REFATORAÇÃO: UMA METODOLOGIA DE APLICABILIDADE DAS TÉCNICAS BASEADA EM UM MODELO EM NÍVEIS DE QUALIDADE E MATURIDADE DE CÓDIGO

Manoel Sancho da Silva Neto[1], Éldman de Oliveira Nunes[2]

Resumo. Diversos modelos de referência têm sugerido práticas necessárias para garantir a qualidade do processo de desenvolvimento de software. Entretanto, percebe-se a ausência de uma metodologia que vise a maturidade do código de programação e não apenas do processo de desenvolvimento. Este artigo propõe uma metodologia inspirada no modelo de referência CMMI que representa a maturidade e qualidade do código dos sistemas de *software* a partir da aplicação categorizada das técnicas de refatoração de código. A metodologia proposta é exemplificada em um estudo de caso em que um código, em seu mais baixo nível de qualidade, evolui passo-a-passo nos níveis de maturidade do modelo. A sequência da aplicação das técnicas de refatoração agrupadas em categorias conduzem ao aumento horizontal e vertical da qualidade do código, o primeiro como forma de aperfeiçoamento e o segundo como forma de elevação da maturidade do código.

Palavras-chave: Técnicas de Refatoração. Código. Qualidade. Maturidade. Software.

Abstract. Various models of reference have suggested practices necessary to ensure the quality of the software development process. However, we find the lack of a methodology aimed at the maturity of the code of programming and not just the process of development. This article proposes a methodology based on a reference model that represents the CMMI maturity and quality of the code of systems software from the application of the techniques categorized refactoring of code. The proposed methodology is exemplified in a case study where a code on their lower level of quality, moves step-by-step on levels of maturity of the model. The sequence of applying the techniques of refatoração grouped into categories lead to increased horizontal and vertical quality of the code, the first as a way of improving and the second as a way of lifting the maturity of the code.

Keywords: Refactoring Tecniques, Code, Quality, Maturity, Software

[1] Bacharelado em Análise de Sistemas/Universidade do Estado da Bahia. Escola de Administração do Exército(EsAEx), Salvador/BA, Brasil. manoelsancho@msn.com
[2] Doutorado em Ciência da Computação/Universidade Federal Fluminense. Escola de Administração do Exército (EsAEx), Salvador/BA, Brasil. eldman@bol.com.br

1 Introdução

Refatoração é um processo aplicado ao código de um sistema de *software* de forma a melhorar a sua estrutura interna sem, contudo, alterar o seu comportamento externo (FOWLER, 2004). É uma sistematização de técnicas que visam aperfeiçoar o projeto de código após ele ter sido escrito. Refatorar tem o objetivo de organizar e aprimorar a estrutura do código, tornando-o mais legível, facilitando a reusabilidade e diminuindo o tempo gasto com tarefas de manutenção.

Devido à importância e os resultados que a refatoração mostra ter, atualmente já é exigido dos desenvolvedores de grandes projetos de *software* o conhecimento das técnicas de refatoração, como ferramenta para construção de um código robusto, legível e bem projetado. Utilizando-a após a conclusão de um projeto, a lista das técnicas de refatorações conhecidas é extensa e não há uma metodologia ou uma sequência lógica para o uso delas de modo eficiente, apropriado para cada nível de qualidade de código encontrada nos mais variados *softwares* desenvolvidos.

Em uma visão mais ampla da engenharia de *software*, é notável a carência, por parte dos processos de desenvolvimento de sistemas, de uma metodologia que consiga avaliar e determinar em que estágio a codificação dos projetos está e quais são as ações necessárias para aperfeiçoá-lo e elevar seu nível de maturidade.

Antes mesmo de começar a se pensar em dividir e categorizar as técnicas de refatoração, é preciso definir algo que sustente a ideia de criar uma metodologia para a aplicabilidade das mesmas. Logo, como o objetivo final da refatoração está contemplado no alcance de uma melhor qualidade de escrita do código, e consequentemente no melhor desempenho e integridade estrutural dos projetos de *software*, foram realizadas pesquisas em vários códigos de sistemas existentes, no intuito de colher informações necessárias para estruturar o Modelo de Qualidade e Maturidade de Código (MQMC), este sim orientando a categorização que será realizada nas técnicas de refatoração. O modelo MQMC será apresentado na seção 4 deste artigo. Antes disso, serão contextualizados os objetivos propostos no artigo aos setores de possíveis áreas de interesse do Exército Brasileiro, e, em seguida, uma discussão dos modelos de qualidade e maturidade de processos de *software* existentes atualmente.

