

## REFATORANDO O SIMGRIP: UM ESTUDO DE CASO ACERCA DA APLICAÇÃO DE TÉCNICAS DE REFATORAÇÃO DE SOFTWARE

**Carlos LIMA(1); Gabriela GUEDES(2); Luiz CHAVES(3); Frederico PEREIRA(4);  
Marcelo SIQUEIRA(5)**

(1) Centro Federal de Educação Tecnológica da Paraíba, e-mail: [diegoquirino@gmail.com](mailto:diegoquirino@gmail.com)

(2) Centro Federal de Educação Tecnológica da Paraíba, e-mail: [gabriguedes@gmail.com](mailto:gabriguedes@gmail.com)

(3) Centro Federal de Educação Tecnológica da Paraíba, e-mail: [lucachaves@gmail.com](mailto:lucachaves@gmail.com)

(4) Centro Federal de Educação Tecnológica da Paraíba, e-mail: [fred@cefetpb.edu.br](mailto:fred@cefetpb.edu.br)

(5) Centro Federal de Educação Tecnológica da Paraíba, e-mail: [marcelo@cefetpb.edu.br](mailto:marcelo@cefetpb.edu.br)

### RESUMO

Apesar do reconhecido amadurecimento da Engenharia de *Software*, a forma como as aplicações no geral vêm sendo desenvolvidas, sem contemplar fatores como manutenibilidade e legibilidade de código, compromete significativamente a sua evolução. Existe o consenso entre os desenvolvedores de que adicionar funcionalidades ao *software* não é tarefa simples. Princípios da Engenharia de *Software* são subtraídos, sobretudo na fase de implementação, muitas vezes devido às pressões dos cronogramas não-realistas. A adoção de métodos sistemáticos baseados na experiência bem sucedida de outros projetos é a solução mais recomendada. O objetivo deste trabalho é demonstrar a eficácia de dois desses métodos, os Padrões de Projeto e a Refatoração de Código, em sistemas construídos sob o paradigma de Orientação a Objetos através de um estudo de caso com a aplicação SimGrIP (Simulador Gráfico de Roteamento IP) como forma de melhorar a qualidade de *software*. Tomou-se por escopo a geração da interface gráfica do *software*, o que possibilitou análise e composição dos artefatos que, ao final, serviram para escolha das técnicas e dos padrões adequados. Por fim, comparou-se o código obtido com o inicial e, com base em critérios como legibilidade, acoplamento e flexibilidade, para verificar se houve melhorias nestes quesitos.

**Palavras-chave:** Engenharia de Software, Refatoração, Padrões de Projeto, Manutenção de código.

## 1. INTRODUÇÃO

Na última década, a Engenharia de Software apresentou diversos avanços significativos, muitos deles graças ao surgimento de novas tecnologias e metodologias de desenvolvimento com a finalidade de, dentre outras coisas, oferecer produtos de melhor qualidade e que pudessem atender de forma satisfatória as demandas e anseios dos usuários em geral.

Nesta perspectiva, a **refatoração de código** surge como uma técnica capaz de possibilitar uma melhoria no projeto e na estrutura do código de uma aplicação, de maneira tal que a mesma possa acompanhar as constantes mudanças das regras de negócio. Um outro aspecto importante diz respeito ao fato de essa técnica apresentar um conjunto de orientações a fim de melhorar a qualidade de um software já existente, conseqüentemente demandando menos esforços aos seus desenvolvedores (FOWLER, 1999).

Utilizar **padrões de projetos** é fundamental. A estrutura codificada do programa necessita seguir um padrão de construção, de chamada de métodos, de regras fixas para comunicação entre objetos, assim como persistir corretamente os dados manipulados.

Tais fundamentos levaram ao estudo da estrutura de código do SimGrIP, um software de caráter educativo cujo objetivo é simular redes IP para fins didáticos de ensino de conceitos chaves de roteamento IP. Este software está parcialmente implementado, o que inclui a construção gráfica e de negócio de hosts, roteadores e links, além da apresentação para o usuário (janelas, botões, entre outros). Porém, a simulação do encaminhamento dos pacotes não estava previamente concluída.

Diante deste fato foi feito um levantamento inicial dos principais pontos críticos da estrutura do código existente na referida ferramenta para só então decidir sobre a refatoração a ser realizada.

