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

Luiz Felipe de Oliveira Mendes

2006041449

Projeto CONNECTA: Conectividade em Módulos

Implementação, Experimentos e Análise da Métrica de Estabilidade de Myers e do Modelo de K3B em CONNECTA

Relatório de Iniciação Científica

Belo Horizonte – MG
2009 / 2º semestre

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

Implementação, Experimentos e Análise da Métrica de Estabilidade de Myers e do Modelo de K3B em CONNECTA

por

Luiz Felipe de Oliveira Mendes

Relatório de Iniciação Científica

Apresentado como requisito do Projeto de Pesquisa de Iniciação Científica

Prof. Dra. Mariza Andrade da Silva Bigonha
Orientadora

Luiz Felipe de Oliveira Mendes
Bolsista

_____-

Luiz Felipe de Oliveira Mendes - Bolsista

_____

Mariza Andrade da Silva Bigonha - Orientadora

_____

Kecia Aline Marques Ferreira - Co-Orientadora

Belo Horizonte – MG
2009 / 2º semestre

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
MI	Índice de Manutenibilidade

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, já 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 uma adaptação da métrica de estabilidade de Myers[1] e o modelo para predição de amplitude da propagação de modificações contratuais em softwares orientados a objeto OO , 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 [2] 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 model for predicting time of maintenance in object-oriented software, called K3B, and provide some experiments in order to calibrate and validated it.

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

0.1 Informações sobre a iniciação

0.1.1 Sobre a Bolsa

Modalidade da Bolsa: Iniciação Científica

Vigência: 23/10/2007 a 23/10/2009

Orientadora: Mariza Andrade da Silva Bigonha

Co-Orientadora: Kécia Aline Marques Ferreira

Laboratório: Laboratório de Linguagens de Programação

Instituição de Ensino: Instituto de Ciências Exatas - UFMG

Agência: FAPEMIG

0.1.2 Sobre o Projeto

Título: Projeto CONNECTA: Conectividade em Módulos

Edital: FAPEMIG 001/2007 Universal

Coordenadora: Mariza Andrade da Silva Bigonha

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 haver 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[3][4][5][6][7]. Dentre eles destacam-se dois: CONNECTA[7] e o Modelo de Predição de Amplitude da Propagação de Modificações Contratuais de software orientado por objetos[3], 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[8]. 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.[3], 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.

Baseado nesses fatos, este trabalho elenca os seguintes objetivos:

- Implementar uma adaptação da métrica de estabilidade proposta inicialmente por Myers[1]. Detalhes dessa métrica são apresentados na Seção 3.
- Realizar um estudo experimental que visa comparar os resultados da métrica de estabilidade com a métrica que mede o índice de manutenibilidade, conhecida na literatura como MI(Índice de Manutenibilidade)[9]. Essa métrica será descrita na Seção 3.
- Incorporar a métrica implementada na ferramenta CONNECTA[7].
- Inserir na ferramenta CONNECTA a fórmula do modelo matemático proposto por Ferreira[3]
- Realizar experimentos para avaliar os resultados das métricas implementadas.

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

1. A implementação de uma adaptação da métrica de estabilidade de Myers.
2. A incorporação da métrica de estabilidade adaptada em CONNECTA.
3. Estudo da métrica MI para futura comparação com as métricas do CONNECTA.
4. A implementação da fórmula do modelo matemático proposta por Ferreira em CONNECTA.

1.2 Organização do Texto

Este documento está organizado 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 desse capítulo é apresentar a teoria necessária para entender do que se trata o trabalho.Nele 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 o 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 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.

2.3.1 Tipos de Coesão

Myers[1] 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.

2.4 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.

2.4.1 Tipos de Acoplamento

Myers[1] 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.

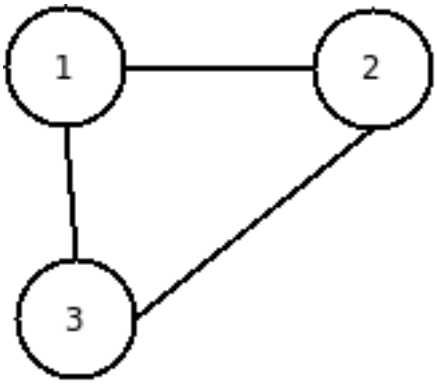

Figura 2.1: Exemplo de Grafo utilizado por Myers

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.

