

UFMG
UNIVERSIDADE FEDERAL
DE MINAS GERAIS

Programação e Desenvolvimento de Software 2

Revisão de código e Refatoração

Prof. Douglas G. Macharet
douglas.macharet@dcc.ufmg.br

DCC
DEPARTAMENTO DE
CIÊNCIA DA COMPUTAÇÃO

Revisão de código

- Tarefa construtiva de rever o código e a documentação para identificar erros de interpretação, incoerências e outras falhas
 - Confirmação/verificação de outro membro antes de aceitar alterações ou novo código
- Propósito
 - Melhorar o código
 - Melhorar o programador

DCC UFMG

PDS 2 - Revisão de código e Refatoração

2

Revisão de código

Benefícios

- Taxa média de detecção de defeitos
 - Inspeções de design e código: 55% – 60%
- 11 programas desenvolvidos (mesma equipe)¹
 - 5 (sem revisões): 4,50 erros a cada 100 LoC
 - 6 (com revisões): 0,82 erros a cada 100 LoC
- Conhecimento compartilhado
 - Melhor entendimento do código
 - Programadores iniciantes (novos na equipe)

¹Freedman D. P., and Weinberg G. M. (1982) Software Inspections: An Effective Verification Process. IEEE Software.

DCC UFMG

PDS 2 - Revisão de código e Refatoração

3

Revisão de código

Dinâmica

Quem

- Desenvolvedor do código e o responsável pela revisão (desenvolvedor mais experiente), às vezes juntos pessoalmente, às vezes separados

Como

- Revisor dá sugestões de melhoria em um nível lógico e/ou estrutural, de acordo com conjunto previamente acordado de padrões de qualidade
- Correções são feitas até uma eventual aprovação do código

Quando

- Após o autor do código finalizar uma alteração do sistema (não muito grande/local), que está pronta para ser incorporada ao restante

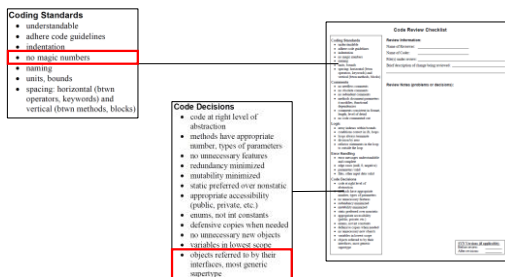
DCC UFMG

PDS 2 - Revisão de código e Refatoração

4

Revisão de código

Checklist



DCC UFMG

PDS 2 - Revisão de código e Refatoração

5

Revisão de código

Indústria

"All code that gets submitted needs to be reviewed by at least one other person, and either the code writer or the reviewer needs to have readability in that language. Most people use Mondrian [Rietveld] to do code reviews, and obviously, we spend a good chunk of our time reviewing code."

– Amanda Camp, Software Engineer, Google

Modern Code Review: A Case Study at Google: <https://ai.google/research/pubs/pub47025>

DCC UFMG

PDS 2 - Revisão de código e Refatoração

6

Revisão de código

Indústria – Git

- Pull request
 - Requisição para que o novo código seja integrado ao restante do sistema



<https://isham.hm/2016/01/01/how-to-make-a-pull-request-on-github-a-quick-tutorial/>
<https://homes.cs.washington.edu/~merrisaadvice/github-pull-request.html>

DCC

PDS 2 - Revisão de código e Refatoração

7

Revisão de código

Exemplo

```
#include <cmath>

class Account {
public:
    double principal, rate; int daysActive, accountType;
    int STANDARD = 0, BUDGET=1, PREMIUM=2, PREMIUM_PLUS = 3;

    double calculateFee(Account accounts[], int n) {
        double totalFee = 0.0;
        Account account;
        for (int i=0; i<n; i++) {
            account = accounts[i];
            if (account.accountType == PREMIUM ||
                account.accountType == PREMIUM_PLUS)
                totalFee += <math>0.025 \times</math>
                    account.principal * pow(account.rate, (account.daysActive/365.25));
            totalFee += account.principal;
        }
        return totalFee;
    }
};
```

