

Compreensão, Manutenção e Evolução de Software

Cap. 4 - Design Flexível a Mudanças

Prof. Marco Tulio Valente

Licença CC-BY; permite copiar, distribuir, adaptar etc; porém, **créditos devem ser dados ao autor dos slides**

Se você quer facilitar mudanças e evoluções em um sistema, você deve pensar no design que vai adotar

Design Flexível \Rightarrow evolução mais fácil

Definição de Design de Software

- Design: escolher as "partes" que vão formar um sistema
- Em ES, essas partes são chamadas de módulos
- Ou então: classes, componentes, pacotes, subsistemas, ou, mesmo, funções

Módulos = interface + implementação

- **Interface**: métodos que o módulo oferece para o resto do sistema; ou seja, são públicos
- **Implementação** dos métodos da interface; são privados
- Todo acesso ao módulo ocorre via sua interface
- Essa propriedade é chamada de **information hiding** ou encapsulamento

Origem do conceito (David Parnas, 1972)

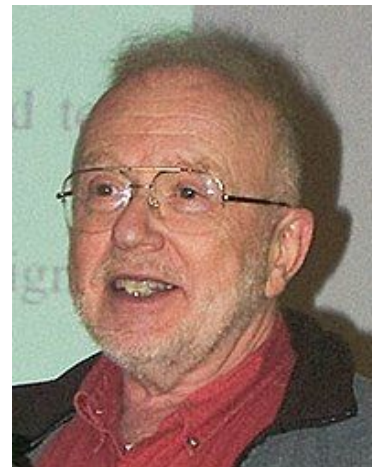
On the Criteria To Be Used in Decomposing Systems into Modules

D.L. Parnas
Carnegie-Mellon University

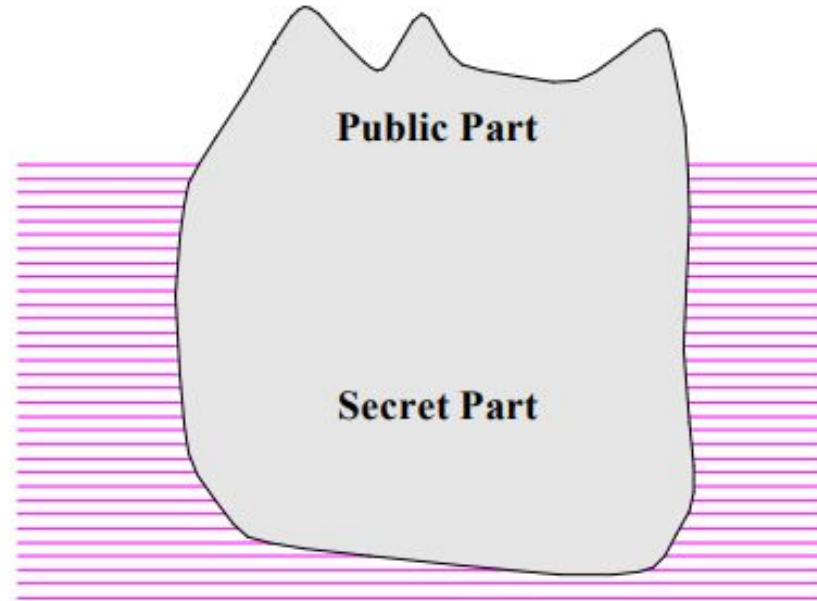
This paper discusses modularization as a mechanism for improving the flexibility and comprehensibility of a system while allowing the shortening of its development time. The effectiveness of a “modularization” is dependent upon the criteria used in dividing the system into modules. A system design problem is presented and

Introduction

A lucid statement of the philosophy of modular programming can be found in a 1970 textbook on the design of system programs by Gouthier and Pont [1, ¶10.23], which we quote below:¹



Information Hiding em 1 slide



Fonte: Bertrand Meyer, Object-oriented software construction, 1997 (pág. 51)

Qual a vantagem de Information Hiding?

