Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Departamento de Ciências da Computação

Luiz Felipe de Oliveira Mendes

Implementação em CONNECTA de Métricas para Estimativa de Amplitude da Propagação de Modificações Contratuais em Sistemas Orientados por Objetos

Monografia de Projeto Orientado em Computação II

Belo Horizonte – MG
2009 / 2º semestre

Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Departamento de Ciências da Computação

Implementação em CONNECTA de Métricas para Estimativa de Amplitude da Propagação de Modificações Contratuais em Sistemas Orientados por Objetos

por

Luiz Felipe de Oliveira Mendes

Monografia de Projeto Orientado em Computação II

Apresentado como requisito da disciplina de Projeto Orientado em Computação II do Curso de Bacharelado em Ciência da Computação da UFMG

Prof. Dra. Mariza Andrade da Silva Bigonha
Orientadora

Prof. Msa. Kecia Aline Marques Ferreira
Co-Orientadora

_____

Luiz Felipe de Oliveira Mendes - Aluno

_____

Mariza Andrade da Silva Bigonha - Orientadora

_____

Kecia Aline Marques Ferreira - Co-Orientadora

Belo Horizonte – MG

2009 / 2º semestre

Aos professores,
aos colegas de curso,
aos meus familiares,
dedico este trabalho.

Agradecimentos

Gostaria de agradecer a minha família e a minha namorada querida pelo apoio.
Agradeço também meus professores pelo conhecimento adquirido.
Agradeço ao Michael Jordan por ser um grande ídolo que me ensinou a nunca desistir
Finalmente gostaria de agradecer os grandes amigos que fiz no curso, amizades que me fizeram chegar até aqui, e que levarei para sempre comigo.

"I can accept failure, everyone fails at something. But I can’t accept not trying."

Michael Jordan

“Some people want it to happen, some wish it would happen, others make it happen.”

Michael Jordan

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Siglas

OO	Orientados por Objetos
MACSOO	Modelo de Avaliação de Conectividade em Sistemas Orientados por Objetos

Resumo

O custo total da produção de um software é altamente dependente de seu custo de manutenção. Estima-se que a manutenção pode chegar a 70% desse custo total.

Vários fatores contribuem para baixar o custo de manutenção de software, dentre eles destaca-se a conectividade, a coesão e o acoplamento. A conectividade é definida como o grau de intercomunicação entre módulos, o acoplamento é o grau em que um certo módulo do sistema se comunica com outro e a coesão tem haver com as responsabilidades internas de um módulo.

Tendo em vista que a conectividade é um fator de suma importância na manutenibilidade do sistema, o presente trabalho tem como objetivo implementar no software CONNECTA a métrica para predição de amplitude da propagação de modificações contratuais em softwares orientados a objeto, chamado de K3B, e fazer experimentos para calibração e validação do mesmo.

Palavras-chave: Métricas, custo, manutenibilidade, conectividade, orientação por objeto, estabilidade, K3B, coesão, acoplamento.

Abstract

The total cost of production of software is highly dependent on its cost of maintenance. Meyer [1] indicates that maintenance may reach 70 percent of the total cost.

Given that the connectivity is of paramount importance in the maintainability of the system, this paper aims to implement in the software CONNECTA a metric for predicting time of maintenance in object-orientedOO software, called K3B, and provide some experiments in order to calibrate and validated it.

Keywords: Metrics, maintainability , cost, prediction, connectivity, K3B.

1 INTRODUÇÃO

Um dos problemas mais importantes na produção de software é o custo e a maior parcela deste custo decorre da manutenção. Poder estimar o mais cedo possível o custo de manutenção de um software traduz-se na possibilidade de tomar medidas para reduzir o seu custo total.

A manutenibilidade de um software é a facilidade de se realizar manutenções nele. Vários fatores são determinantes para a manutenibilidade, entre eles a conectividade, o acoplamento e a coesão interna de módulos. A conectividade é o grau de intercomunicação entre os módulos de um sistema, o acoplamento é o grau em que um certo módulo do sistema se comunica com outro(Seção 2.3) e a coesão tem a ver com as responsabilidades internas de um módulo(Seção 2.2).

A comunidade produtora de software ainda não conta com meios confiáveis que lhe permita controlar o custo usando a manutenibilidade, muito embora existam vários softwares e modelos que tratam desse assunto[2][3][4][5][6]. Dentre eles destacam-se dois: CONNECTA[6] e o Modelo de Predição de Amplitude da Propagação de Modificações Contratuais de software orientado por objetos[2], por terem uma relação direta com o trabalho de pesquisa descrito neste texto.

CONNECTA é uma ferramenta para coleta de métricas em softwares escritos em Java cujo objetivo é medir a conectividade de um sistema OO. Para isso, CONNECTA utiliza o modelo MACSOO, Modelo de Avaliação de Conectividade em Sistemas Orientados por Objetos[7]. Este modelo indica quais métricas devem ser avaliadas, bem como quais aspectos devem ser melhorados, caso obtenha-se um valor não apropriado para o aspecto principal do software avaliado: a conectividade.

Na tentativa de solucionar o problema exposto, Ferreira et al.[2], em seu trabalho de doutorado, investiga métodos para estimar o custo de manutenção de softwares OO. Neste sentido é proposto um novo modelo, onde não só a conectividade, mas também a coesão e o acoplamento são considerados na manutenção de um sistema. A questão principal investigada pelo

modelo proposto está relacionada com o tempo e número de passos até a estabilização de um software quando um ou mais módulos sofrem manutenção que altera suas interfaces.

1.1 Objetivo, Justificativa e Motivação

Dentre os fatores de avaliação da qualidade de um software, destaca-se a manutenibilidade, a medida da facilidade de realizar sua manutenção. Esse trabalho de pesquisa considera a conectividade como o fator preponderante na avaliação da manutenibilidade e consequentemente no custo total do sistema, pois quanto maior o grau de conectividade de um software, mais rígida a sua estrutura, menor a manutenibilidade e maior o custo do sistema.