DCC

PDS 2 - Revisão de código e Refatoração

8

Revisão de código

Exemplo

```
#include <cmath>
#include <iostream>

// An individual account. Also see CompoundAccount.
class Account {
private:
    Type accountType;
    double _principal = 0;
    // the yearly, compounded rate (at 365.25 days per year).
    double _rate = 0;
    // Days since last interest payout.
    int _daysActive = 0;

public:
    // the variation of account one bank offers.
    enum Type {STANDARD, BUDGET, PREMIUM, PREMIUM_PLUS};
    // the portion of the interest that goes to the banks.
    static constexpr double BUDGET_FEE_FRACTION = 0.025;

    Account(Type type) : _accountType(type) {}

    // compute interest on this account.
    double interest() {
        double years = daysActive / 365.25;
        double compoundInterest = _principal * pow(_rate, years);
        return compoundInterest - _principal;
    }

    // Return true if this is a premium account.
    bool isPremium() {
        return _accountType == Type::PREMIUM ||
            _accountType == Type::PREMIUM_PLUS;
    }

    // Return the sum of interest from the all given accounts.
    static double calculateFeeList(Account* accounts) {
        double totalFee = 0.0;
        for (auto account : accounts) {
            if (account.isPremium()) {
                totalFee += BUDGET_FEE_FRACTION * account.interest();
            }
            return totalFee;
        }
    }
};
```

Necessário?



DCC

PDS 2 - Revisão de código e Refatoração

9

Revisão de código

Exemplo

```
int main() {
    Account acl(Account::Type::PREMIUM);
    list<Account> accs;
    accs.push_back(acl);
    double totalFee = Account::calculateFee(accs);
    cout << totalFee << endl;
    return 0;
}
```

DCC

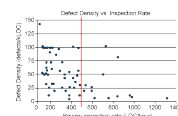
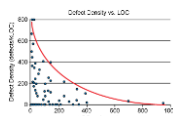
PDS 2 - Revisão de código e Refatoração

10

Revisão de código

Boas práticas

- Estabeleça metas quantificáveis para revisão
- Utilize listas de verificação
- Verifique se os defeitos são consertados
- Você não irá revisar tudo de uma vez!



<https://www.ibm.com/developerworks/rational/library/11-proven-practices-for-peer-review/index.html>

DCC

PDS 2 - Revisão de código e Refatoração

11

Refatoração

- Fatoração
 - Na matemática consiste em representar um número ou expressão como produto de fatores
- Refatoração
 - Processo de reescrever algum material para melhorar sua estrutura de maneira geral
- Na fatoração o resultado muda?
 - Qual a diferença então?
 - Outra representação → Ganhos implícitos

DCC

PDS 2 - Revisão de código e Refatoração

12

Refatoração

- Modificação (pequena) no sistema que não altera o comportamento funcional, mas que melhora qualidades não funcionais
 - Flexibilidade, clareza, robustez, ...
- Alteração no *design* de uma aplicação
 - Atividade que estava implícita
 - Preceito básico de eXtreme Programming (XP)

Refatoração

- Design (Code) Smells
 - Características (odores) que são perceptíveis em softwares de má qualidade (podres)
 - Rigidez, Fragilidade, Imobilidade, Viscosidade, Complexidade, Repetição, Opacidade
- Propor as refatorações adequadas a partir da identificação de um desses problemas

Refatoração



Exemplos

- Mudança no nome de variáveis e métodos
- Redução de código duplicado
 - É mais fácil fazer um "Ctrl-c, Ctrl-v"
- Generalizar/flexibilizar métodos
- Membros não encapsulados (públicos)
- Mudanças arquiteturais
 - Classes, Interfaces, ...

Refatoração

- Não é uma reestruturação arbitrária
 - Código ainda deve funcionar (não inserir bugs)
 - Testes tentam provar/garantir isso
 - Pequenas mudanças (não reescrever tudo)
 - A semântica deve ser preservada
 - Resultado
 - Alta coesão / Baixo acoplamento
 - Reusabilidade, legibilidade, testabilidade, ...

Refatoração

- Coesão 
 - Grau de dependência entre os elementos internos de um mesmo módulo
 - Funções, responsabilidades (mesmo objetivo)
- Acoplamento 
 - Grau de interdependência entre módulos
 - Alteração em um demanda alteração no outro

Refatoração

- Quais são exemplos de refatoração?
 - Adicionar novas funcionalidades
 - Melhorias no desempenho
 - Correção de erros existentes
 - Detecção de falhas de segurança



" ... build your program in a well-factored manner without paying attention to performance until you begin a performance optimization stage, usually fairly late in development. During the performance optimization stage, you follow a specific process to tune the program."
 — Martin Fowler, Refactoring, p. 59

Refatoração

- Quando fazer
 - Encontrou um “bad smell”
 - Sabe uma maneira *melhor* de fazer as coisas
 - Alteração não vai quebrar o código
- Quando NÃO fazer
 - Código estável que não precisa mudar
 - Prazo para entrega se aproximando
 - Pouco conhecimento do código (terceiros)

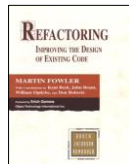
Refatoração

- Geralmente são mudanças simples
 - Operações sistemáticas e óbvias
 - Catálogo de refatorações [Fowler, 1999]
- Localmente pode não ser tão perceptível, porém no todo o impacto é considerável