2.5 Estabilidade de Myers

Myers[1] propôs um modelo de avaliação para a estabilidade de software. Este modelo resulta em uma métrica global para o sistema que indica a quantidade média de módulos que devem ser alterados devido a alguma mudança em um dado módulo do sistema.O modelo tem como base teórica a Probabilidade e Teoria dos Grafos.

2.5.1 Cálculo da Estabilidade de Myers

Myers considera o sistema como um grafo não direcionado, nos quais os módulos são os vértices e uma aresta entre dois módulos representa uma comunicação entre eles, como mostrado na Figura 2.1. Isso resulta em um matriz *m* X *m* onde m é o total de módulos do sistema.

Para fazer os cálculos necessários Myers utiliza os valores de coesão e acoplamento como mostrado nas Tabelas 2.1 e 2.2, a partir dos tipos de coesão e acoplamento já explicados nessa seção.

Tipo de Acoplamento	Valor Associado
Conteúdo	0,95
Dado Comum	0,7
Elemento Externo	0,6
Controle	0,5
Informação	0,2
Dado Local	0,2

Tabela 2.1: Tipos de Acoplamento

Tipo de Coesão	Valor Associado
Coincidental	0,95
Lógica	0,4
Procedimental	0,4
Comunicacional	0,25
Funcional	0,2
Clássica	0,6
Informacional	0,2

Tabela 2.2: Valores de Coesão

Deve-se levar em consideração que esses valores foram definidos de forma arbitrária por Myers, com base na análise de problemas gerados pelos tipos de coesão e acoplamento.

O cálculo da estabilidade é feito da seguinte forma:

1. Constrói-se uma matriz de dependência *m* X *m* da seguinte forma: avalia-se o tipo de acoplamento existente entre cada par de módulo e preenche-se a posição Cij da matriz, correspondente ao par analisado, com o valor associado ao tipo de acoplamento.

2. Constrói-se um vetor S de m posições. Avalia-se o tipo de coesão interna de cada módulo preenche-se a posição Si correspondente do vetor como o valor associado ao tipo de coesão.

3. Constrói-se a matriz D de dependência de primeira ordem, a partir da seguinte fórmula:

$$\begin{array}{|l} \mathrm{D}_{ij} = 0.15(S_i + S_j) + 0.7C_{ij}, \text{ se } \mathrm{C}_{ij} \neq 0 \\ \mathrm{D}_{ij} = 0 \text{ se } \mathrm{C}_{ij} = 0 \\ \mathrm{D}_{ii} = 1 \text{ para todo i} \end{array}$$

Pode-se perceber nessa fórmula que Myers considera o acoplamento mais importante que a coesão no valor da estabilidade, já que o acoplamento é multiplicado por 0.7 e a coesão por 0.15.

A matriz gerada fornece um número que representa qual a probabilidade de haver mudanças no módulo j uma vez que houve uma mudança no módulo i diretamente, isto é, desconsidera a alteração nos outros módulos.

Para encontrar a probabilidade real de haver uma mudança em j uma vez que foi feito uma alteração em i, deve-se considerar todos caminhos entre i e j. Para isso é construída uma matriz E de dependência completa considerando as alterações indiretas. Esse modelo de avaliação considera os 3 caminhos de maior probabilidade entre dois módulos. A probabilidade de um caminho é dada pelo produto das probabilidades.

A matriz E é obtida da seguinte forma: para cada par de módulos i e j:

1. Encontram-se todos caminhos entre i e j
2. Se existe apenas um caminho Eij +Eji = P(x)
3. Se existe dois caminhos de i para j, Eij = Eji = P(x) + P(y) - P(x)P(y), onde P(x) ae P(y) são as probabilidades dos dois caminhos.
4. Se existem três caminhos de i para j, Eij = Eji = P(x) + P(y) + P(z) - P(x)P(y)-P(x)P(z) - P(y)P(z) + P(x)P(y)P(z), onde P(x), P(y) e P(z) são as probabilidades dos três caminhos.
5. Se existem mais de três caminhos de i a j, encontram-se os caminhos de maiores probabilidades. Aplica-se a regra do item anterior para esses três caminhos.