Resumidamente, no POCI[8] foi implementada e incorporada uma adaptação da métrica de estabilidade de Myers [9], foi realizado um estudo da métrica Índice de Manutenibilidade (MI)[10], também foi incorporada uma versão inicial da fórmula(K3B) do modelo matemático proposta por Ferreira em CONNECTA.

Dando continuidade a pesquisa e implementação desenvolvidos durante o POC I, este trabalho elenca os seguintes objetivos:

- Implementar na ferramenta CONNECTA a fórmula do modelo matemático K3B (Veja Seção 2.3) proposto por Ferreira[2]. Esta fórmula é uma adaptação daquela implementada no POC I com novos parâmetros introduzidos por questões de completeza.
- Fazer testes inicias com a fórmula para calibrar seus valores.
- Realizar experimentos para avaliar os resultados do modelo.

Tendo em vista os objetivos propostos, destacam-se como contribuições deste trabalho:

1. Evolução da ferramenta CONNECTA, que disponibiliza aos projetistas de software métricas que auxiliam na avaliação de um sistema ou classe.
2. A implementação do modelo, K3B, que dado o número de classes a serem modificadas reflete o número de passos de modificação necessários para que o software volte a estabilidade.

1.2 Organização do Texto

Esta monografia está organizada da seguinte forma:

- Seção 2: apresenta o principais conceitos necessários para o bom entendimento do que foi elaborado nesse projeto.
- Seção 3: apresenta a os procedimentos metodológicos utilizados para realização desse trabalho.
- Seção 4: descreve os resultados da implementação e dos experimentos realizados, além de analisar e discutir sobre os mesmos.
- Seção 5: apresenta as conclusões tiradas após o término do trabalho e descreve possíveis trabalhos futuros.

2 REFERENCIAL TEÓRICO

O objetivo dessa seção é apresentar a teoria necessária para entender do que se trata o trabalho. Nela serão explicados algumas definições e o funcionamento de alguns algoritmos utilizados nas implementações realizadas.

2.1 Programação Orientada por Objetos

O paradigma orientado por objetos, é um paradigma de análise, projeto e programação de sistemas de software baseado na composição e interação entre diversas unidades de software chamadas de objetos.

Os sistemas criados por esse paradigma são chamados de sistemas orientados por objeto, eles são compostos de módulos. No caso da linguagem Java, esse módulos são representados por classes que determinam o comportamento (definido nos métodos) e os estados possíveis (atributos) de seus objetos, assim como o relacionamento com outros objetos.

Um bom projeto de software deve definir módulos o mais independentes possível, que possam ser entendidos, implementados e alterados sem a necessidade de conhecer o conteúdo dos demais módulos e com o menor impacto possível sobre eles.

2.2 Métricas de Software

A medição sempre esteve presente na área da engenharia, isso também é válido na engenharia de software, que utiliza métricas de software que auxiliam no entendimento do comportamento e funcionamento de sistemas, avaliam metas e critérios de aceitação, controlam processos e serviços, além de prever valores e comportamentos importantes.

As métricas de software são uma forma de quantificar e qualificar características e comportamentos de sistemas, classes e programas. São utilizadas com objetivos variados, como:

- Indicar a qualidade do produto.
- Avaliar a produtividade dos que desenvolvem o produto.
- Determinar os benefícios derivados de novos métodos e ferramentas de engenharia de software.
- Formar uma base para as estimativas.
- Ajudar na justificativa de aquisição de novas ferramentas ou de treinamentos adicionais.

2.3 Acoplamento

Acoplamento é o grau em que um certo módulo do sistema se comunica com outro.Se existe uma comunicação entre dois módulos então esses módulos estão acoplados. Quanto mais forte for essa comunicação, maior o nível de acoplamento e dependência desses módulos. Um sistema em que seus módulos são muito acoplados é mais difícil de se manter, por isso é desejável diminuir o acoplamento médio de um sistema.

2.3.1 Classificação de Acoplamento

Myers[9] propôs a seguinte classificação para o acoplamento entre dois módulos.

Acoplamento por Conteúdo

É o tipo mais forte de acoplamento entre dois módulos. Este tipo de acoplamento existe quando um módulo faz referência direta ao conteúdo do outro módulo. O acesso é feito a elementos não exportados do módulo.

Acoplamento por Dado Comum

Esse tipo de acoplamento acontece quando um grupo de módulos compartilham uma área de estrutura de dados comum e é facultado a cada um deles uma interpretação específica da área de dados.

Acoplamento Externo

Existe este tipo de acoplamento em grupo de módulos que referenciam um mesmo termo declarado externamente. Este tipo de acoplamento é muito próximo do acoplamento

por dado comum. A diferença básica está na unidade compartilhada entre os módulos: no acoplamento externo os módulos usam itens de dados individuais declarados em uma área de dados comum e não na área comum completa, como ocorre no acoplamento por dado comum.

Acoplamento de Controle

O acoplamento entre dois módulos é desse tipo quando um módulo passa um parâmetro que determina diretamente o fluxo de execução do outro módulo.

Acoplamento de Referência ou *Stamp*

Ocorre quando dois módulos compartilham uma área de dados não declarada globalmente. Isso ocorre quando a comunicação entre dois módulos é realizada por meio de chamada de rotinas com passagem de parâmetro por referência. Este tipo de acoplamento é menos grave do que os acoplamentos por dado comum e externo, porém ainda apresenta o problema de efeito colateral de alterações sobre os dados compartilhados.

Acoplamento por Informação

Dois módulos estão acoplados por informação se a comunicação entre eles for feita por chamada de rotina com passagem de parâmetros por valor, desde que tais parâmetros não sejam elementos de controle.

Desacoplado

Dois módulos são desacoplados quando não existe comunicação entre eles.