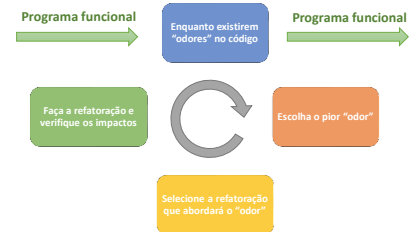
“If you want to refactor, the essential precondition is having solid tests.”
– Martin Fowler, Refactoring, p. 89

Refatoração

Problema	Refatoração
Código duplicado	Extract Method Substitute Algorithm
Método muito longo	Extract Method Introduce Parameter Object
Classe muito grande	Extract Class Extract Subclass Extract Interface
Feature Envy	Extract Method Move Method
Inappropriate Intimacy	Extract Class Hide Delegate Move Field Move Method



Refatoração Ciclo



Refatoração Exemplo 1

```

class Employee {
};

class Salesman : public Employee {
public:
    void getName() {
        cout << "Salesman" << endl;
    }
};

class Engineer : public Employee {
public:
    void getName() {
        cout << "Engineer" << endl;
    }
};
  
```

Refatoração Exemplo 1

```

class Employee {
public:
    virtual void getName();
};

class Salesman : public Employee {
public:
    void getName() override {
        cout << "Salesman" << endl;
    }
};

class Engineer : public Employee {
public:
    void getName() override {
        cout << "Engineer" << endl;
    }
};
  
```

Refatoração

Exemplo 1

■ Pull Up Method

You have methods with identical results on subclasses.

Move them to the superclass.

DCC 

PDS 2 - Revisão de código e Refatoração

25

Refatoração

Exemplo 1 – Pull Up Method

■ Motivação

- Métodos duplicados funcionam (comportamento), porém são fontes de problemas futuros (alterações)

■ Mecânica

- Inspect the methods to ensure they are identical.
 - If the methods look like they do the same thing but are not identical, use algorithm substitution on one of them to make them identical.
- If the methods have different signatures, change the signatures to the one you want to use in the superclass.
- Create a new method in the superclass, copy the body of one of the methods to it, adjust, and compile.
 - If you are in a strongly typed language and the method calls another method that is present on both subclasses but not the superclass, declare an abstract method on the superclass.
 - If the method uses a subclass field, use Pull Up Field or Self Encapsulate Field and declare and use an abstract getting method.
- Delete one subclass method.
- Compile and test.
- Keep deleting subclass methods and testing until only the superclass method remains.
- Take a look at the callers of this method to see whether you can change a required type to the superclass.

DCC 

PDS 2 - Revisão de código e Refatoração

26

Refatoração

Exemplo 2

```
class Employee {
    private:
        double _salesQuota;
};

class Salesman : public Employee {
};

class Engineer : public Employee {
};
```

DCC 

PDS 2 - Revisão de código e Refatoração

27

Refatoração

Exemplo 2

```
class Employee {
};

class Salesman : public Employee {
    private:
        double _salesQuota;
};

class Engineer : public Employee {
};
```

DCC 

PDS 2 - Revisão de código e Refatoração

28

Refatoração

Exemplo 2

■ Push Down Field

A field is used only by some subclasses.

Move the field to those subclasses.

DCC 

PDS 2 - Revisão de código e Refatoração

29

Refatoração

Exemplo 2 – Push Down Field

■ Motivação

- Push Down Field é oposto ao Pull Up Field. Use-o quando você não precisar de um campo na superclasse, mas apenas em uma subclasse.

■ Mecânica

- Declare the field in all subclasses.
- Remove the field from the superclass.
- Compile and test.
- Remove the field from all subclasses that don't need it.
- Compile and test.

DCC 

PDS 2 - Revisão de código e Refatoração

30

Refatoração

Exemplo 3

```
class Project {
public:
    vector<Person> participants;
};

class Person {
public:
    int id;
    bool participate(Project p) {
        for(auto x : p.participants) {
            if (x.id == id)
                return true;
        }
        return false;
    }
};

... if (x.participate(p)) ...
```

DCC

PDS 2 - Revisão de código e Refatoração

31

Refatoração

Exemplo 3

```
class Project {
private:
    vector<Person> participants;
public:
    bool participate(Person x) {
        for(auto person : participants) {
            if (person.id == x.id)
                return true;
        }
        return false;
    }
};

... if (p.participate(x)) ...
```

DCC

PDS 2 - Revisão de código e Refatoração

32

Refatoração

Exemplo 3

■ Move Method

A method is, or will be, using or used by more features of another class than the class on which it is defined.

Create a new method with a similar body in the class it uses most. Either turn the old method into a simple delegation, or remove it altogether.