O embasamento teórico desta pesquisa e as soluções vitais para o melhoramento da infra-estrutura de código do SimGrIP levam em consideração o conceito de refatoração e os exemplos propostos por Fowler (1999), assim como a aplicação de padrões de projetos propostos por Gamma e colegas (GAMMA et al., 1995). Outras fontes bibliográficas, como Willian C. Wake e consultas informativas à Internet, também foram utilizadas, com o intuito de esclarecer e pormenorizar os estudos propostos, tudo isto para apontar os erros na estrutura de código do SimGrIP e corrigir as falhas identificadas no software em análise.

Neste trabalho são apresentados alguns conceitos de refatoração e de padrões de projeto. É apresentado também o estudo de caso realizado com a aplicação SimGrIP, listando alguns problemas encontrados, soluções utilizadas e resultados obtidos.

## 2. METODOLOGIA

O SimGrIP foi desenvolvido no Centro Federal de Educação Tecnológica da Paraíba a fim de apoiar o ensino do funcionamento de redes de computadores baseadas no protocolo IP (*Internet Protocol*).

Recentes necessidades de expansão encontraram impasse e dificuldade de serem realizadas, pois além de haver um forte acoplamento entre algumas classes, havia trechos de código de difícil compreensão. Isso evidenciou uma necessidade por refatoração. Inicialmente, foi feita uma análise do código e um levantamento das informações necessárias para discutir mais detalhadamente a melhor forma de começar a re-organizar a estrutura da aplicação.

Segundo Fowler (1999), a refatoração também deixa o código mais fácil de compreender, então, na escolha da funcionalidade a ser refatorada, levou-se em consideração a clareza do código também. Pois um código mais legível permitiria que, no futuro, os desenvolvedores, ao relerem determinado trecho, compreendessem a função que este desempenhava mais facilmente.

Na fase seguinte, feito o levantamento dos pontos críticos e, escolhido o estudo de caso, iniciaram-se as aplicações das técnicas de refatoração e da utilização de padrões de projeto. Neste momento, o objetivo principal era o de revitalizar uma funcionalidade da estrutura de código do SimGrIP: **a interface de apresentação para o usuário** (ou seja, a janela principal do sistema). Nesta etapa foram utilizados o ambiente de programação Eclipse, versão 3.2.2, e a plataforma de desenvolvimento J2SE 6.0, além do repositório de dados CVS (*Concurrent Version System*), o que possibilitava interação permanente entre a equipe de refatoração.

Atualmente, o sistema se encontra com uma funcionalidade refatorada e com outros pontos de refatoração detectados. Executadas as devidas correções, poder-se-á inicializar a fase de testes para validação. Após esta etapa, o SimGrIP finalmente poderá continuar ser utilizado pelos clientes.

### 3. CONCEITOS BÁSICOS

#### 3.1. Refatoração de código

**Refatorar código** é transformar um código já existente e problemático, às vezes caótico, em algo legível e de fácil manutenção sem, contudo, alterar a aparência e o funcionamento externo do software.

A última consideração da definição anterior é bastante frisada por Fowler (1999) em sua afirmação “[...] *refatoração não muda o comportamento observável do software. O software ainda realiza a mesma função que realizava antes*”.

Refatorar significa adequar o código aos padrões. Quando um código é bem estruturado, é mais fácil de manipulá-lo, de alterá-lo e de encontrar possíveis falhas. E foi com esse objetivo que surgiram as técnicas de refatoração.

#### 3.2. Técnicas de Refatoração

Até agora foram citados os benefícios da refatoração, mas ela pode aumentar os problemas do desenvolvedor se praticada precipitadamente, sem antes ser feito um levantamento geral dos principais problemas do código a ser alterado.

Primeiro, deve-se olhar o código como um todo, procurar os principais problemas e trabalhar um de cada vez. A refatoração deve ser feita através de pequenas mudanças e, a cada mudança, testes devem ser feitos com o software para garantir que os resultados obtidos são sempre os mesmos. Fowler (1999) fala que “[...] *o efeito cumulativo dessas pequenas mudanças pode melhorar radicalmente o projeto*”.

Existem diversos livros, manuais e tutoriais que ajudam o programador a encontrar os principais erros de um código. Além disso, a maioria deles fornece um passo a passo sobre como proceder com a refatoração para cada tipo de erro.

As técnicas usadas neste estudo de caso foram: *Extract Class*, *Extract Method*, *Move Method* e *Inline Temp*. A ocasião em que cada uma foi utilizada será descrita em outra seção deste artigo.