2.4 Coesão

A coesão mede quão relacionadas ou focadas estão as responsabilidades de um módulo, isto é, é a medida do relacionamento dos elementos internos de um módulo. Módulos com alta coesão tendem a ser preferíveis porque a alta coesão está associada com várias características desejáveis de software, incluindo robustez, confiabilidade, reusabilidade e compreensibilidade. Já a baixa coesão está associada com características indesejáveis, como sendo difícil de manter, difícil de testar, de difícil reutilização, e mesmo difícil de compreender.

2.4.1 Classificação de Coesão

Myers[9] propõe a seguinte classificação para a coesão interna de um módulo.

Coesão Coincidental

Há nenhuma (ou pouca) relação construtiva entre os elementos de um módulo. Em software orientado por objetos isso corresponde a:

- Um objeto não representa nenhum conceito OO.
- Uma coleção de código comumente usado e herdado via herança, provavelmente múltipla .

Coesão Lógica

Um módulo possui um conjunto de funções relacionadas, uma das quais é escolhida por meio de um parâmetro ao ser chamado. Está associada ao acoplamento de controle.

Coesão Clássica(temporal)

Elementos estão agrupados no mesmo módulo porque são processados no mesmo intervalo de tempo. Exemplos comuns:

- Método de inicialização que provê valores padrão para um conjunto de ações relacionadas temporalmente entre si.
- Método de finalização que realiza um conjunto de ações relacionadas ao encerramento do programa, por exemplo, fechamento de arquivo e conexões com banco de dados.

Coesão Procedural

Associa elementos de acordo com seus relacionamentos procedurais ou algorítmicos. Um módulo procedural depende muito da aplicação sendo tratada. Não se consegue entender o módulo sem entender o programa e as condições que existem quando o módulo é chamado.

Coesão Comunicacional

Todas as operações de um módulo operam no mesmo conjunto de dados e/ou produzem o mesmo tipo de dado de saída.

Coesão Funcional

Um módulo com esse tipo de coesão, é caracterizado por executar uma única função bem definida.

Coesão Informacional

Um módulo com esse tipo de coesão implementa um tipo abstrato de dados.

2.5 Coesão de Interesses - CoIn

Para entender a métrica CoIn é necessário conhecer algumas definições, como: interesse, relacionamento e propriedade transitiva.

- **Interesse:** conjuntos disjuntos formados por todos os métodos relacionados entre si de uma classe compõem os interesses dessa classe. Ou seja, um interesse é um conjunto de métodos relacionados.
- **Relacionamento:** dois métodos de uma classe C estão relacionados se utilizam pelo menos um atributo da classe C em comum ou pelo menos um método da classe C em comum.
- **Propriedade Transitiva:** se um método *a* está relacionado a um método *b* pela definição de relacionamento anterior, e *b* está relacionado a um método *c*, então o método *a* está também relacionado a *c*.

2.5.1 Definição da CoIn

A métrica coesão contratual baseia-se no número de interesses que uma classe implementa. Seu valor é então dado por 1/total de interesses da classe . Se uma classe implementa dez interesses, por exemplo, o valor assumido pela métrica é 0.1. Da mesma forma, em uma classe que implementa apenas um interesse a métrica assume o valor 1. Ou seja, quanto mais

próximo de 1 é o valor da métrica, melhor a sua Coesão de Interesse. Quanto mais próximo de 0, pior. A métrica de interesse é definida a seguir:

```
Seja C o conjunto de conjuntos disjuntos formados por métodos com
relacionamento entre si. Seja o número de interesses coesos N = |C|.
  Coesao = 1/N, se N > 0
  Coesao = 0, caso contrário
```

2.6 Fator de Acoplamento - COF

Utilizada para avaliar o acoplamento, a métrica COF foi introduzida por Abreu e Carapuça [11] e é utilizada no CONNECTA para o cálculo da K3B.

Um software fortemente conectado é normalmente de difícil manutenção, pois possui uma estrutura rígida e seus módulos possuem um alto grau de dependência entre eles. Sendo assim, a medida da COF se mostra de suma importância para medir a qualidade e facilidade e manutenção de um sistema. Esta métrica utiliza o conceito de relação cliente-servidor entre os módulos do software em questão.

2.6.1 Relação Cliente-Servidor

Se uma classe A referencia pelo menos um membro de uma classe B, sendo esse membro uma variável de instância ou um método, então a classe A é um cliente da classe servidora B. Essa relação de cliente-servidor caracteriza uma conexão entre as classes A e B. A Figura 2.1 ilustra essa representação de conexões entre classes de um sistema, nesse caso, A é um cliente de B e de C e B é um cliente de A.

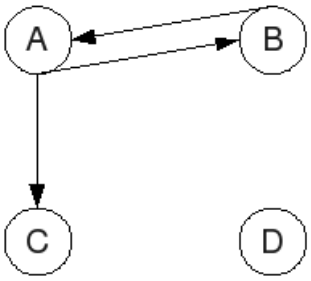

Figura 2.1: Relacionamento Cliente-Servidor entre classes

2.6.2 Cálculo da COF

Para calcular a métrica COF é feita uma simples operação que leva em consideração que o número máximo de conexões em um sistema com n módulos é $n^2 - n$.

A métrica COF é a razão entre o número total de conexões existentes no sistema e o número máximo de conexões, por exemplo: se um sistema possui 4 classes, e 3 conexões então o COF desse sistema é $3/4^2 - 4 = 0,25$. Sendo assim, um software totalmente conectado tem COF = 1.

2.7 CONNECTA

CONNECTA[6] é uma ferramenta de coleta de métricas implementada em Java que viabiliza a aplicação das métricas propostas no modelo MACSOO para softwares desenvolvidos em Java.

A idéia central do Modelo de Avaliação de Conectividade em Sistemas Orientados por Objetos, ou MACSOO[7], é prover um conjunto de heurísticas com objetivo de auxiliar o projetista na tomada de decisão durante a contrução de software na tentativa de reduzir a sua conectividade. MACSOO foi concebido da seguinte forma:

- Identificaram-se os principais fatores de impacto na conectividade, como: ocultamento de informação, acomplamento entre classes, coesão interna de classes e profundidade da árvores de herança.
- Levantaram-se as métricas que podem ser utilizadas para medir cada um destes fatores.
- Com foco na redução da conectividade de software, foram propostas heurísticas que tem como elemento principal a avaliação dos fatores impactantes na conectividade por meio de métricas.

