notifyListeners() {
18         for (CounterListener l : listeners)
19             l.valueChanged();
20     }
21
22     public void addListener(CounterListener l) {
23         listeners.add(l);
24     }
25
26     public int getCount() {
27         return count;
28     }
29 }
```

Acoplamento estático-dinâmico

```
1  public class Counter extends JPanel implements CounterListener {
2      private CounterModel count;
3      private JButton button;
4      private JLabel label;
5
6      public Counter() {
7          count = new CounterModel();
8          label = new JLabel("" + count.getCount());
9          add(label);
10         button = new JButton("Increment");
11         add(button);
12         button.addActionListener(new ActionListener() {
13             public void actionPerformed(ActionEvent e) {
14                 count.increment();
15             }
16         });
17         count.addListener(this);
18     }
19
20     public void valueChanged() {
21         label.setText("" + count.getCount());
22     }
23 }
```

Acoplamento estático-dinâmico

- Java possui alto grau de acoplamento estático-dinâmico.
- Suporte difícil à ferramentas de refatoração. (Cut-and-Paste)
- Intenção original era separar o código, mas complexidade foi inserida.
- A implementação de listener traz complexidade desnecessária.
- Java impõe limitação estrutural na qual a remodularização deve ser feita mudando a estrutura dinâmica do programa.

SubjectJ

SubjectJ

- Projetada para ser muito parecida com Java, porém com menor acoplamento estático-dinâmico.
  - Programas Java devem ser programas SubjectJ válidos.
  - Suporte a IDE semelhante.
- SubjectJ decompõe Programas Java em unidades modulares chamadas Sujeitos.
- A sintaxe do Sujeito é essencialmente a sintaxe de um programa Java com anotações.
- Anotações deixam explicitas quais sujeitos dependem de outros sujeitos.
- Sujeitos devem ser *declaratively complete*.

SubjectJ

Annotation	Attached to	Meaning
`@Export`	Field, Method, Constructor	The signature of this declaration can be imported by other subjects.
`@Import`	Field, Method, Constructor	The signature of this declaration must be exported by another subject. This annotation should only be attached to method declarations without a body, and field declarations without an initializer.
`@Shared`	Class	The class header (including "extends" clause, but not necessarily all "implements" clauses) can be shared with other subjects. "Implements" clauses are shared automatically if the corresponding interface is shared.
`@Shared`	Interface	The interface header (but not necessarily all "extends" clauses) can be shared with other subjects. "Extends" clauses are shared automatically if the corresponding interface is shared.

Table 1. Annotations defining a subject's interface

SubjectJ

```
1  public class Counter extends JPanel {
2      private int count = 0;
3      private JButton button;
4      private JLabel label;
5
6      public Counter() {
7          label = new JLabel("" + getCount());
8          add(label);
9          button = new JButton("Increment");
10         add(button);
11         button.addActionListener(new ActionListener() {
12             public void actionPerformed(ActionEvent e) {
13                 increment();
14             }
15         });
16     }
17
18     public int getCount() {
19         return count;
20     }
21
22     public void increment() {
23         count++;
24         updateDisplay();
25     }
26
27     private void updateDisplay() {
28         label.setText("" + getCount());
29     }
30 }
```

SubjectJ

```
1   @Shared
2   public class Counter extends JPanel {
3       private JButton button;
4       private JLabel label;
5
6       public Counter() {
7           label = new JLabel("" + getCount());
8           add(label);
9           button = new JButton("Increment");
10          add(button);
11          button.addActionListener(new ActionListener() {
12              public void actionPerformed(ActionEvent e) {
13                  increment();
14              }
15          });
16      }
17
18      @Import
19      public int getCount();
20
21      @Import
22      public void increment();
23
24      @Export
25      private void updateDisplay() {
26          label.setText("" + getCount());
27      }
28  }
```

SubjectJ