DCC

PDS 2 - Revisão de código e Refatoração

33

Refatoração

Exemplo 3 – Move Method

■ Motivação

- Quando classes têm muito comportamento ou estão colaborando demais (alto acoplamento)

■ Mecânica

- Examine all features used by the source method that are defined on the source class. Consider whether they also should be moved.
 - For features used only by the method you are about to move, you might as well move it, too. If the feature is used by other methods, consider moving them as well. Sometimes it is easier to move a clutch of methods than to move them one at a time.
- Check the sub- and super-classes of the source class for other declarations of the method.
 - If there are any other declarations, you may not be able to make the move, unless the polymorphism can also be expressed on the target.
- Declare the method in the target class.
 - You may choose to use a different name, one that makes more sense in the target class.
- Copy the code from the source method to the target. Adjust the method to make it work in its new home.
 - If the method uses its source, you need to determine how to reference the source object from the target method. If there is no mechanism in the target class, pass the source object reference to the new method as a parameter.
 - If the method includes exception handlers, decide which class should logically handle the exception. If the source class should be responsible, leave the handlers behind.
- Complete the target class.
 - Determine how to reference the correct target object from the source.
 - There may be an existing field or method that will give you the target. If not, use whether you can easily create a method that will do so. Failing that, you need to create a new field in the source that can store the target. This may be a permanent change, but you can also make it temporary until you have refactored enough to remove it.
- Compile and test.
- Decide whether to remove the source method or retain it as a delegating method.
 - Leaving the source as a delegating method is easier if you have many references.
- If you remove the source method, replace all the references with references to the target method.
 - You can compile and test after changing each reference, although it is usually easier to change all references with one search and replace.
- Compile and test.

DCC

PDS 2 - Revisão de código e Refatoração

34

Refatoração

Exemplo 4

```
class Department {
public:
    double getTotalAnnualCost();
    string getName();
    int getHeadCount();
};

class Employee {
public:
    double getAnnualCost();
    string getName();
    int getID();
};
```

DCC

PDS 2 - Revisão de código e Refatoração

35

Refatoração

Exemplo 4

```
class Party {
public:
    virtual double getAnnualCost();
    string getName();
};

class Department : public Party {
public:
    double getAnnualCost() override;
    int getHeadCount();
};

class Employee : public Party {
public:
    double getAnnualCost() override;
    int getID();
};
```

DCC

PDS 2 - Revisão de código e Refatoração

36

Refatoração

Exemplo 4

■ Extract Superclass

You have two classes with similar features.

Create a superclass and move the common features to the superclass.

DCC 

PDS 2 - Revisão de código e Refatoração

37

Refatoração

Exemplo 4 – Extract Superclass

■ Motivação

- Uma forma de código duplicado é quando se tem duas classes que fazem coisas da mesma maneira ou coisas semelhantes de maneiras diferentes

■ Mecânica

- Create a blank abstract superclass; make the original classes subclasses of this superclass.
- One by one, use Pull Up Field, Pull Up Method, and Pull Up Constructor Body to move common elements to the superclass.
 - It's usually easier to move the fields first.
 - If you have subclass methods that have different signatures but the same purpose, use Rename Method to get them to the same name and then use Pull Up Method.
 - If you have methods with the same signature but different bodies, declare the common signature as an abstract method on the superclass.
 - If you have methods with different bodies that do the same thing, you may try using Substitute Algorithm to copy one body into the other. If this works, you can then use Pull Up Method.
- Compile and test after each pull.
- Examine the methods left on the subclasses. See if there are common parts; if there are you can use Extract Method followed by Pull Up Method on the common parts. If the overall flow is similar, you may be able to use Form Template Method.
- After pulling up all the common elements, check each client of the subclasses. If they use only the common interface you can change the required type to the superclass.

DCC 

PDS 2 - Revisão de código e Refatoração

38

Considerações finais

■ Benefícios Desenvolvedores

- Melhora manutenção
- Aumenta reusabilidade
- Facilita testes automatizados
- Alta coesão, Baixo acoplamento
- Maior robustez do código
- Facilita o trabalho em equipe

■ Benefícios Negócio

- Baixo índice de erros (\$)
- Código de maior valor (\$)
- Facilita adaptar requisitos (\$)
- Fácil adicionar novas features (\$)
- Produto liberado mais rápido (\$)
- Atacar Desempenho/Segurança (\$)

"In my view refactoring is not an activity you set aside time to do. Refactoring is something you do all the time in little bursts. You don't decide to refactor, you refactor because you want to do something else, and refactoring helps you do that other thing."
— Martin Fowler, Refactoring, p. 49

DCC 

PDS 2 - Revisão de código e Refatoração

39