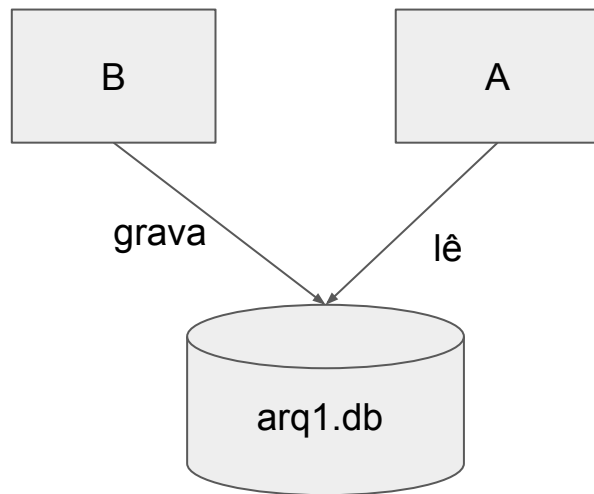
Por que essa propriedade facilita mudanças
em um software?

Information hiding facilita mudanças internas!

- Devs do módulo podem mudar sua implementação, até de forma radical, sem afetar os clientes
- Por exemplo, podem realizar manutenções corretivas, preventivas e refatorações
- Supondo que a interface continua valendo

Como violar "information hiding"?

Anti-padrão: acoplamento por dados (ou por estruturas de dados globais)



Por que acoplamento por dados é ruim?

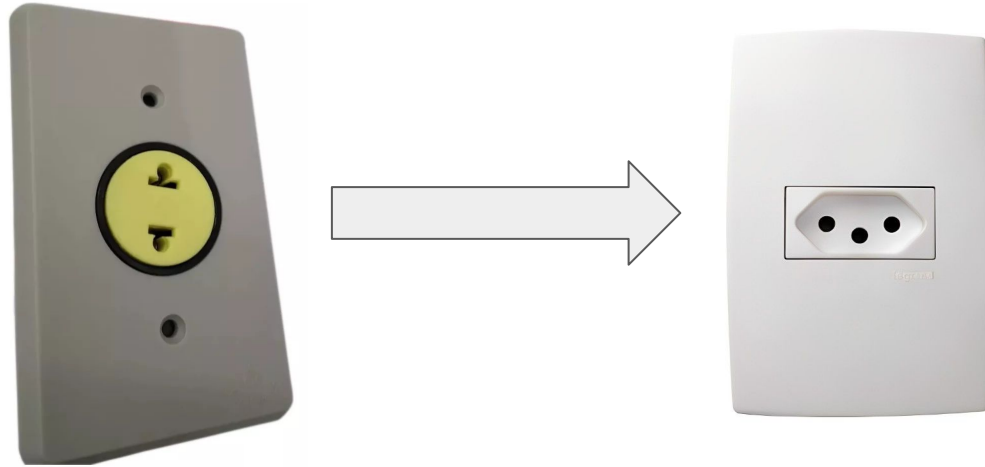
Por que ele pode dificultar a evolução de um sistema?