```
1   @Shared
2   public class Counter extends JPanel {
3       private int count = 0;
4
5       @Export
6       public void increment() {
7           count++;
8           updateDisplay();
9       }
10
11      @Export
12      public int getCount() {
13          return value;
14      }
15
16      @Import
17      private void updateDisplay();
18  }
```

SubjectJ Tools

- SubjectJ possui um conjunto de ferramentas com o intuito de editar, compilar e executar programas SubjectJ refatorados.
  - Compose
  - Decompose
  - Checker

SubjectJ

Attached to	Meaning
`@Subject`($S_1, \ldots$)	
Class	The class header (including "extends" clause but not necessarily "implements" clauses) belongs to subjects $S_1, \ldots$
Interface	The interface header (not necessarily including "extends clauses") belongs to subjects $S_1, \ldots$
`@Subject`(S)	
Field	The field (signature plus optional initialer) belongs to subject S
Method, Constructor	The signature and optional body belong to subject S
`@Export`(S_1, ...)	
Field, Method, Constructor	The declaration's signature is imported by subjects $S_1, \ldots$ and is exported by the subjects the declaration belongs to.
`@Implement( @Mapping`(key=S_1, values = $I_1^1, I_1^2, \ldots$),...)	
Class	For each subject S_i, only the "implements" clauses for interfaces $I_i^1, I_i^2, \ldots$ belong to S_i. For subjects not explicitly mapped, implicitly include all "implements" clauses.
Interface	Like for a class, but applies to "extends" instead of "implements" clauses.

Table 2. "Tracking" annotations that allow decomposing Java programs into subjects.

SubjectJ

```
1  @Subject({ "UI", "OTHER" })
2  public class Counter extends JPanel {
3      private int count = 0;
4      @Subject("UI")  private JButton button;
5      @Subject("UI")  private JLabel label;
6
7      @Subject("UI")  public Counter() {
8          label = new JLabel("" + getCount());
9          add(label);
10         button = new JButton("Increment");
11         add(button);
12         button.addActionListener(new ActionListener() {
13             public void actionPerformed(ActionEvent e) {
14                 increment();
15             }
16         });
17     }
18
19     @Export("UI")   private int getCount() {
20         return count;
21     }
22
23     @Export("UI")   public void increment() {
24         count++;
25         updateDisplay();
26     }
27
28     @Subject("UI")  @Export("OTHER")
29     private void updateDisplay() {
30         label.setText("" + getCount());
31     }
32 }
```

SubjectJ

```
1   @Shared
2   public class Counter extends JPanel {
3       private JButton button;
4       private JLabel label;
5
6       public Counter() {
7           label = new JLabel("" + getCount());
8           add(label);
9           button = new JButton("Increment");
10          add(button);
11          button.addActionListener(new ActionListener() {
12              public void actionPerformed(ActionEvent e) {
13                  increment();
14              }
15          });
16      }
17
18      @Import
19      public int getCount();
20
21      @Import
22      public void increment();
23
24      @Export
25      private void updateDisplay() {
26          label.setText("" + getCount());
27      }
28  }
```

SubjectJ