2 Desenvolvimento de sistemas no EB

Apesar dos esforços do Exército Brasileiro no aperfeiçoamento dos processos de engenharia de *software* e de qualidade dos projetos de sistemas, muito ainda pode ser melhorado no campo de desenvolvimento de *software* quando comparados com os exércitos das nações desenvolvidas.

A automatização dos tantos processos e atividades correntes do dia-a-dia da vida militar ajudaria bastante na redução da burocracia, no ganho de tempo, na maior confiabilidade das tramitações, entre outros aspectos.

Ainda assim, há setores no EB que participam no desenvolvimento de sistemas para a Força. Existem Organizações Militares cuja atividade fim é justamente essa, onde são construídos sistemas corporativos, enquanto que em outras, é uma atividade secundária e/ou intermitente. Essas, na maioria das vezes, são responsáveis pelo suporte a sistemas e as redes de computadores.

Durante a construção e a posterior manutenção dos sistemas do EB, os militares responsáveis devem conhecer e aplicar uma metodologia para o desenvolvimento de sistemas com qualidade. A metodologia proposta neste artigo pode ser implementada tanto durante, quanto após a construção do projeto de *software*. Nos dois casos, os códigos dos sistemas do Exército Brasileiro estariam ganhando maturidade e agregando os benefícios que a refatoração propicia.

3 Modelo de qualidade e maturidade

Um modelo de qualidade e/ou maturidade propõe-se a avaliar algo e subsidiar sua melhoria, de forma padronizada e por meio de práticas metodológicas bem fundamentadas e estruturadas. Dois modelos voltados para o desenvolvimento de *software* foram estudados como fonte para a idealização do MQMC: o CMMI e o MPS.BR.

O CMMI (*Capability Maturity Model Integration*) é um modelo de referência que contém práticas necessárias à garantia da maturidade em processos específicos das Organizações (CMMI, 2007). A exemplo de processos contemplados pelo modelo tem-se Engenharia de *Software*, Gerência de Requisitos, Gerência de Projetos e de Produtos. Desenvolvido pelo SEI (*Software Engineering Institute*), o CMMI procura estabelecer um modelo único para o processo de melhoria corporativo, cujo objetivo é a melhoria da maturidade e capacidade dos pro-

cessos.

O MPS.BR ou Melhoria de Processos do *Software* Brasileiro é simultaneamente um movimento para a melhoria e um modelo de qualidade de processo voltada para a realidade do mercado de pequenas e médias empresas de desenvolvimento de *software* no Brasil (MPS.BR, 2007).

4 Modelo de qualidade e maturidade de código

O Modelo de Qualidade e Maturidade de código (MQMC) procura estabelecer, como meta maior, um modelo único para o processo de melhoria na qualidade do código produzido. Esse modelo foi idealizado, a princípio, para vislumbrar e orientar, de maneira metodológica, o uso das técnicas de refatoração, categorizando suas aplicações em níveis, e definindo a forma com que cada uma afetará na qualidade da codificação. Porém nota-se que o modelo é mais amplo e capaz de avaliar a maturidade da codificação de sistemas de *software*.

4.1 Níveis do MQMC

Assim como o CMMI, o MQMC disponibiliza uma sequência pré-determinada para melhoria baseada em estágios que não deve ser desconsiderada, pois cada estágio serve de base para o seguinte (CMMI, 2007). Isto quer dizer que só se alcança um nível acima quando todos os níveis abaixo dele estiverem implementados. Funciona como um pré-requisito para o próximo.

A figura 1 ilustra os níveis do modelo e as atividades e objetivos que devem ser alcançados para alcançar o nível superior.