#### 3.3. Padrões de Projeto

Em seu livro Gamma, Helm, Johnson e Vlissides (1995) fazem a seguinte pergunta: “*Quantas vezes você teve um déjà-vu de um projeto – aquele sentimento de que você resolveu o problema antes, mas não sabe exatamente onde ou como?*”.

*Foi para permitir que os desenvolvedores se lembrem de uma solução e possam reutilizá-la que existem os padrões de projetos. Os padrões de projetos descrevem os problemas e suas soluções, para que estas possam ser utilizadas sempre que necessárias.*

Um padrão de projeto possui quatro elementos essenciais:

- O **nome** que será usado para descrever o padrão. Nomear os padrões ajuda a criar um vocabulário comum entre os desenvolvedores e para fins de documentação;
- O **problema** que descreve as situações onde o padrão pode ser aplicado;
- A **solução** que descreve de forma abstrata como organizar os elementos (classes e objetos) de maneira a melhorar o projeto;
- As **conseqüências** que resultam ao aplicar o padrão;

Assim sendo, **padrões de projeto** constituem estruturas recorrentes no projeto de software orientado a objetos com a finalidade de guiar a metodologia de desenvolvimento, criando padronizações que devem ser

obedecidas por todos os integrantes da equipe de desenvolvimento. Eles *nomeiam, abstraem e identificam* aspectos chave de uma estrutura de projeto, tornando útil e adaptável toda a estrutura de código.

#### 4. IMPORTÂNCIA DA REFATORAÇÃO DE CÓDIGO

A importância da refatoração consiste em cada vez mais agilizar o processo de manutenção e facilidade de extensão. Como o desenvolvimento de software está associado a uma grande variabilidade de concretização, então podem ser encontradas muitas soluções, e em grande parte, soluções ruins. E isso, às vezes, acaba se tornando uma constante em equipes de desenvolvimento de software ainda inexperientes, que geralmente são cobradas pelo tempo e custo, o que acaba acarretando falta de modelagem e planejamento de boas maneiras de implementação dos requisitos do sistema. Esse fato é atestado por Fowler, que cita: “*Números significativos de projetos inadequados de programas têm sido criados por desenvolvedores menos experientes, resultando em aplicações que são ineficientes e de difícil manutenção e extensão*”.

Para evitar isso, a refatoração é composta de técnicas que no geral são simples, porque uma pequena modificação no sistema, que não altere o seu comportamento funcional, já garante uma grande melhoria. E elas se fortalecem principalmente se forem baseadas em algumas mudanças de qualidade não-funcional: simplicidade, flexibilidade, clareza, desempenho. Alguns dos seus exemplos são:

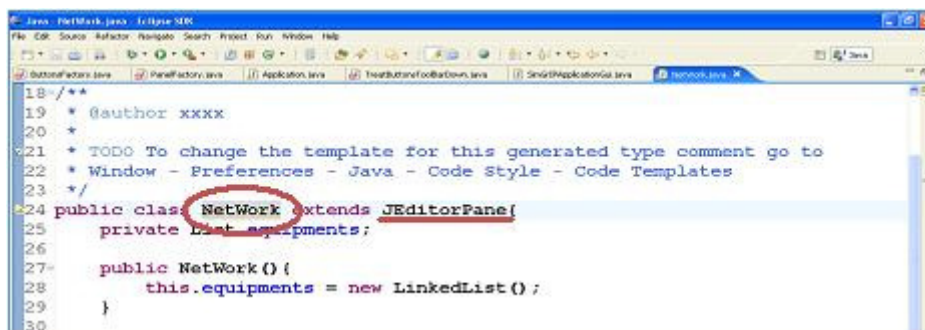
- A mudança de trechos de códigos para outras áreas;
- A exclusão de trechos repetitivos ou redundantes;
- A mudança de alguns trechos para um escopo de algum método, ou de um único método;
- O deslocamento de certas características para uma hierarquia maior (generalização);
- Fazer um detalhamento dos requisitos do sistema a fim de obter uma melhor modelagem do código, pois com a introdução dos requisitos é que se parte para o problema;
- Renomear algumas variáveis, métodos ou classes com o objetivo de manter uma maior correspondência com seu sentido real;
- Aplicar padrões de projeto para garantir modularização;