Breaking Changes

Breaking Changes

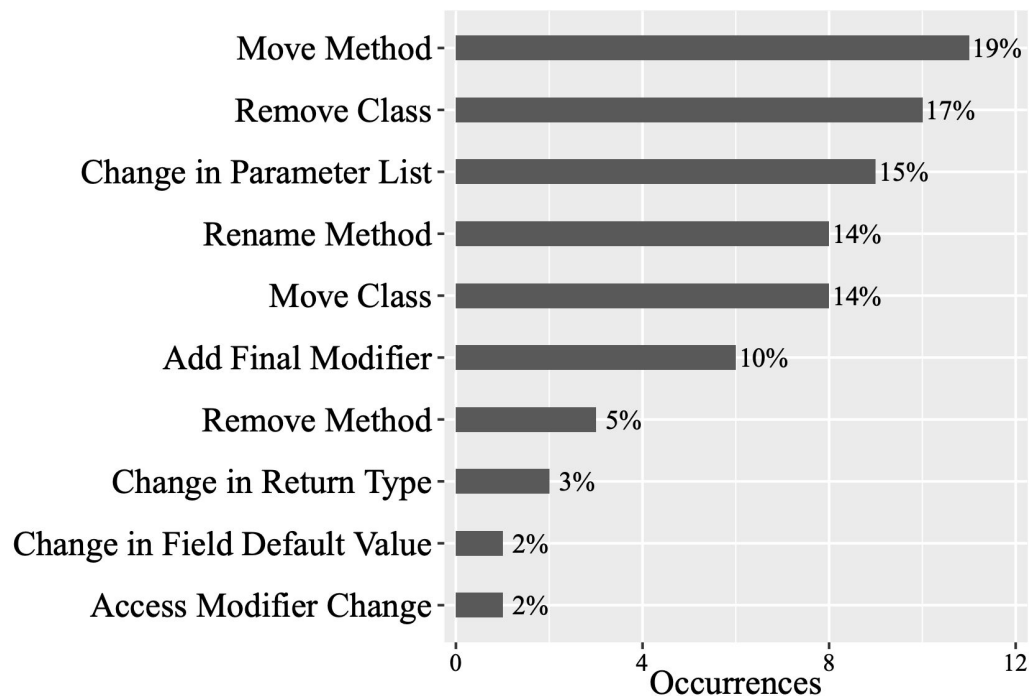
- Mudança na interface de um módulo:
 - Renomear um método
 - Remover um método
 - Adicionar ou remover um parâmetro
 - Mudança nos tipos dos parâmetros
 - Mudança no tipo de retorno
 - etc

Breaking Changes em 1 slide



Uma "tomada" funciona como uma interface para um sistema elétrico complexo, que inclui distribuição, transmissão, várias formas de geração de energia, etc.

Breaking Changes mais comuns em Java



🔄 You Retweeted



Gene Kim @RealGeneKim · Jul 5



An astonishing paper that may explain why it's so difficult to patch.

They monitored 400 libraries. In 116 days, they saw 282 breaking changes!

Each day, there's 6.1% chance of breaking chg, for each lib you use!

@topopal @mtnygard @mik_kersten @ctxt

arxiv.org/pdf/1801.05198...

By following a firehouse interview method [10], we monitored 400 real world Java libraries and frameworks hosted on GitHub during 116 days. During this period, we detected 282 possible breaking changes, sent 102 emails, and received 56 responses, which represents a response rate of 55%. With the study, we provide the following contributions: (1) to the best of our knowledge, this is the first large-scale field study that reveals the reasons of concrete breaking changes introduced by practitioners in the source code of popular Java APIs; (2) we show how breaking changes are introduced in the source code, including the most common program transformations used to break APIs; (3) we provide an extensive list of implications of our study, including implications to language designers, tool builders, software engineering researchers, and practitioners.

💬 31

🔄 428

❤️ 647



Motivações para Breaking Changes

TABLE III
WHY DO WE BREAK APIs?

Motivation	Description	Occur.
NEW FEATURE	BCs to implement new features	19
API SIMPLIFICATION	BCs to simplify and reduce the API complexity and number of elements	17
MAINTAINABILITY	BCs to improve the maintainability and the structure of the code	14
BUG FIXING	BCs to fix bugs in the code	3
OTHER	BCs not fitting the previous cases	6

Motivações para Breaking Changes: New Feature

- "The changes were just a setup before implementing a new feature: chart data retrieval."
- "The changes were adding new functionality [...] but to avoid unnecessary duplications I had to change the method name to better reflect what the method would be doing after the changes."

Motivações para Breaking Changes: API Simplification

- "We can access the argument without it being provided using another technique."
- "This method should not accept any parameters, because they are ignored by server."