```
1   @Shared
2   public class Counter extends JPanel {
3       private int count = 0;
4
5       @Export
6       public void increment() {
7           count++;
8           updateDisplay();
9       }
10
11      @Export
12      public int getCount() {
13          return value;
14      }
15
16      @Import
17      private void updateDisplay();
18  }
```

Experimento

Experimento

Application and Description†	LOC	Code to Separate
Tetris—game of Tetris `http://cslibrary.stanford.edu/112/`	1036	GUI handling.
TyRuBa—logic programming language `http://tyruba.sourceforge.net/`	22116	Storing and persisting "facts" used by the language.
JHotDraw—simple drawing application `http://www.jhotdraw.org/`	14611	All functionality for creating and modifying text figures.
Chinese Chess—game of Chinese chess `https://chinese-chess-xiang-qi.dev.java.net/`	3073	Logic for AI opponent.
MineRay—game of minesweeper `https://mineray.dev.java.net/`	3478	Logic for populating map with mines.
DrawSWF—simple animation application `http://drawswf.sourceforge.net/`	7540	All functionality for creating and modifying text figures.
FindBugs—Java source code bug finder `http://findbugs.sourceforge.net/`	70833	Saving of bug analysis results.
JChessBoard—game of chess `http://jchessboard.sourceforge.net/`	6190	GUI handling.

† All website references verified February 2008.

Table 3. Overview of selected code bases and corresponding refactoring tasks for case studies. LOC is the total number of non-blank and non-comment lines in the code base.

Ameaças à validade

Ameaças à validade

- Variáveis do experimento são linguagens enquanto se deseja entender o acoplamento estático-dinâmico.
- SubjectJ diferir além do acoplamento estático-dinâmico.
- Ferramentas de suporte a refatoração com suporte distintos para as linguagens.
- Bugs, tempo de refatoração podem não ser um indicadores de complexidade.
- Generalização dos resultados pode ser inviável.

Ameaças à validade

- Ganho de conhecimento pelo programador na remodularização.
- Tipos de remodularização feita nos oitos sistemas.
- Remodularização Java first têm mais linhas.
- Remodularizações de outros tipos precisam de mais investigações.
- Programador é o primeiro autor do artigo e criador de SubjectJ.

Resultados

Resultados

Code Base	SubjectJ *(hours)*	Java *(hours)*	Difference
SubjectJ remodularization performed first (total hours = 63.4)			
Tetris	3.0	3.5	+17%
TyRuBa	18.0	20.3	+13%
JHotDraw	4.2	4.0	-5%
Chinese Chess	3.8	6.6	+74%
Sum(1)	29.0	34.4	+19%
Java remodularization performed first (total hours = 66.9)			
MineRay	0.7	2.3	+252%
DrawSWF	2.0	5.8	+183%
FindBugs	9.9	30.7	+210%
JChessBoard	3.0	12.5	+325%
Sum(2)	15.6	51.3	+229%
Aggregated results (total hours = 130.3)			
Sum(1)+Sum(2)	**44.6**	**85.7**	**+92%**

Table 4. Overview of time data results from case studies.

Introdução de Bugs

Bugs

#	Code Base	Behavior	Cause
Bugs introduced during Java remodularization tasks			
1	JHotDraw	NullPointerException	Dual is referenced before it is created.
2	Chinese Chess	NullPointerException	Reference to dual is not initialized.
3	FindBugs	NullPointerException	"Getter" method returns `null` instead of reference.
4	FindBugs	Program freeze	"Getter" method calls itself infinitely.
5	JChessBoard	NullPointerException	Dual of `JChessBoard` used before it is fully initialized.
6	JChessBoard	NullPointerException	Dual of `History` used before it is fully initialized.
7	DrawSWF	NullPointerException	Manually extracted code from method erroneously sets local variable to `null`.
8	DrawSWF	IllegalArgumentException	Reference to original class instead of new subclass.
Bugs introduced during SubjectJ remodularization tasks			
9	DrawSWF	NullPointerException	Manually extracted code from method erroneously returns `null`.

Table 5. Overview of bugs introduced during case studies (arbitrarily numbered for reference convenience).

Estratégias de Separação

Estratégias de Separação

- SubjectJ
  - Mudanças na estrutura dinâmica foram requeridas quando possuía código dentro de métodos de interesse distinto do próprio método.

- Java
  - MVC.
  - Método estático.
  - Split Class em superclasse e subclasse.
  - Estratégia Dual.

Estratégias de Separação

```
1  public class BugInstance {
2      private BugProperty propertyListHead;
3      private SAVE_BugInstance saveBugInstance;
4      ...
5      public BugInstance(...) {
6      ...
7          saveBugInstance = new SAVE_BugInstance(this);
8      ...
9      }
10     public BugProperty getPropertyListHead() {
11         return propertyListHead;
12     }
13 }
14
15 public class SAVE_BugInstance {
16     private BugInstance bugInstance;
17
18     public SAVE_BugInstance(BugInstance bugInstance) {
19         this.bugInstance = bugInstance;
20     }
21
22     public void writeXML() {
23         ...
24         BugProperty prop = bugInstance.getPropertyListHead();
25         ...
26     }
27     ...
28 }
```

Suporte à Refatoração

- Nível de suporte que pode ser alcançado por uma ferramenta de refatoração é indiretamente afetado pelo nível de acoplamento estático-dinâmico.
- Refatorações podem requerer transformações complexas, difíceis de implementar.
- Eclipse IDE não suporta satisfatoriamente refatorações.
- Ferramenta de refatoração são complementares a linguagens com menor acoplamento estático-dinâmico.

Conclusão

Conclusão

- *Como e quanto o acoplamento estático-dinâmico em uma linguagem impacta na complexidade da remodularização do programa?*
- Fornece resultados quantitativos para responder a questão.
- Sugerem que devido ao acoplamento estático-dinâmico a remodularização Java demora mais tempo e insere mais bugs.
- Alto acoplamento estático-dinâmico em Java traz complexidade acidental.
- Estratégias de refatoração necessárias para remodularizar códigos Java não são bem suportadas por ferramentas.

Thanks

vitormsales@dcc.ufmg.br

Bugs

```
1   @Subject("Text")
2   @Export("OTHER")
3   private static DrawObject createObject2_TEXT(int drawing_mode) {
4
5       if (drawing_mode == TEXT) {
6           new Text();
7       }
8
9       return null;
10  }
```