2.6 Adaptação a Estabilidade de Myers

A métrica de estabilidade de Myers foi definida para o paradigma estruturado. Esta métrica tem como objetivo prover a seguinte informação: caso haja uma alteração em um módulo do sistema, quantos outros módulos serão alterados. Ferreira et al. descreve em [7] que esta métrica mostra-se também muito útil para avaliação da estabilidade em sistemas OO se forem feitas as adaptações necessárias para atender às características desse paradigma.

A partir das adaptações a nova métrica foi implementada, mas durante a fase de experimentos algumas limitações foram identificadas na métrica adaptada. Uma delas diz respeito a sua utilização em um sistema com um número grande de classes. Nessa situação, o tempo necessário para o processamento dos dados é muito grande, o que torna inviável a sua aplicação. Um dos objetivos desse trabalho é encontrar uma maneira de contornar essas limitações e implementar essa adaptação no CONNECTA.

2.7 Índice de Manutenibilidade

O Índice de Manutenibilidade(MI)[9] é um conjunto de métricas polinomiais desenvolvidas pela Universidade de Idaho, que utiliza a métrica de Halstead(*Halstead's effort/module*) e Complexidade ciclomatica de McCabe (*McCabe's cyclomatic complexity/module*) entre outros fatores. O MI é usado principalmente para determinar a dificuldade de se manter um código. Sua saída é um número entre 0 e 100, sendo que número mais altos são melhores.

O MI não depende de linguagem e foi validado pela Hewlett-Packard (HP). A HP concluiu que módulos com MI menor que 65 são difíceis de manter. Essa métrica será vista com mais detalhes na Seção 4 desse documento.

O estudo desse conjunto de métricas é importante pois seus objetivos estão muito próximos daqueles definidos nesse trabalho. Sendo assim um melhor entendimento da MI pode proporcionar comparações com as métricas presentes no CONNECTA além de possibilitar possíveis otimizações nas mesmas.

2.8 CONNECTA

CONNECTA[7] é uma ferramenta de coleta de métricas implementada em Java que viabiliza a aplicação das métricas propostas no modelo MACSOO para software desenvolvidos em Java.

A idéia central do Modelo de Avaliação de Conectividade em Sistemas Orientados por Objetos, ou MACSOO[8], 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 considera a conectividade como sendo o principal fator na avaliação

da qualidade estrutural de um software e, consequentemente, deve ser tida como um fator de suma importância na manutenção e no custo do sistema.

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.

2.9 Algoritmo de Dijkstra

O algoritmo de Dijkstra[10], cujo nome se origina de seu inventor, o cientista da computação Edsger Dijkstra, soluciona o problema do caminho mais curto em um grafo dirigido ou não com arestas de peso não negativo, em tempo computacional O([m+n]log n) onde m é o número de arestas e n é o número de vértices.

O algoritmo de Dijkstra foi usado na adaptação da estabilidade de Myers, sua utilização será descrita em mais detalhes na Seção 4 deste documento.

2.9.1 Pseudo Algoritmo

Em linhas gerais, o algoritmo funciona da seguinte maneira:

Inicialmente inicializa as varáveis como mostrado no pseudo-código abaixo:

para todo $v \in V[G]$
$\quad d[v] \rightarrow \infty$
$\quad \pi[v] \rightarrow nulo$
$d[s] \rightarrow 0$

- V[G] é o conjunto de vértices(v) que formam o Grafo G.
- s é o vértice inicial.
- d[v] é o vetor de distâncias de s até cada v. Admitindo-se a pior estimativa possível, o caminho infinito.
- $\pi[v]$ identifica o vértice de onde se origina uma conexão até v de maneira a formar um caminho mínimo.

Temos então que usar dois conjuntos:

1. S, que representa todos os vértices v onde d[v] já contem o custo do menor caminho
2. Q, que contem todos os outros vértices.