Motivações para Breaking Changes: Maintainability

- "Make support class lighter, by moving methods to Class and Method info [utility classes]"

Breaking Changes Comportamentais

Breaking Changes Comportamentais

- Mudança no comportamento de um método da interface
- Na maioria das vezes, sem mudança na assinatura
- Exemplo: um método retornava um resultado em milhas e agora retorna em quilômetros

Mais um exemplo: Argumentos Default

```
def sum(a=0, b=0):  
    return a + b
```

```
sum(10, 20)
```

```
30
```

```
sum(10)
```

```
10
```

```
sum()
```

```
0
```

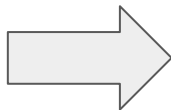
Default Arguments Breaking Changes (DABC)

```
def sum(a=0, b=0):  
    return a + b
```

```
sum(10, 20)  
30
```

```
sum(10)  
10
```

```
sum()  
0
```



```
def sum(a=0, b=1):  
    return a + b
```

```
sum(10, 20)  
30
```

```
sum(10)  
11
```

```
sum()  
1
```

Sem erro de compilação ou de tipos, mas
comportamento (resultado) do programa mudou

Quão frequentes são DABCs?

- Estudo com várias releases do scikit-learn (biblioteca de machine learning para Python)

TABLE II
DABCs FOR EACH VERSION.

Version	DABCs	
	#	%
0.19	3	3.9
0.20	11	14.3
0.21	3	3.9
0.22	43	55.8
0.23	3	3.9
0.24	2	2.6
1.0	1	1.3
1.1	11	14.3
Total	77	100

Boas práticas para tratar breaking changes

1. Anotação deprecated (na prática, adia-se a BC)

```
public class Example {  
  
    @Deprecated  
    public void oldMethod() {  
        System.out.println("This method is deprecated.");  
    }  
  
    public void newMethod() {  
        System.out.println("This is the new method.");  
    }  
  
    public static void main(String[] args) {  
        Example example = new Example();  
  
        example.oldMethod();           // you'll see a warning  
  
        example.newMethod();  
    }  
}
```

2. Versionamento Semântico: MAJOR.MINOR.PATCH

- **PATCH:** incrementado quando existem apenas correções de bugs. Exemplo: 1.5.1 \Rightarrow 1.5.2
- **MINOR:** incrementado quando existem novas features, mas sem breaking changes. Exemplo: 1.5.2 \Rightarrow 1.6.0
- **MAJOR:** incrementado quando existem novas features, com breaking changes. Exemplo: 1.6.0 \Rightarrow 2.0.0

3. Documentar

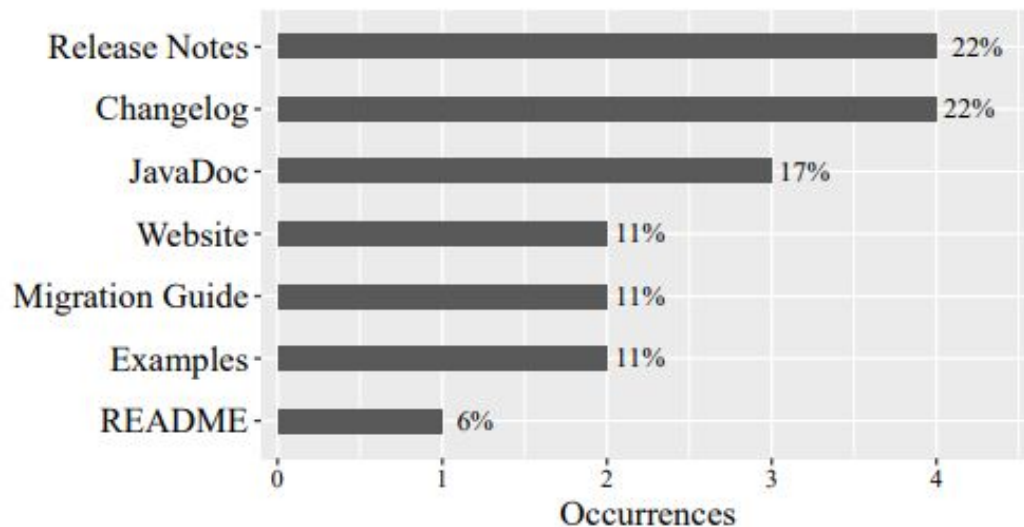


Fig. 12. How do you plan to document the detected BCs?