O CONNECTA recebe softwares em Java e utilizando os bytecodes dos mesmos, faz cálculos e aplica as métricas já discutidas, mostrando os resultados por meio de sua interface gráfica. O CONNECTA está atualmente sofrendo atualizações e modificações por uma equipe de desenvolvimento da qual faço parte.

2.8 Conclusão

O objetivo dessa seção foi prover alguns conceitos importants para o entendimento do trabalho desenvolvido e relatado nas próximas seções.

3 METODOLOGIA

O trabalho proposto é de natureza tecnológica, com uma pequena parte teórica e uma parte de implementação. A parte teórica engloba o entendimento do modelo teórico e das métricas envolvidas, tornando possível a imaplementação de uma ferramenta para auxiliar o projetista no desenvolvimento do software.

3.1 Procedimentos Metodológicos

Afim de atingir os objetivos propostos nesse trabalho de pesquisa, inicialmente fez-se uma implementação apenas do modelo matemático(K3B) proposto por Ferreira. Após testes iniciais e uma leve calibração nos valores utilizados pelo modelo, o mesmo foi implementado no software CONNECTA.

Para validar a implementação, foram feitos experimentos, que consistiram resumidamente em realizar modificações no próprio CONNECTA salvando as métricas obtidas, o tempo e o número de passos realizados.

3.1.1 Experimentos

Para realizar os experimentos de avaliação da métrica K3B, foi necessário identificar necessidades de modificação no software CONNECTA. Para isso, utilizou-se a métrica CoIn 2.5 como indicador, isto é, classes que receberam um valor indesejado para essa métrica foram verificadas e aquelas que apresentaram possíveis melhorias foram acrescentadas a uma lista de modificações. Boa parte dessas necessidades foram identificadas desta forma, mas outros três casos foram identificados somente via análise direta do código.

Além das modificações de reestruturação, outros tipos de modificações foram aplicados, a saber: na interface e em padrão de programação. Todas essas modificações foram escolhidas por meio da observação e análise do software.

Dada as modificações encontradas dividiu-se as mesmas em 3 grupos:

- Interface
- Padrão
- Reestruturação

Para cada um desses grupos, foi feito uma tabela que segue o seguinte padrão:

Nº	Alteração	Nº Classes	Passos	Tempo	KB3	Alpha	Beta	COF	Obs.	Versão Gerada

Tabela 3.1: Tabela Padrão dos Experimentos

sendo que:

- **Nº** - Identificador da modificação.
- **Alteração** - Descrição da Modificação.
- **Classes** - Número de classes que serão modificadas.
- **Passos** - Número de passos realizados para efetuar a modificação.
- **Tempo** - Tempo em minutos utilizados para realizar a modificação.
- **KB3,Alpha,Beta e COF** - Métricas da nova versão de CONNECTA.
- **Observações** - Observações sobre a modificação realizada.
- **Versão Gerada** - Número da versão gerada.

Para cada uma dessas alterações devem ser realizados os seguintes passos:

1. Antes de iniciar a alteração, coletar as métricas do software e gravar.
2. Estimar o número de classes que sofrerão alterações.
3. Medir o tempo (em minutos) necessários para a conclusão da alteração.
4. Contar o número de passos necessários para a conclusão da tarefa.
5. Registrar, se necessário, as observações sobre a alteração.

6. Coletar e gravar as métricas do software após a conclusão da alteração.

Os resultados obtidos com a metodologia proposta são comparados aos valores de KB3. A hipótese a ser investigada é que KB3 pode estimar o tempo ou o número de passos necessários para realização de alterações em software.

3.2 Conclusão

Esta seção expôs a metodologia utilizada nesse POCII. A Seção 4 mostra os resultados da aplicação dessa metodologia e discute sobre a implementação do modelo K3B e dos resultados obtidos no experimento.

4 RESULTADOS E DISCUSSÃO

Esta seção trata dos resultados do trabalho do POCII incluindo a discussão sobre as implementações, experimentos e as informações obtidas nos estudos realizados.

4.1 Implementação do Modelo Matemático K3B em CONNECTA

Partindo do pressuposto que a conectividade é o fator mais importante no custo de manutenção de um software, Ferreira et al. [6] propoem uma maneira de investigar o esforço de manutenção em sofware orientados por objeto. Para auxiliar nesse processo, Ferreira et al. definiram um modelo para predição de amplitude da propagação de modificações contratuais em softwares orientados a objeto, chamado K3B, baseado na conectividade, cujo objetivo é prever, por meio das métricas coletadas, o tempo de estabilização do sistema. Um software é considerado estável, quando todas as alterações decorrentes da primeira alteração foram concluídas.

4.1.1 A Fórmula Matemática K3B

A definição de K3B faz uso de diversas métricas coletadas pelo CONNECTA, como:

- COF ϕ - Fator de Acoplamento do sistema, calculado dividindo-se o número máximo de conexões possíveis do sistema pelo número existente.
- Média dos Acoplamentos do Sistema α.
- Média da Coesão de Interesse(CoIn) β, é uma métrica de classe, que quantifica a coesão do sistema dado o número de interesses dessa classe.

Dada a complexidade da fórmula proposta por Ferreira[2], e sua difícil incorporação no sistema CONNECTA, decidimos por implementar uma simplificação da fórmula, ilustrada na Tabela 4.1, que parecia, em primeiro momento, corresponder ao resultado do modelo completo.

$n!\frac{\alpha^{n-1}\phi^{n-1}}{\beta^{n-1}}$

Tabela 4.1: Fórmula do modelo matemático

sendo:

- n o número de módulos do sistema
- α, o acoplamento médio.
- ϕ, o grau de acoplamento, isto é, COF do sistema.
- β, a coesão interna média, isto é, CoIn média do sistema.