Realizam-se nos componentes desse conjuntos uma série de relaxamentos das arestas, de acordo com o código:

```
enquanto Q ≠ vazio
        u ← extraia-mín (Q)
        S ← S ∪ u
        para cada v adjacente a u
            se d[v] > d[u] + w(u,v) //relaxe(u,v)
                então d[v] ← d[u] + w(u,v)
                        π[v] ← u
```

- w(u, v) representa o peso da aresta que vai de u a v.
- u e v são vértices quaisquer e s é o vértice inicial.
- extraia-mín(Q), pode ser um heap de mínimo ou uma lista ordenada de vértices onde obtém-se o menor elemento, ou qualquer estrutura do tipo.

No fim do algoritmo obtém-se o menor caminho entre s e qualquer outro vértice de G.

2.10 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.

2.10.1 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. Sua definição é mostrada 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
```

2.11 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.

2.11.1 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 2.2 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.

2.11.2 Cálculo da COF

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

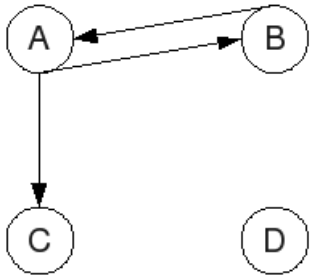

Figura 2.2: Relacionamento Cliente-Servidor entre classes

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.

2.12 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 produção de uma ferramenta para auxiliar o projetista no desenvolvimento do software (implementação).

3.1 Procedimentos Metodológicos

Afim de atingir os objetivos propostos nesse trabalho de pesquisa, inicialmente foi feito um levantamento bibliográfico, no qual pode-se destacar o estudo da dissertação de mestrado de Ferreira[7] e o estudo da ferramente CONNECTA. Além disso foram implementadas mudanças na métrica de estabilidade de Myers dentro do software CONNECTA, e implementou-se também o modelo matemático proposto por Ferreira chamado de K3B.

4 RESULTADOS E DISCUSSÃO

Esta seção trata os resultados do trabalho de Iniciação Científica incluindo a discussão sobre as implementações e as informações obtidas nos estudos feitos.

4.1 Adaptação da métrica de Estabilidade de Myers

Como já explicado na Seção 2.4 desse documento, Myers[1] propôs uma métrica para medir a estabilidade de um programa. Devido a época em que foi criada, essa métrica calcula a estabilidade de software que seguem o paradigma estruturado. Ferreira[7], propõe mudanças nessa métrica para adaptá-la para o paradigma orientado a objeto(Seção 2.1).

4.1.1 Diferenças

Nessa seção explicarei quais foram as modificações feitas na métrica de Myers.

O grafo que representa o sistema a ser analisado passa de não direcionado para direcionado como mostra a Figura 4.1. Essa mudança se faz necessária pois se dois módulos A e B estão conectados, o fato de que uma modificação em A impactar em B nem sempre implica que uma alteração em B também impactará A. Por esse motivo, a melhor representação seria com um grafo direcionado.

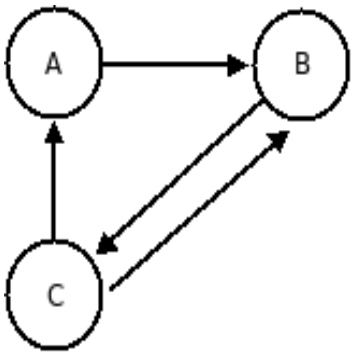

Figura 4.1: Exemplo de grafo utilizado na adaptação

Outra mudança que se faz necessária é a modificação nas escalas de acoplamento e coesão para Orientação a Objeto. As Tabelas 4.1 e 4.2 mostram esses novos valores.

Tipo de Acoplamento	**Estruturado**	**Orientado a Objeto**
Conteúdo	0,95	0,95
Dado Comum	0,7	0,7
Inclusao	–	0,7
Elemento Externo	0,6	0,6
Controle	0,5	0,5
Referência	–	0,35
Informação	0,2	0,2