#### 5. ESTUDO DE CASO: REFATORAÇÃO DA INTERFACE DE APRESENTAÇÃO AO USUÁRIO

##### 5.1. Análise preliminar

Analizando o código do software SimGrIP e seu diagrama de classes, notou-se alguns pontos que necessitavam de refatoração. Dentre eles pode-se citar:

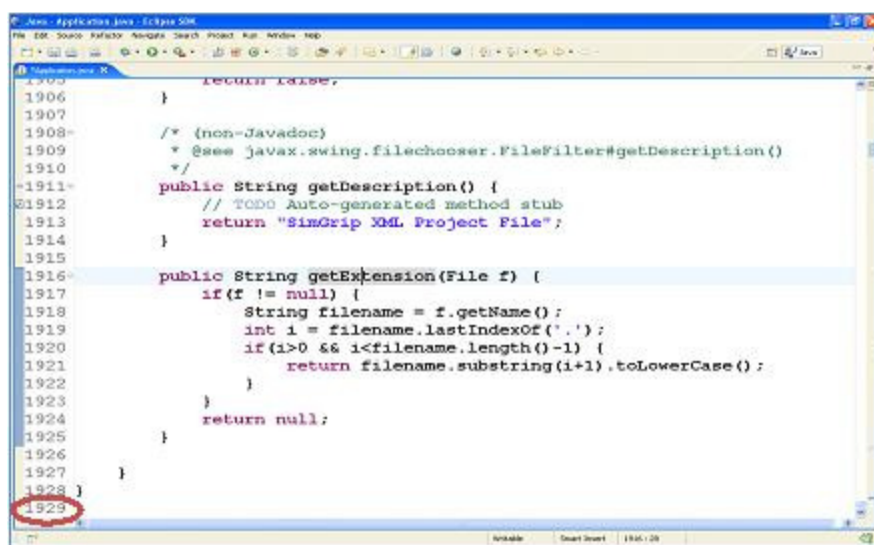
- A classe *Application*, que sozinha montava, controlava e tratava os elementos de interface. Isso a tornava uma estrutura monolítica.
- Uma classe interna à classe *Application* para manipular arquivos *xml*.
- Nomes de classes não condizentes com seus papéis. A Figura 1 apresenta um trecho de código de uma classe cujo nome é *NetWork*, mas cuja finalidade é criar um painel para exibir os elementos da rede.



```
18-/**
19- * @author xxxx
20- *
21- * TODO To change the template for this generated type comment go to
22- * Window -> Preferences -> Java -> Code Style -> Code Templates
23- */
24-public class NetWork extends JEditorPane{
25-    private List equipments;
26-
27-    public NetWork() {
28-        this.equipments = new LinkedList();
29-    }
30-}
```

Figura 1: Trecho de código da classe NetWork.

Além da repetição de código dentro da classe *Application*. Tudo isso somado à ausência de comentários e documentação, tornava o SimGrIP em um software difícil de ler e a inserção das outras funcionalidades numa tarefa muito custosa. Optou-se por fazer a refatoração na classe *Application*, pois esta, por concentrar um grande número de funções, contava com um número excessivo de classes internas – num total de dez – e toda a criação do frame de apresentação da interface estava contida em apenas um método. O arquivo que continha esta classe era enorme, com mais de mil e quinhentas linhas. Na Figura 2 é apresentado um trecho de código desta classe.



```
1906-    }
1907-
1908-    /* (non-Javadoc)
1909-     * @see javax.swing.filechooser.FileFilter#getDescription()
1910-     */
1911-    public String getDescription() {
1912-        // TODO Auto-generated method stub
1913-        return "SimGrip XML Project File";
1914-    }
1915-
1916-    public String getExtension(File f) {
1917-        if(f != null) {
1918-            String filename = f.getName();
1919-            int i = filename.lastIndexOf('.');
1920-            if(i>0 && i<filename.length()-1) {
1921-                return filename.substring(i+1).toLowerCase();
1922-            }
1923-        }
1924-        return null;
1925-    }
1926-
1927-}
1928-
1929-}
```

Figura 2: Trecho de código da classe Application.

## 5.2. Utilizando padrões

Antes de refatorar o SimGrIP, fez-se uma análise para determinar quais padrões de projeto poderiam ser introduzidos para dar mais flexibilidade ao projeto. Decidiu-se por utilizar os padrões de projeto apresentados na Tabela 1.