Os testes iniciais com essa fórmula mostraram que os resultados da simplificação não estavam próximos do resultado do modelo completo como era esperado, sendo assim, foi necessário modificar a fórmula de modo que ela fosse implementada sem simplificaçõe s[2].

$$E(n,1) = 1 + 2(n-1)(n-2)!\,\frac{\alpha^{n-1}\phi^{n-1}}{\beta^{n-1}} + \sum_{k=1}^{k=n-2} \frac{2(n-1)(n-2)!}{k!}\,\frac{\alpha^{n-k-1}\phi^{n-k-1}}{\beta^{n-k-1}} \tag{4.1}$$

$$E(n,i) = 1 + E(n,i-1) + \sum_{k=i-1}^{k=n-2} 2(n-i)\,\frac{(n-i-1)!}{(k-i+1)!}\,\frac{\alpha^{n-k-1}\phi^{n-k-1}}{\beta^{n-k-1}} \tag{4.2}$$

Problemas Encontrados

A fórmula do modelo completo foi implementada inicialmente fora do software CONNECTA, e alguns testes iniciais revelaram problemas. Na fórmula, existe um calculo de fatorial:

$$2(n-1)(n-2)! \tag{4.3}$$

Como esse fatorial pode produzir números enormes, isso gera problemas de armazenamento em seu cálculo. A fórmula possui também números pequenos elevados a uma grande potência, como pode ser visto em 4.4.

$$\alpha^{n-k-1}\phi^{n-k-1}\beta^{n-k-1} \tag{4.4}$$

O cálculo dessas potências para grandes valores de n produziam números que tendiam a 0, e não foi possível armazenar tais números via variáveis de ponto flutuante.

Para contornar os problemas descritos, utilizou-se transformações matemáticas, por exemplo, logarítimos e a função LnGamma[12] .

Para um inteiro positivo a, LnGamma retorna o logarítimo do produto $1*2*3*...*(a-1) = ln((a-1)!)$. LnGamma é utilizada para calcular fatoriais quando os resultados são grandes demais para serem representados por floats.

Para o cáculo do LnGamma utilizou-se uma biblioteca livre chamada WEKA [13].

Chegou-se então a uma fórmula final, que é basicamente uma transformação da fórmula mostrada em 4.1 e 4.2, mas utilizando LnGamma no lugar dos fatoriais e logarítimos nos outros termos. A forma final pode ser encontrada em [2].

4.2 Experimentos

A metodologia para realização dos experimentos foi especificada na Seção 3 deste documento.

4.2.1 Modificações

Modificações de Reestruturação

A avaliação das classes do software CONNECTA utilizando-se da métrica CoIn como indicador gerou a Tabela 4.2.

Nº	Classe	CoIn	Conexões	Avaliação	Avaliação da CoIn
1	frmMain	0.2	1	É a classe principal. Possui vários métodos com um propósito em comum: acionar outros objetos do software. Esta característica não é capturada pela métrica de coesão.	Positivo
2	frmSelectFiles	0.5	1	A classe implementa pelo menos dois contratos: interface com usuário e implementação de regras de negócios	Positivo
3	ModelMetrics	0.25	6	Trata-se de uma classe do tipo model, que possui somente métodos do tipo get e set.	Positivo
4	ConnectionPath	0.5	1	Há um método printPath, que não faz nada. Isso gerou o valor 0,5.	Positivo
5	ReadHistory	0.5	1	Trata-se de uma classe utilitária. Possui somente um método estático que recebe o nome de um arquivo e retorna um objeto do tipo ModelHistory contendo os dados do arquivo. O valor 0,5 provavelmente considera o método construtor default.	Negativo
6	frmResultSystem	0.5	7	É uma classe similar à principal: aciona outros objetos do software, a depender da opção do usuário. Esta característica não é capturada pela métrica de coesão. O alto valor de conexões deve-se à existência de classes auxiliares geradas pelo NetBeans (aquelas que possuem $ no nome. Retirar essas classes da análise?).	Positivo
7	frmCadastrarEscolherProjeto	0.5	2	A classe implementa três contratos: interface com usuário (tela e mensagens), regras de negócios (teste) e persitência (conexão com banco, comandos SQL). A métrica capturou somente dois contratos, pois não avalia coesão de método. Há um só método que implementa os serviços da classe.	Negativo
8	ModelHistory	0.33	2	Trata-se de uma classe do tipo model, que possui somente métodos do tipo get e set.	Positivo
9	GraphSystem	0.2	5	A classe mostra o grafo que representa o sistema avaliado. Possui atributos e métodos que deveriam estar em SystemCollector.	Positivo
10	SaveHistory	0.5	1	Trata-se de uma classe utilitária. Possui somente um método estático que recebe o nome de um arquivo e retorna um objeto do tipo ModelHistory contendo os dados do arquivo.	Negativo
11	ModelClassHistory	0.5		Trata-se de uma classe do tipo model, que possui somente métodos do tipo get e set.	Positivo
12	frmDetalhesClasseBD	1		Embora a métrica de coesão tenha indicado valor 1, esta classe implementa dois contratos: interface com usuário e seleção de dados em banco de dados. Boa parte das rotinas é implementada dentro de um único método.	Negativo
13	frmDetalhesProjetoBD	1		Idem	Negativo
14	frmEscolherClasses	1		Idem	Negativo

Tabela 4.2: Avaliação das Classes de CONNECTA

Modificações de Interface

A observação do CONNECTA em utilização gerou as modificações de interface presentes na Tabela 4.3.