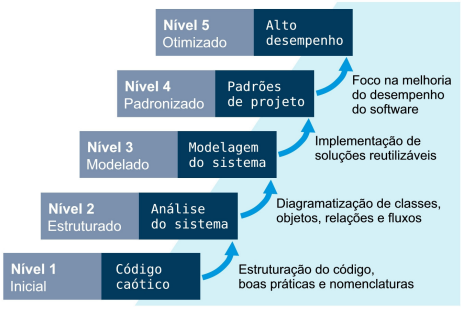

Figura 1 - MQMC proposto em 5 níveis

Na base do modelo, tem-se o **nível 1**, ou **inicial**, que é o código sem estruturação ou modelagem reconhecida, que não garante ao sistema um ciclo de vida longo, pois a manutenibilidade, legibilidade e extensibilidade do código estão visivelmente comprometidos. São feitos sem planejamento e na maioria das vezes, por apenas um desenvolvedor, sendo o único que detém o conhecimento ou entendimento do que está programado. É o código mal escrito, obscuro, implementado

com "gambiarras", e instável em sua execução. Não há preocupação com performance e redução de complexidade.

O **nível 2** é o código fonte **estruturado**, no qual há alguma qualidade na escrita, mas ainda não segue os conceitos da modelagem OO. Nota-se a aplicação de boas práticas de programação, com comentários nos trechos de códigos, adequada indentação dos comandos, padronização na nomenclatura das variáveis, métodos e parâmetros, entre outros. Verifica-se também uma divisão do código em módulos conceituais do sistema e uma preocupação em manter a aplicação em uma estrutura conveniente de diretórios.

O **nível 3** é o código **modelado**, que segue algum modelo ou programação de sistemas de *software*, como a Programação Orientada a Objetos, ou até a Orientação a Aspectos. Dentro do mundo OO, este nível de código implementa as unidades básicas de *software* chamadas de objetos, com seus atributos e métodos (ou estados e comportamentos), e o relacionamento e a troca de mensagens entre os mesmos. É esperado no código que, visto as possibilidades da orientação a objetos, seus conceitos fundamentais estejam bem definidos, como o polimorfismo, abstração, sobrecarga, encapsulamento, interface, além de outros.

O **nível 4**, ou código **padronizado**, é o sistema modelado, só que com implementações reutilizáveis de Padrões de Projeto. Também muito conhecido pelo termo original em inglês, *Design Patterns*, os Padrões de Projeto descrevem soluções para problemas recorrentes no desenvolvimento de sistemas de *software* orientados a objetos (GAMMA, 2000). São soluções de sucesso para os mais comuns e variados problemas encontrados em modelagens OO. Códigos assim padronizados aproximam-se do mais alto nível de manutenibilidade, legibilidade e extensibilidade.

Por fim, no topo do modelo tem-se o **nível 5**, que é o código **otimizado**, que apresenta soluções robustas e eficientes, em um nível de complexidade elevada. São aplicadas técnicas que focam no melhor desempenho algorítmico, muitas vezes específicos para uma determinada solução. Pode-se citar, como exemplos usados atualmente, os algoritmos genéticos, as redes neurais, tabelas hash, criptografias, algoritmos de compressão e busca, entre outros.

Pode ser questionado a possibilidade, por exemplo, de um código encontrar-se orientado a objetos, mas sem estruturação, totalmente desco-

mentado, com nomenclatura de métodos e variáveis inadequados. Não se deve pensar que, pelo fato de o código estar modelado segundo a programação OO, o mesmo se encontra no nível 3. Para que esse código possa estar no nível 3, todas as práticas dos níveis anteriores deveriam estar implementados. Logo, esse código ainda está no nível 1.

5 Técnicas de Refatoração

As técnicas de refatoração são, em resumo, soluções para os comuns e mais variados problemas encontrados na modelagem e codificação dos sistemas de *software*, que comprometem quesitos importantes para a evolução e a vida longa dos mesmos. Os principais deles são a legibilidade, reusabilidade, manutenibilidade, modularidade, extensibilidade, performance, clareza, simplicidade, dentre outros.

Surgiram em função do crescimento do tamanho e da complexidade dos programas, os quais vem tornando difíceis as tarefas de manter e compreender o comportamento de seus códigos.