API Internas

API: Externa vs Interna

- API Externa: funções públicas que um módulo exporta para seus clientes externos
- API Interna: funções públicas que um módulo implementa para consumo interno
- Boa prática: deixar claro no nome do método que ele implementa uma API interna. Exemplo:
`org.eclipse.jdt.internal.*`

Problema: nada impede que sejam
usadas externamente!

Um exemplo real

- 9.702 projetos clientes dependem do Eclipse
- 2.277 clientes (23,5%) dependem de interfaces internas do Eclipse

Andre Hora, Marco Tulio Valente, Romain Robbes, Nicolas Anquetil. When Should Internal Interfaces be Promoted to Public? In *24th International Symposium on the Foundations of Software Engineering (FSE)*, 2016.

"Solução": Promoção para API Externa

promoted BaseJavaElementContentProvider to API under the name of StandardJavaElementContentProvider

master v200711131101_perf33x ... Git_migration

Erich Gamma authored on May 17, 2002 1 parent 739d172 commit

4 org.eclipse.jdt.ui/ui/org/eclipse/jdt/internal/ui/browsing/JavaBrowsingContentProvider.java

	@@ -43,10 +43,10 @@
43	43
44	44 import org.eclipse.jdt.internal.ui.javaeditor.EditorUtility;
45	45 import org.eclipse.jdt.internal.ui.preferences.JavaBasePreferencePage;
46	-import org.eclipse.jdt.internal.ui.viewsupport.BaseJavaElementContentProvider;
46	+import org.eclipse.jdt.ui.StandardJavaElementContentProvider;
47	47
48	48
49	-class JavaBrowsingContentProvider extends BaseJavaElementContentProvider implements IElen
49	+class JavaBrowsingContentProvider extends StandardJavaElementContentProvider implements I
50	50
51	51 private StructuredViewer fViewer;
52	52 private Object fInput;

Design for Change

Módulos Configuráveis

- Módulo oferece um serviço, via sua interface
- Quase sempre o cliente precisa configurar esse serviço
- Configurar \Rightarrow parametrizar, adaptar, customizar, estender
- Princípio Aberto/Fechado: módulo deve estar fechado para edições, mas aberto para extensões
- Como conseguir isso?

Como tornar um módulo configurável?

- Função de mais alta ordem (lambda function)
- Padrões de projeto, por exemplo Template Method
- Injeção de dependência

Função de mais alta ordem (lambda function)

```
List<Person> people = new ArrayList<>();  
people.add(new Person("Alice", 30));  
people.add(new Person("Bob", 25));  
people.add(new Person("Charlie", 35));  
  
Collections.sort(people, (p1, p2) -> Integer.compare(p1.age, p2.age));  
  
people.forEach(System.out::println);
```

Template Method

```
abstract class Funcionario {  
  
    double salario;  
  
    ...  
    abstract double calcDescontosPrevidencia();  
    abstract double calcDescontosPlanoSaude();  
    abstract double calcOutrosDescontos();  
  
    public double calcSalarioLiquido() { // template  
method  
        double prev = calcDescontosPrevidencia();  
        double saude = calcDescontosPlanoSaude();  
        double outros = calcOutrosDescontos();  
        return salario - prev - saude - outros;  
    }  
}
```

Podem mudar, de acordo com o tipo de Funcionario

Injeção de Dependências

Injeção de Dependências

- Padrão de projeto (porém não é um padrão GoF)
- Usado para desacoplar uma classe de suas dependências mais importantes e sujeitas a mudanças

Exemplo

```
class A {  
    private ServicoPagto pagto;    // A depende de ServicoPagto  
    ...  
}
```

Solução que **não** usa injeção de dependência

```
class A {  
    private ServicoPagto pagto = new PagtoPayPal();  
    ...  
}
```



Qual o problema?