Nº	Descrição
1	Na tela Métricas do Sistema, incluir os valores calculados de alpha e beta
2	Na tela Detalhes do Sistema, no grupo Métricas do Sistema, diminuir a distância entre os rótulos e os textos
3	Na tela Detalhes do Sistema, no grupo Métricas do Sistema, exibir os valores de alpha e beta.
4	Na tela Detalhes do Sistema, ordenar os dados por quantidade de conexões.
5	Na tela Detalhes da Classe, no grupo Métricas do Sistema, incluir o valor de K3B.
6	Na tela Detalhes da Classe, diminuir o espaços entre os rótulos e os textos.
7	Na tela Detalhes da Conexão, mudar a mensagem que aparece quando não há conexões por informação. Está aparecendo: "Nenhum ponto de conexão por stamp". Mudar para: "Nenhuma conexão por informação".
8	Na tela Detalhes de Conexão, mudar as mensagens que aparecem informando que "não há ponto de conexão por... "para "não há conexão por ..."
9	Ao gravar os dados em arquivo, gravar também os valores de alpha e beta.
10	Na tela Historico (Consulta por Arquivo), mostrar os dados de alpha e beta.
11	Na classe Pegador, em vez de exibir mensagem de erro no Console, exibir com uma interface Java.
12	Na tela Historico (Consulta por Arquivo), mostrar o número de conexões do sistema.

Tabela 4.3: Modificações de Interface

Modificações de Padrão

Observando o software CONNECTA durante o desenvolvimento, percebeu-se uma inconsistência entre suas classes e métodos. A maioria das classes e métodos foram criadas com nomes em inglês, mas algumas poucas classes e métodos estavam com nomes em português, logo a modificação de padrão resultante é a presenta na Tabela 4.4

Nº	Descrição
1	Nomes de classes e métodos que estão em português, devem ser modificadas para língua inglesa.

Tabela 4.4: Modificação de Padrão

4.2.2 Resultados dos Experimentos

Os experimentos foram realizados, utilizando as modificações propostas na Seção 4.2 e utilizando a metodologia proposta na Seção 3.

Os resultados desses experimentos serão expostos utilizando as Tabelas citadas na Seção 4.4, para melhor visualização desses resultados as Tabelas foram inseridas nesse documento no formato paisagem (*landscape*).

Interface

As modificações de interfaces expostas na Seção 4.2.1, foram realizadas e o resultados das mesmas estão expostas nas Tabelas 4.5 e 4.6 .

Padrão

A modificação de padrão exposta na Seção 4.2.1, foi realizada e o resultado da mesma esta exposta na Tabela 4.7.

Reestruturação

As modificações de reestruturação expostas na Seção 4.2.1, foram realizadas e o resultados das mesmas estão expostas nas Tabelas 4.8 e 4.8.

Nas Tabelas 4.8 e 4.8 aparece uma nova coluna, entitulada de *Causa* que é uma referência a Tabela 4.2 que são as causas de cada modificação.

N^o	Descrição	Classes alteradas	N^o de passos	Tempo	K3B	α	β	ϕ	Observações	Versão
1	Na tela Métricas do Sistema, incluir os valores calculados de alpha e beta	1	1	10	2.5381	0.1879	0.8797	0.031	1)Modifiquei pela interface gráfica do NetBeans. E inseri o código para vincular as variáveis na classe : frmResultSystem.java. 2) Foi mais fácil do que o 3 pois já tinha feito boa parte das modificações.	v04
2	Na tela Detalhes do Sistema, no grupo Métricas do Sistema, diminuir a distância entre os labels e os textos	1	1	5	2.5381	0.1879	0.8797	0.031	1-Modifiquei a classe frmSystemDetail.java por meio da interface gráfica	v03
3	Na tela Detalhes do Sistema, no grupo Métricas do Sistema, exibir os valores de alpha e beta.	3	3	15	2.5381	0.1879	0.8797	0.031	Usei a Versao01 do connecta. || Passo 1 - Adicionar em frmsystemdetail os labels Passo 2 - Em ModelMetrics System adicionar variaveis Passo 3 - Passar o valor de alpha e beta em System Collector	v02
4	Na tela Detalhes do Sistema, ordenar os dados por quantidade de conexões	1	1	30	2.553	0.1882	0.876	0.03	1- Colocar na classe uma função para Jtable. Gastei muito tempo mexendo tentando arrumar a ordem, mas nao consegui	v12
5	Na tela Detalhes da Classe, no grupo Métricas do Sistema, incluir o valor de K3B	1	1	10	2.553	0.1882	0.876	0.03	1 Por algum motivo fazer esse mudança criou uma classe a mais nos sistema.	v06
6	Na tela Detalhes da Classe, diminuir o espaços entre os labels e os textos.	1	1	9	2.5381	0.1879	0.8797	0.031	Modifiquei apenas a classe frmClassDetail.java. Via interface do Netbeans, tive problemas com alinhamento.	v05
7	Na tela Detalhes da Conexão, mudar a mensagem que aparece quando não conexões por informação. Está aparecendo: "Nenhum ponto de conexão por stamp". Mudar para: "Nenhuma conexão por informação".	1	1	1	2.553	0.1882	0.876	0.03	Modifiquei a msg	v10

Tabela 4.5: Resultados das modificações de Interface

N^o	**Descrição**	**Classes alteradas**	N^o **de passos**	**Tempo**	**K3B**	α	β	ϕ	**Observações**	**Versão**
9	Ao gravar os dados em arquivo, gravar também os valores de alpha e beta.	2	2	8	2.553	0.1882	0.876	0.03	1) Colocar na classe de historico 2) Setar o valor no resultsystem	v08
10	Na tela Historico (Consulta por Arquivo), mostrar os dados de alpha e beta.	1	1	5	2.553	0.1882	0.876	0.03	1- Colocar na interface do netbeans	v09
11	Na classe Pegador, em vez de exibir mensagem de erro no Console, exibir com uma interface Java.	1	1	7	2.553	0.1882	0.876	0.03	Alterar a classe pegador acrescentando joptionpane com mensagem de erro	v11
12	Na tela Historico (Consulta por Arquivo), mostrar o número de conexões do sistema	3	4	23	2.553	0.1882	0.876	0.03	1) Modifiquei a Classe para aparecer 2) Modifiquei a classe ModelHistory.java colocando a variavel totalconnections 3) Coloquei para ligar a variavel recem criada no ModelHistory 4) Voltei a classe de interface para ligar a variavel criada no history	v07
Média		1.3	1.4	10.8	2.54853	0.18811	0.87711	0.0303		