Com a criação do Modelo de Qualidade e Maturidade de Código, é possível então identificar os grupos de técnicas que atuarão nos determinados estágios do modelo e definir a função que cada uma terá naquele nível. Todas elas possuem objetivos distintos e juntas formam objetivos ainda maiores. Estes serão categorizados e estruturados na medida em que a sua aplicabilidade forneça aumento da qualidade e aperfeiçoamento do projeto de código.

5.1 Descrição das técnicas selecionadas para categorização

As técnicas de refatoração de Fowler escolhidas para a categorização seguiram a melhor adequação aos critérios de objetividade e aplicabilidade, em razão e apreço ao entendimento da metodologia. A Tabela 1 descreve os objetivos de cada uma delas (FOWLER, 2004).

5.2 Categorização

Esta seção apresenta a divisão das principais técnicas em categorias, a fim de construir a metodologia de aplicabilidade das mesmas ao MQMC. Primeiramente, as técnicas foram separadas em 2 grupos:

- Grupo de Refatoração Vertical;
- Grupo de Refatoração Horizontal.

Tabela 1 – Descrições das técnicas selecionadas

Renomear Método	Adequação do nome do método ao seu funcionamento
Remover Flag de Controle	Substituir as flags de controle por declarações mais claras e eficientes.
Introduzir Variável Explicativa	Descomplicar uma expressão utilizando variáveis temporárias
Consolidar Expressão Condicional	Substituir expressões condicionais por um único método *booleano*.
Encapsular Campo	Tornar campo privado e fornecer métodos de acesso
Internalizar Variável Temporária	Substituir a variável temporária pela atribuição feita a ela.
Converter Projeto Procedural em Objetos	Transformar os registros de dados em objetos
Substituir Vetor por Objeto	Criar um objeto que tenha um campo para cada elemento do vetor
Extrair Superclasse	Criar superclasse e mover o que há de comum para ela
Descer Método na Hierarquia	Levar a devida responsabilidade do método para uma subclasse
Criar um Método Padrão	Criar método que contenha o que houver de duplicado nas subclasses
Substituir Enumeração pelo Padrão State/Strategy	Criar objeto que represente o estado do objeto original
Substituir o Construtor por um Método Fábrica	Implementar o Padrão de Projeto *Factory*

O GRV ou Grupo de Refatoração Vertical é um conjunto de técnicas de refatoração que propiciam a mudança do nível de maturidade de código dentro do modelo MQMC, sugerido neste artigo. São técnicas que, quando aplicadas, fornecem um incremento vertical indispensável para a evolução do código, conforme sugerido pelo modelo.

O GRH ou Grupo de Refatoração Horizontal é também um conjunto de técnicas de refatoração, que garantem um aperfeiçoamento na qualidade do código, mas o mantém no mesmo nível em que se encontrava no modelo. Códigos de sistemas que já estão fundamentados em um determinado nível por não ter a pretensão ou a viabilidade de elevá-lo podem, com eficiência, aplicar este grupo de técnicas e adquirir melhoria horizontal, tais como manutenibilidade, legibilidade e clareza no seu código.

A próxima categorização foi realizada com a separação das técnicas que iriam atuar em cada nível do modelo, de acordo com os objetivos e resultados que cada uma delas possui. Por exemplo, a técnica *Acrescentar Parâmetro* pode ser incluída no grupo GRH do nível 2, pois o acréscimo de parâmetro que é sugerido não contribui diretamente para elevar o estágio de maturidade do código e sim como melhoria na qualidade do código de nível 2. Já a técnica *Substituir Enumeração Por Classe,* por exemplo, deve ser inserida no grupo GRV para o nível 3. Isso deve-se porque o objetivo da técnica é colaborar para a

codificação orientada a objetos (nível 3), com ganho de maturidade por parte do código fonte.

A Tabela 2 apresenta a categorização das técnicas de refatoração selecionadas e descritas na Tabela 1.