Tabela 1 - Descrição de padrões e seus usos no SimGrIP

Padrão	Utilização
Factory Method	Este padrão foi usado para flexibilizar a criação de componentes de interface, como botões, menus e painéis. Pode-se verificar sua implementação nas classes do pacote <i>br.edu.cefetpb.simgrip.gui</i> cujo nome termina com o palavra

<b>Command</b>	“Factory”.
<b>Façade</b>	Padrão utilizado no tratamento de eventos dos componentes da interface. As classes do pacote <i>br.edu.cefetpb.simgrip.gui.listener</i> implementam este padrão.
	O padrão Façade foi usado para a criação e manipulação de todo o frame contendo os componentes da interface. A classe <i>ApplicationGuiFacade</i> é a “fachada” da aplicação, ou seja, ela é a interface de acesso aos componentes do frame.

### 5.3. Aplicando técnicas de Refatoração.

Tendo os padrões definidos, foi necessário aplicar algumas técnicas de refatoração para implementar estes padrões. As técnicas mais usadas foram as descritas na Tabela 2.

Tabela 2 - Descrição de técnicas de refatoração e seus usos no SimGrIP

<i>Técnica de Refatoração</i>	<i>Utilização</i>
<b>Extract Class</b>	Técnica que explana os procedimentos necessários para a extração de uma classe de dentro de outra. Utilizada no SimGrIP para extrair as classes internas e fábricas.
<b>Extract Method</b>	Técnica que descreve como extrair métodos menores e concisos de um método extenso que faz várias tarefas. Foi usada para dividir o método <i>main(String args[])</i> da antiga classe <i>Application</i> em métodos menores, que foram inseridos nas fábricas, responsáveis por criar componentes específicos.
<b>Move Method</b>	Técnica que explica como mover métodos de uma classe para outra. Usada para mover os métodos de criação, previamente construídos a partir do uso da técnica <i>Extract Method</i> , para dentro das fábricas.
<b>Inline Temp</b>	Técnica usada para diminuir a criação de variáveis temporárias. Foi utilizada nas fábricas de painéis e menus, nas chamadas dos métodos de adição de componentes.

### 5.4. O produto final.

O SimGrIP conta com dois novos pacotes: um para criação da *Graphical User Interface* (GUI) e outro para tratá-la. Há uma melhor divisão de tarefas e uma modularização maior conforme pode ser visto na Figura 3.