Tabela 4.6: Segunda Tabela de resultados das modificações de Interface

N^o	**Descrição**	**Classes alteradas**	**N^o de passos**	**Tempo**	**K3B**	α	β	ϕ	**Observações**	**Versão**
1	Nomes de classes e métodos devem aparecer em inglês.	7	7	95	2.553	0.1882	0.876	0.03	IMPORTANTE- Modifiquei todas classes utilizando a refatoração do NetBeans, sendo assim, o número de passos ficou MASCARADO.	v13

Tabela 4.7: Resultados da modificação de Padrão

N^o	Causa	Descrição	Classes alteradas	N^o de passos	Tempo	K3B	α	β	ϕ	Observações	Versão
1	Geral	Criar três pacotes: view (interface), control (controle), model(modelo). Distribuir as classes nesses pacotes, dependendo de sua característica.	?	4	38	2.553	0.1882	0.876	0.03	1) Refatorei meu projeto, pois estava uma bagunca 2) Adicionei dentro do pacote connecta o pacote connecta.Interface e adicionei os arquivos frm 3) Adicionei dentro do pacote connecta o pacote connecta.Model e adicionei os arquivos 4) Adicionei dentro do pacote connecta o pacote connecta.Control e e adicionei os arquivos restantes O resultado final ficou assim: http://i33.tinypic.com/21kxykp.jpg	v14
2	1	Criar uma classe de controle para coleta de métricas e mover para essa classe os métodos da classe frmSelectFiles que implementam regras de negócio.	2	2	15	2.543	0.189	0.876	0.03	Troquei a funcao : computeCoupling(sys) do frmselectfiles para SystemCollector	v15
3	7	Criar uma classe de controle para gravação de dados e uma classe de persistência. Mover para cada classe suas respectivas responsabilidades.	3	24	195	para i=1 ->2.5782 para i=3 ->7,618	0.1878	0.8596	0.029	Criei uma classe que cria as sqls - chamada DbQuerys.java Retirei da classe frmCadastrarEscolherProjeto que mudou de nome para frmSubscribeChooseProject, todas SQLs. Falta criar a parte de controle. A classe utilizava varias vezes a mesma SQl, por isso a criação da classe DbQuerys foi algo bom. Pois ao inves de reescrever a SQL é so reutilizar a funcao pronta. Caso a SQL mude por alguma alteração no banco, precisa apenas mudar a aclass DbQuerys. Criei uma classe de controlle chamada SubscribeChooseProjectController que funciona como intermediario entre a camada de de interface e a camada de persistência	v16
4	9	Mover os atributos (e respectivos métodos get e set) para a classe SystemCollector: k3b, sumCouplings, sumCoCo.	3	5	21	para i =1 ->2.5504 para i =3 ->7,537	0.1878	0.8684	0.029	1) Retirei os métodos e variáveis da class GraphSystem 2) Coloquei os métodos e variáveis na class SystemCollector 3) Retirei uma chamada ao increaseCoCo na frmSelectFiles 4) Criei a funcao que calcula o SumCoco 5) Modifiquei chamadas aos métodos modificados	v17
5	12	Utilizar a classe de persistência criada na alteração 3 desta lista, movendo para ela as rotinas de seleção de dados no BD.	3	4	19	para i=1 ->2.547 para i=3 ->7,528	0.1879	0.8702	0.028	1) Retirei da classe todas sqls 2) Coloquei uma funcao com sql chamada no DbQuerys 3) Criei uma classe de controle e coloquei nela uma função chamada para recuperar dados 4) Coloquei a funcao acima na classe frm	v18

Tabela 4.8: Resultados das modificações de Restruturação

N^o	**Causa**	**Descrição**	**Classes alteradas**	N^o **de passos**	**Tempo**	**K3B**	α	β	ϕ	**Observações**	**Versão**
6	13	Utilizar a classe de persitência criada na alteração 3 desta lista, movendo para ela as rotinas de seleção de dados no BD.	3	9	15	para i =1 ->2.5441 para i=3 ->7,522	0.1881	0.8717	0.028	1- Transferi para DBQuery Sql 2)Criei funcao no controller. 3)Coloquei na classe de interface. 4) Transferi para DBQuery Sql. 5)Criei uma funcao no controller. 6)Coloquei na classe de interface. 7) Transferi para DBQuery Sql. 8)Criei funcao no controller. 9)Coloquei na classe de interface.	v19
7	14	Utilizar a classe de persitência criada na alteração 3 desta lista, movendo para ela as rotinas de seleção de dados no BD.	3	13	22	para i=1 ->2.5783 para i=3 ->7,623	0.1884	0.8735	0.028	1) Transferi para DBQuery Sql. 2)Criei funcao no controller. 3)Coloquei na classe de interface. 4) Transferi para DBQuery Sql. 5)Criei funcao no controller. 6)Coloquei na classe de interface. 7) Transferi para DBQuery Sql. 8)Criei funcao no controller. 9)Coloquei na classe de interface. 10)Criei funcao no controller. 11)Coloquei na classe de interface. 12)Criei funcao no controller. 13)Coloquei na classe de interface. Não tive que criar algumas sqls que ja existiam	v20
Média			2.83	8.14	46.42	2.548	0.188	0.870	0.0288		

Tabela 4.9: Segunda Tabela de resultados das modificações de Restruturação

4.3 Análise dos Resultados

Os experimentos realizados nesse trabalho, ajudaram a tirar algumas conclusões, primeiramente vou dissertar sobre as Tabelas 4.5 e 4.6 que mostram os resultados das modificações na interface.

O número de passos foi no geral menor do que esperado pelo resultado de K3B, mas isto pode ser explicado, devido ao fato de que as mudanças da interface são no geral mais simples e que grande parte das mudanças foram facilitadas pelo uso da interface gráfica do software NetBeans, muitas vezes mascarando o resultado.