Tabela 2 – Divisão das técnicas nas categorias propostas pela metodologia

Técnica	GRH	GRV	Do nível	P/ o nível
Renomear Método	-	X	1	2
Remover Flag de Controle	-	X	1	2
Introduzir Variável Explicativa	-	X	1	2
Consolidar Expressão Condicional	X	-	2	-
Encapsular Campo	X	-	2	-
Internalizar Variável Temporária	X	-	2	-
Converter Projeto Procedural em Objetos	-	X	2	3
Substituir Vetor por Objeto	-	X	2	3
Extrair Superclasse	X	-	3	-
Descer Método na Hierarquia	X	-	3	-
Criar um Método Padrão	X	-	3	-
Substituir Enumeração pelo Padrão State/Strategy	-	X	3	4
Substituir o Construtor por um Método Fábrica	-	X	3	4

A partir da tabela, nota-se que os níveis 4 e 5 ficam comprometidos pela falta de técnicas disponíveis a serem aplicadas, em especial o nível 5. Pode-se dizer que, esta ausência de refatoração nestes níveis, deve-se ao fato de já existir outros complementos e soluções de *software* que são cabíveis de aplicação. Porém, não se deve excluir a oportunidade de criar novas técnicas e preencher esta lacuna no campo da refatoração. No nível 4, somente duas técnicas que lidam com padrões de projeto foram encontradas. O complemento para este nível 4 seriam as próprias soluções reutilizáveis de *software* orientado a objetos, os conhecidos Padrões de Projeto. Segundo Gamma (2000, p.18), "os padrões de projeto podem melhorar a documentação e a manutenção de sistemas ao fornecer uma especificação explícita de interações de classes e objetos e o seu objetivo subjacente. Em suma, ajudam um projetista a obter mais rapidamente um projeto adequado".

No nível 5, que é o código otimizado, a única técnica de refatoração que poderia ser aplicada é a *Substituir Algoritmo*. Qualquer algoritmo que proporcionasse melhor desempenho e resultados otimizados serviria como candidato para a substituição daquele algoritmo no código. Mas na verdade, esta técnica sugere apenas a

substituição do algoritmo por um mais claro (FOWLER, 2004). Logo, sua utilização limita-se somente a favor da clareza do código. Porém, na visão das ideias deste artigo, esta técnica poderia contemplar também os objetivos de otimização, ampliando-se o escopo da mesma.

6 Aplicação das técnicas

Nesta seção, será apresentado um exemplo dos resultados da aplicação metodológica das técnicas de refatoração. Partir-se-á de um exemplo de código em nível 1 até alcançar o nível 4. O nivelamento para o 5º estágio não será demonstrado visto que, como já foi dito anteriormente, não se chega até o mesmo por meio de refatorações, pelo menos até o momento em que este artigo foi concebido.

O código-exemplo foi escrito na linguagem Java, que é uma linguagem de programação que tem, como principais características, a independência de plataforma, a disponibilidade de diversos recursos de rede, distribuição de um vasto conjunto de bibliotecas e possui facilidades para criação de programas distribuídos e multitarefa (JAVA, 2007).

O Quadro 1 inicia a aplicação-exemplo com o código em nível inicial de maturidade e qualidade.

Observe o código desestruturado, sem padronização de nomenclatura dos métodos e variáveis, que não denota um sentido lógico para suas funcionalidades. Pela tabela 1, para esse nível, é possível aplicar uma técnica GRV para o nível 2, o *Renomear Método*, conforme é apresentado no Quadro 2.

Quadro 1 – código-exemplo em nível inicial de maturidade

```
String vetor[] = new String[4];
vetor[0] = "joao.neto";
vetor[1] = "5jbsebf";
vetor[2] = "0";
vetor[3] = "1";

public void xxx(String[] u){
      if (u[2].equals("0") && u[3].equals("1")){
      ...
      }
      ...
}
```

Quadro 2 – código após a aplicação da técnica *Renomear Método*

```
Map Usuario = new Map();
// Map com informações do usuário
Usuario.put("Login", "joao.neto");
Usuario.put("Senha", "5jbsebf");
Usuario.put("TipoUsuario", "0");
// (0 = Cliente, 1 = Gerente)
Usuario.put("TipoAcesso", "1");
// (0 = Comum, 1 = Administrador, 2 = VIP)

public void logar(Map Usuario){
      if (Usuario.get(“TipoUsuario”).equals("0") &&
Usuario.get(“TipoAcesso”).equals("1")){
            // este usuario tem a visão e o acesso ao sistema da forma 1
      }
      ...
}
```

O código refatorado agora apresenta um nome adequado para o método. Como a técnica ainda permite que sejam feitas alterações pertinentes para a legibilidade do código, o nome das variáveis também foi modificado, o *array* indexado por números foi substituído por uma estrutura *Map* (que permite a indexação por texto, facilitando o entendimento conceitual dos valores que ali está contido) e inserido um comentário explicando as informações que cada valor representa. Percebe-se uma modificação considerável para elevar o código para o nível 2.