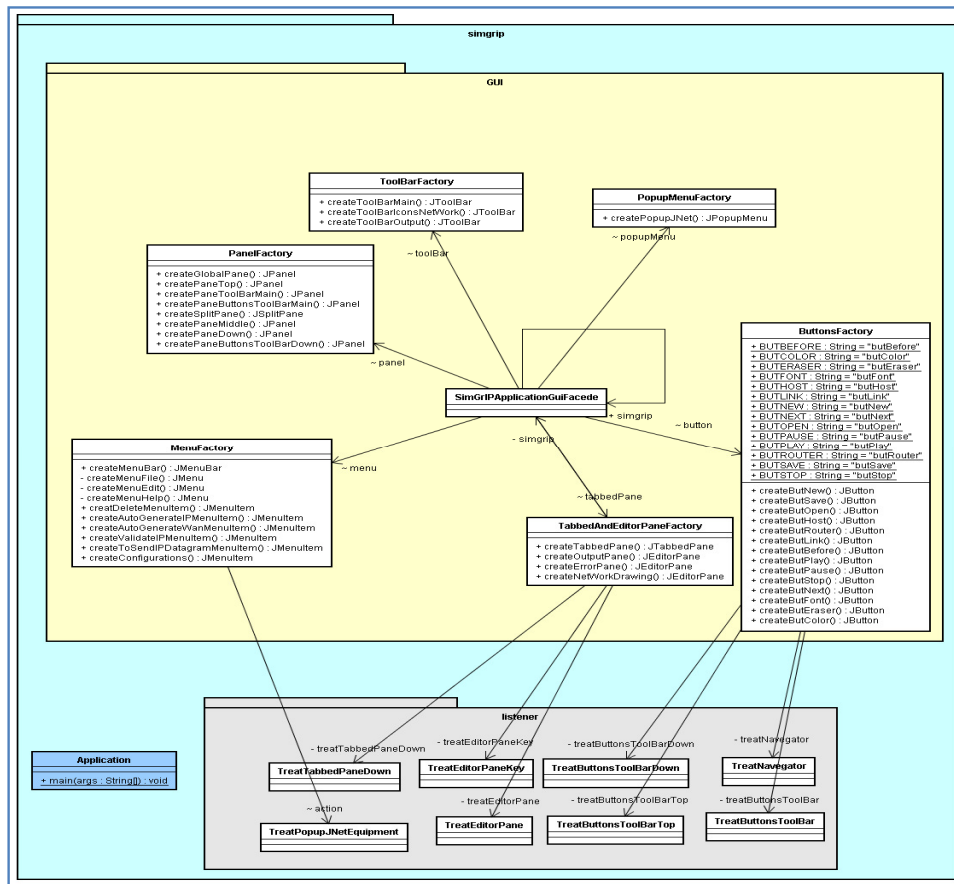


Figura 3: Arquitetura modular do SimGrIP.

A inserção, modificação e retirada de componentes da interface foi facilitada, pois agora, com o código bem dividido, sabe-se qual trecho lida com tais componentes e qual trecho modificar.

É necessário salientar que a interface do usuário continua a mesma, apesar de toda a modificação na estrutura interna. E é assim que a refatoração deve ser feita.

## 5.5. Pontos Críticos de Implementação / Refatoração Futura.

O ponto de partida para a refatoração feita foi uma classe apenas, *Application*, mas existem outras que não estão bem projetadas e devem ser alvos de refatorações futuras, como:

- Classes de modelo que deveriam apenas implementar a lógica do negócio, mas que ainda estão acessando e modificando elementos visuais diretamente, quebrando o padrão de design MVC.
- Métodos extensos, que implementam várias tarefas, cuja lógica pode ser dividida em um número maior de métodos.
- Classes que possuem métodos com comandos *switch* extensos e repetitivos.

Uma vez que estes problemas sejam solucionados, o software estará pronto para a adição das outras funcionalidades previstas anteriormente, como a geração automática de endereços IP para uma rede e o gerenciamento automático de tabelas de roteamento, por exemplo.

## 6. CONCLUSÕES E TRABALHOS FUTUROS

Dadas as necessidades de expansão, manutenção e legibilidade do código, a refatoração aplicada ao software SimGrIP possibilitou uma melhor adequação das funcionalidades modificadas aos seus requisitos funcionais, beneficiando desenvolvedores e clientes.

Utilizados os conceitos e técnicas de refatoração, obteve-se uma funcionalidade do software mais adequável e manutenível. A mesma poderá ter elementos incluídos com maior facilidade, ou seja, se tornará estendível, atualizável. Esses resultados estimulam a aplicação das técnicas de refatoração nas demais funcionalidades do sistema, onde foram detectados pontos críticos (item 5.5), passíveis de manutenção.

Como trabalho futuro, pretende-se aumentar o nível de refatoração e a aplicação de mais padrões de projeto. Dessa forma esperamos que a ferramenta possa acomodar facilmente novos elementos de redes de computadores não previstos na versão atual, como por exemplo *switches*, roteadores multicast, etc., dessa forma, ampliando o escopo das tecnologias estudadas pelos seus usuários.

Assim como a refatoração mudou o código do SimGrIP, ela pode beneficiar outras aplicações. Para softwares cujo código encontra-se pouco legível e claro, a refatoração pode ser uma ferramenta auxiliar para que os desenvolvedores aumentem a clareza do código. Para softwares de estrutura monolítica, pode ser uma solução na busca de uma arquitetura flexível.

Seja qual for a motivação inicial para a refatoração, é importante avaliar a código, analisar quais padrões de projeto podem ser empregados, traçar um plano de refatoração de forma que problemas grandes sejam divididos e as mudanças aconteçam aos poucos até que se chegue ao objetivo final.

## REFERÊNCIAS

FOWLER, Martin et al. **Refactoring: Improving the Design of Existing Code**. 1. ed. USA: Addison Wesley, 1999.

FREEMAN, Eric & FREEMAN, Elisabeth. **Head First Design Patterns**. 1. ed. USA: O'Reilly Media, 2004.

GAMMA, Erich; HELM, Richard; JOHNSON, Ralph & VLISSIDES, John **Design Patterns: Elements of Reusable Object-Oriented Software**. 1.ed. USA: Addison Wesley, 1995.

WAKE, William C. **Refactoring Workbook**. 1. ed. USA: Addison Wesley, 2003.