A Tabela 4.7 mostra a modificação de padrão sugerida para o CONNECTA, que no caso foi modificar nome de classes, métodos e variáveis para língua inglesa. Antes da modificação o CONNECTA possuia 7 classes com nomes, métodos e variáveis em português.

A modificação de padrão alterou pelo menos 7 classes do sistema, que são aquelas que estavam com nomes, métodos e variáveis em português, mas novamente um facilitador do software NetBeans mascarou os resultados do experimento. A função de refatoração desse software modifica o nome de classes, métodos e atributos automaticamente em todo sistema. Um ponto interessante sobre essa modificação foi que mesmo utilizando esse facilitador essa mudança de padrão levou 95 minutos para ser realizada, mais uma vez mostrando a grande quantidade de tempo que a manutenção de um software necessita.

As Tabelas 4.8 e 4.9 mostraram os resultados das principais modificações que foram as de reestruturação do CONNECTA.

Na modificação de número 1 presente na Tabela 4.8 houve uma dúvida quanto a quantidade de classes alteradas, pois nenhuma linha de código fois escrita em nenhum módulo do sistema, mas todas as classes foram de certa forma modificadas, pois foram colocadas em pacotes diferentes. Para essa modificação foi novamente utilizado um facilitador do software NetBeans, que cria automaticamente pacotes e utilizando as funções de clicar e carregar, pode-se mudar qualquer módulo de pacote.

Na modificação de número 3 presente na Tabela 4.8 houve uma peculiaridade em relação ao número de passos, pois 3 classes foram modificadas e foram necessários 24 passos para se completar essa modificação, sendo que K3B previu que iam ser necessários em torno de 7,6 passos. Algo que deve ser levado em consideração é que K3B não prevê o número de passos para cada modificação e sim o número médio de passos, então é de se esperar que algumas modificações necessitem de um número maior de passos do que o previsto pro K3B. Outra observação sobre a modificação de número 3, ela gastou 195 minutos, que é um tempo

considerável para reestruturação de apenas 1 módulo, isto ocorreu pois a classe foi programada sem pensar em reaproveitamento de código, sem lembrar de dividir por contratos e módulos.

Observando as médias presentes na Tabela 4.9 pode-se perceber que o número médio de classes modificados foi de 2,83 e o número médio de passos foi 8.14. Considerando que a previsão do modelo K3B para o sistema CONNECTA tendo 3 classes sendo modificadas é de 7,62 passos, percebe-se uma clara proximidade entre o resultado de K3B e os resultados dos experimentos.

5 CONCLUSÕES E TRABALHOS FUTUROS

Conhecer e prever o custo de produção de um software auxilia na diminuição de prejuízos, tempo e esforço. Sabendo-se que boa parte desse custo se origina da manutenção e que a manutenibilidade é a propriedade que indica o nível de facilidade em manter um software, então, investir na melhoria da manutenibilidade engendra em uma diminuição de custos.

Tendo em vista que nos dias atuais a maior parte dos sistemas produzidos seguem a orientação por objeto, estimar a amplitude da propagação de modificações contratuais nesses sistemas torna-se algo muito valioso, por esse motivo Ferreira et al. criaram o modelo K3B.

Nesse trabalho foi implementado o modelo K3B em CONNECTA que tem como objetivo prever o número de passos necessários para estabilização de um sistema após uma modificação em 1 ou mais módulos do mesmo. Com objetivo de observar como o modelo K3B se comporta em um situação real, foram feitos experimentos no próprio CONNECTA. Esses experimentos consistiram basicamente em realizar modificações de interface, padrão e reestruturação nas classes do CONNECTA e gravar os resultados das métricas e o número de passos necessários para estabilização em cada uma dessas modificações.

Os resultados desses experimentos mostraram uma proximidade entre os resultados do modelo K3B e a realidade, além de mostrar situações em que o modelo não previa, como a utilização de facilitadores dos software utilizados nas modificações.

Referências Bibliográficas

[1] MEYER, B. *Object-oriented software construction.* Prentice Hall International Series in Computer Science: [s.n.], 1997.

[2] FERREIRA, K. et al. *Um Modelo de Predição de Amplitude da Propagação de Modificações Contratuais em Software Orientado por Objetos*. Belo Horizonte, MG: [s.n.], Janeiro 2009.

[3] METRICS Tools. 2006. Disponível em: <http://www.laatuk.com/tools/metric-tools.html>.

[4] JAVACOUNT. 2006. Disponível em: <http://csdl.ics.hawaii.edu/Tools/JavaCount/JavaCount.html>.

[5] KRAKATAU Essencial Metrics. 2006. Disponível em: <http://www.powersoftware.com/>.

[6] FERREIRA, K. A. M. e. a. *Modularidade em Sistemas Orientados por Objetos*. São Paulo, SP: Pioneira Thomson Learning, 1212.

[7] FERREIRA, K.; BIGONHA, R.; BIGONHA, M. *Reestruturação de Software Dirigida por Conectividade para Redução de Custo de Manutenção*. Belo Horizonte, MG: [s.n.], 2008.

[8] MENDES, L. *Implementação de Métricas para Estimativa de Esforço de Manutenção em Sistemas Orientados por Objetos*. Belo Horizonte, MG: Relatório Técnico do Laboratório de Linguagens de Programação, 2009.

[9] MYERS, G. J. *Reliable software through composite design*. Nova York, NY: The MIT Press, 1975.

[10] MAINTABILITY Index. 2009. Disponível em: <http://archive.adaic.com/docs/present/ajpo/pll-cost/html/tsld054.htm>.

[11] ABREU; CARAPUCA. *Object-orientede software engineering: Measuring and controlling the development process.* Proceedings of 4th Int. Conf of software Quality Va, USA: [s.n.], 1994.

[12] WEKA(2009) Weka 3: Data Mining Software in Java. 2009. Disponível em: <http://www.biorecipes.com/DarwinHelp/LnGamma.html>.

[13] DARWINHELP : LnGamma. 2009. Disponível em: <http://www.cs.waikato.ac.nz/ml/weka/index.html>.