Em seguida, pela tabela, pode-se aplicar uma técnica GRH de nível 2, o *Consolidar Expressão Condicional*, que segundo Fowler (2004), sugere combinar uma sequência de testes condicionais em uma única expressão condicional e extraí-la. O Quadro 3 apresenta o código refatorado por esta técnica.

Desse modo, quando houver qualquer alteração nas condições em que a forma 1 de visão e acesso ao sistema deva ser apresentado, fica mais fácil fazê-lo com o código escrito nesta última forma. Por ser do grupo GRH, esta refatoração contribui para a melhoria do código, mantendo o mesmo no nível 2.

O próximo passo seria aplicar as técnicas *Converter Projeto Procedural em Objetos* e *Substituir Vetor por Objeto*, refatorações do grupo GRV, que contribui para que o código atinja o nível 3, demonstrado no Quadro 4.

Quadro 3 – código após a aplicação da técnica *Consolidar Expressão Condicional*

```
...
public void logar(Map Usuario){
      if (condicoesVisaoAcessoForma1(Usuario)){
            // este usuário tem a visão e o acesso ao sistema da forma 1
      }
      ...
}

public boolean condicoesVisaoAcessoForma1(Map Usuario){
      if (Usuario.get(“TipoUsuario”).equals("0") &&
Usuario.get(“TipoAcesso”).equals("1")){
            return true;
      }
      return false;
}
```

Essa alteração é, visivelmente, uma quebra de paradigma da programação. Foram identificados os possíveis objetos na estrutura anterior e criados com seus métodos, atributos e relacionamentos. Tinha-se um vetor com elementos que significavam coisas diferentes. Agora se tem um objeto com métodos de acesso e gravação onde armazenam informações relativas a um único objeto (Usuário) e suas subclasses (Cliente e Gerente).

Para a melhoria de um código de nível 3, podemos aplicar, conforme a tabela 1, a técnica *Subir Método na Hierarquia*. A Figura 2 ilustra a alteração sugerida pela técnica.

O método logar() era idêntico nas 2 subclasses de Usuário. Como o próprio Fowler (2004) comenta, essa duplicação pode causar problemas no futuro, quando uma alteração for feita em um deles e não for feita no outro.

E para finalizar a aplicação das técnicas, temos o *Substituir Construtor por um Método Fábrica*, que adequa o trecho de código do nível 3 ao nível 4, nível padronizado, neste caso sugerindo o uso do Padrão de Projeto *Factory*, como pode ser visto no Quadro 5.

A grande diferença dos dois códigos é que foi separado o destinatário da chamada de criação da classe do objeto criado. Dessa forma, quem controla a criação das subclasses de Usuário é a própria classe Usuário.

Quadro 4 – Alcançado nível 3 de maturidade do código

```
class Usuario{
      private String login;
      private String senha;
      private int tipo;
      public final int COMUM = 0;
      public final int ADMINISTRADOR = 1;
      public final int VIP = 2;

      Usuario (int pTipo){
            tipo = pTipo;
      }

      public boolean condicoesVisaoAcessoForma1(){
            if (this instanceof Cliente && this.tipo.Equals(Usuario.ADMINISTRADOR)){
                  return true;
            }
            return false;
      }

      public String getSenha(){...}
      public setSenha(String senha){...}
      public String etLogin(){...}
      public setLogin(String login){...}
}

class Cliente extends Usuario{
      public int logar(){
            if (condicoesVisaoAcessoForma1()){
                  // este usuário tem a visão e o acesso ao sistema da forma 1
            }
      ...
      }
}

class Gerente extends Usuario{
      public int logar(){
            if (condicoesVisaoAcessoForma1()){
                  // este usuário tem a visão e o acesso ao sistema da forma 1
            }
            ...
      }
}

Cliente cliente = new Cliente(Usuario.ADMINISTRADOR);
```