Solução que **não** usa injeção de dependência

```
class A {  
    private ServicoPagto pagto = new PagtoPayPal();  
    ...  
}
```

Dependência está “hard-coded”

Solução que usa injeção de dependência (via um construtor)

```
class A {  
    private ServicoPagto pagto;  
  
    A(ServicoPagto pagto) { // injeção de dependência  
        this.pagto = pagto;  
    }  
    ...  
}
```

Solução que usa injeção de dependência (via um método set)

```
class A {  
    private ServicoPagto pagto;  
    ...  
    void setServicoPagto(ServicoPagto pagto)  
        this.pagto = pagto;  
}  
...
```

Injeção de Dependência: Resumo

- Injetar dependências: parametrizar uma classe de forma que seja possível alterar suas dependências.
- Objetivo: permitir mudar as dependências de uma classe

Frameworks de Injeção de Dependência

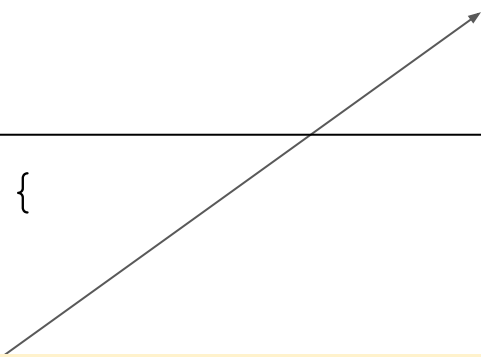
- Assumem a responsabilidade de:
 - Criar uma classe A
 - Criar e injetar as dependências de A
- Exemplos:
 - Java: Spring, Guice, Dagger2
 - TypeScript: InversifyJS
 - Go: wire

Exemplo: Classe na qual a dependência será injetada

```
class A {  
    private ServicoPagto pagto;  
  
    @Inject // anotação do framework  
    A(ServicoPagto pagto) {  
        this.pagto = pagto;  
    }  
    ...  
}
```

Exemplo: Classe que cria a classe da dependência

Nome hipotético de um
framework de injeção de
dependência



```
class Cliente {  
  
    void foo() {  
        A a = DIF.getInstance(A.class);  
        ...  
    }  
  
}
```

Pergunta: como o framework vai inferir a dependência que precisa injetar na classe A?

Como o framework vai inferir a dependência que precisa injetar na classe A?

- Esta informação é declarada em um arquivo externo
- Exemplo (hipotético):

```
{  "dependencies": {  
    "ServicoPagto": {  
        "class": "PgtoPayPal"  
    }  
    ...  
}
```

Exercício: Qual a principal vantagem de usar um DIF?

Design Docs

Design Docs

- Documento que descreve questões de design importantes:
 - Para implementação de um novo sistema
 - Para implementação de uma nova funcionalidade importante de um sistema
- Design Doc deve ser revisado e aprovado pelo time que ficará responsável pela implementação da nova funcionalidade ou sistema

Seções de um Design Doc

1. Contexto e objetivo do novo sistema ou funcionalidade
2. Design proposto, incluindo diagramas de alto nível
3. Trade-offs, i.e., pontos positivos e negativos do design proposto
4. Designs alternativos e justificativa de porque eles não estão sendo adotados
5. Impactos em requisitos não-funcionais, tais como desempenho, segurança, privacidade, etc

Tamanho de um Design Doc

- Não é um documento extenso
- Por exemplo, costuma ter cerca de 5 páginas

Fim