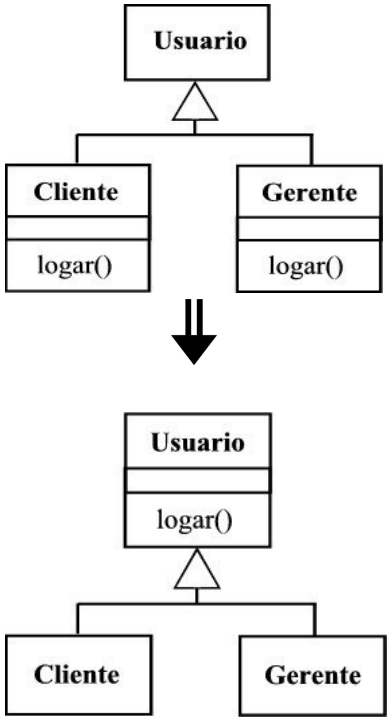

Figura 2 – Generalização sugerida pela técnica *Subir Método na Hierarquia*

7 Conclusão

Os projetos de *software* atualmente não contam com uma metodologia para o incremento da qualidade e maturidade da escrita de seus códigos e nem possuem um modelo que possa avaliá-los e determinar as práticas utilizadas para conseguir este objetivo.

A proposta deste artigo concentrou-se em criar uma metodologia de aplicação das técnicas de refatoração de código no mérito do aumento de qualidade e maturação. Como base, foi apresentado um modelo de qualidade e maturidade de código em 5 níveis, com as práticas que devem ser implementadas para atingir cada nível.

Pode-se concluir, através dos resultados de qualidade de escrita e aumento da maturidade obtidos nos

Quadro 5 – Refatoração aplicando o Padrão de Projeto *Factory* ao código-exemplo

```
class Usuario{      ...
      private Usuario (int pTipo){
            tipo = pTipo;
      }
      static Usuario criar (int tipo) {
            return new Usuario(tipo);
      }
... }

Cliente cliente = Cliente.criar(Usuario.ADMINISTRADOR);
```

exemplos de código, que a aplicação das técnicas de refatoração a partir de uma metodologia baseada no modelo MQMC é relevante, pois fornece ao analista/desenvolvedor, aos setores de desenvolvimento de sistemas do EB, uma sequência de ideias e ações direcionadas para alcançar uma melhoria mais eficiente na qualidade de código do seu projeto de *software*.

O modelo de qualidade e maturidade proposto por este artigo, além de ter a finalidade de direcionar a aplicabilidade da refatoração, avalia em que estágio um projeto de *software* está localizado, e define, de forma genérica, quais ações e processos devem ser implementados para elevar o seu nível. Este modelo contribui, na área de desenvolvimento de sistemas, para a padronização dos níveis do qual os códigos dos *softwares* encontram-se, e como motivação para adquirir qualidade e maturidade por meio de técnicas de refatoração já conhecidas.

Como sugestão para trabalhos futuros, é oportuno expandir a lista de técnicas de refatoração com as que atendam aos níveis mais altos do modelo de qualidade e maturidade do código, completando assim a metodologia e sua aplicabilidade.

Referências

CMMI. Wikipedia. Home Page. 2007. Disponível em: <http://pt.wikipedia.org/wiki/CMMI>. Acesso em: 27 jul. 2007.

FOWLER, Martin et al. Kent Beck; John Brant; William Opdyke; Don Roberts. **Refatoração**: aperfeiçoando o projeto de código existente. Porto Alegre: Bookman, 2004.

GAMMA, Erich et al. (Richard Helm, Ralph Johnson, John Vlissides.) **Padrões de projeto**: Soluções reutilizáveis de software orientado a objetos. Porto Alegre: Bookman, 2000.

JAVA. Wikipedia. Home Page. 2007. Disponível em: <http://pt.wikipedia.org/wiki/Java_%28linguagem_de_programa%C3%A7%C3%A3o%29>. Acesso em: 02 ago. 2007.

MPS.BR. Wikipedia. Home Page. 2007. Disponível em: <http://pt.wikipedia.org/wiki/MPS.BR>. Acesso em: 01 ago. 2007.