Dado Local	0,2	–

Tabela 4.1: Comparação dos pesos de acoplamento

Tipo de Coesão	**Estruturado**	**Orientado a Objeto**
Coincidental	0,95	0,95
Lógica	0,4	0,4
Temporal	–	0,6
Procedimental	0,4	0,4
Comunicacional	0,25	0,25
Contratual	–	0,2
Funcional	0,2	–
Clássica	0,6	–
Informacional	0,2	–

Tabela 4.2: Comparação dos pesos de Coesão

O critério escolhido para os novos valores foi de manter os pesos definidos por Myers para aqueles tipos de coesão e acoplamento presentes nos dois paradigmas, mudando apenas a forma como eles são definidos.

As principais diferenças decorrem de que no paradigma OO existe o tipo de acoplamento de Inclusão que não existe no estruturado. A esse acoplamento foi associado o valor 0,7 pois ele possui muitas semelhanças com o acoplamento por Dado Comum presente no paradigma estruturado. Além disso, em OO, existe a coesão de Interesse que substitui os tipos Informacional e Funcional presentes no paradigma estruturado, por esse motivo essa coesão ganha o valor 0,2, que é o melhor valor possível para coesão interna.

Outro ponto importante do paradigma OO é a existência da herança, sua utilização é de suma importância nos sistemas OO e não devem ser ignorados nos cálculos.Sendo assim, definiu-se que o relacionamento de herança teria o valor 0,7, pois o relacionamento em questão introduz uma dependência muito estreita em que alterações possuem grande probabilidade propagação.

4.1.2 Problema Encontrado Durante a Implementação

Nos testes da adaptação da métrica de Estabilidade no CONNECTA foi encontrado um problema crítico. O programa aparentemente entrava em loop ao analisar um software composto por um número relativamente grande de classes, em torno de 800. Uma primeira análise mostrou que o loop acontecia na parte do programa que fazia o cálculo de todos os caminhos do grafo.

Ao analisar melhor o problema, foi descoberto que o programa não estava realmente em loop, na verdade o cálculo de todos os caminhos possui um custo muito alto, e é considerado um problema NP-completo.

Antes da mudança o cálculo era feito de acordo com o algoritmo da Tabela 4.3.

Para todo i $\in$ a G Acha todos caminhos que partem de i.

Tabela 4.3: Cáculo de todos caminhos

sendo :

- G é um grafo que representa o sistema
- i é um Vértice de G

No algoritmo apresentado, o tamanho total dos caminhos era calculado fazendo a soma dos valores de cada aresta, e só depois de encontrar os 3 maiores caminhos era que se fazia o cálculo da probabilidade.

Proposta de Solução

De forma a contornar esse problema foi preciso modificar o algoritmo. Ao invés de calcular os três caminhos de maior probabilidade no grafo do sistema como mostrado na Tabela 4.3, o novo algoritmo calcula apenas o caminho de maior probabilidade.

Para encontrar o maior caminho em um grafo é possível utilizar algoritmos de calculo de menor caminho, faz-se isso trocando os valores do grafo para negativo, isto é, uma aresta que tinha valor 2 terá valor -2.

Sabendo disso, escolheu-se o algoritmo mais conhecido para encontrar o menor caminho em grafos: Algoritmo de Dijkstra.(Seção 2.8).

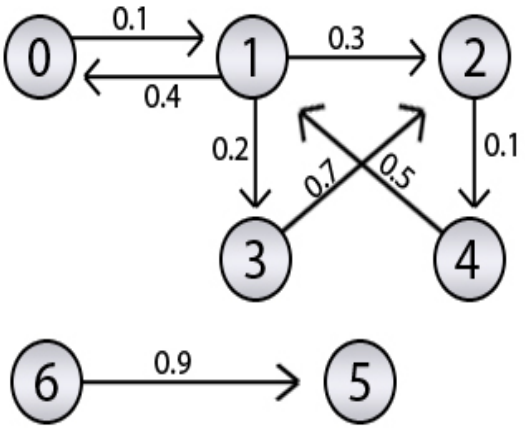

Figura 4.2: Grafo direcionado com ciclos e valores entre 0 e 1.

Contudo, apenas utilizar o algoritmo de Dijkstra não é o bastante para encontrar o caminho de maior probabilidade. Foram necessárias as seguintes mudanças no algoritmo:

- Primeiramente o algoritmo tem de receber um grafo com arestas tendo valor entre 0 e 1 com uma casa decimal, como o exemplo da Figura 4.2.
- O algoritmo deve substituir todos os pesos das arestas de forma que todos valores diferentes de 0 fiquem negativos.
- No terceiro passo do pseudo-algoritmo exposto na Seção 2.8 deve-se colocar uma multiplicação, ao invés de soma, mas tomando os cuidados necessários para manter o resultado negativo.
- Na implementação foi utilizada uma função para truncar os resultados da multiplicação, pois alguma vezes seu resultado era indesejado e incorreto por algum motivo interno a Java e seu tratamento de doubles.

Inicializar $S' \leftarrow 2,3,....,n$,
$S \leftarrow 1$,
$\pi(1) \leftarrow 0$,
$\pi(j) \leftarrow -c_{1j}$ se j conecta com 1
$\pi(j) \leftarrow +\infty$ caso contrário

Enquanto $S' \neq Vazio$ faça:
Selecionar $j \in S'$ tal que $\pi(j) = min_{i \in S'} \pi(i)$
$S' \leftarrow S' - j$ Para $\forall i \in S'$ e se i conecta com j
$\pi(i) \leftarrow min(\pi(i), -(\pi(j) * c_{ji}))$
fim do enquanto

Tabela 4.4: Pseudo-Algoritmo do Dijkstra modificado

4.1.3 Testes

Para testar se o problema exposto na Seção 4.1.2 foi resolvido com uso a solução proposta, foram feitos vários testes dessa métrica tendo como entrada sistemas que possuem um grande número de classes.

Primeiramente utilizou-se o software no qual o problema foi identificado, e o CONNECTA que antes ficava computando a métrica de estabilidade por mais de quinze horas sem terminar, mostrou o resultado em menos de 1 minuto. Além desse programa, utilizou-se outras quinze entradas para o CONNECTA, sendo que três delas possuiam um número maior de classes que o primeiro, chegando a ter mais de 3500 classes. Em todos os casos a métrica não encontrou problemas e terminou em poucos minutos.

Para verificar se a modificação feita na métrica de estabilidade não gerava resultados inconsistentes foi feito um pequeno teste. Utilizou-se para esse teste um software de controle acadêmico com poucas classes. Este programa possui 4 versões, cada uma delas passou pelo CONNECTA antes e depois da modificação feita devido ao problema exposto na Seção 4.2.1.

Primeira Versão do Controle Acadêmico

Essa versão possui classes de baixa coesão e um número grande de conexões entre elas, com forte acoplamento, devido ao uso de atributos públicos. A Tabela 4.5 mostra os resultados antes e depois da modificação.

Métrica	Valor Anterior	Valor Atual
Número de Classes	5	5
Número de Conexões	6	6
Estabilidade	**2,1**	**2,08**
COF	0,3	0,3
CoCo Média	–	0,7
Modelo Matemático	–	0,0258541

Tabela 4.5: Valores para Primeira Versão do Controle Acadêmico

Pode-se perceber que, como esperado, a nova métrica de estabilidade resultou em um valor muito próximo ao da antiga, apenas 0,02 de diferença.

Segunda Versão do Controle Acadêmico

Na segunda versão da aplicação uma de suas classes foi reestruturada, sendo substituida por duas classes de maior coesão. A Tabela 4.6 mostra os resultados antes e depois da modificação.

Métrica	Valor Anterior	Valor Atual
Número de Classes	6	6
Número de Conexões	8	8
Estabilidade	**2,2**	**2,2**
COF	0,27	0,27
CoCo Média	–	0,75
Modelo Matemático	–	0,0188392

Tabela 4.6: Valores para Segunda Versão do Controle Acadêmico

Nessa versão o resultado da métrica de estabilidade foi 2,2 tanto para versão antiga, quanto para nova.

Terceira Versão do Controle Acadêmico

Na terceira versão da aplicação uma de suas classes, que antes possuia todas constantes utilizadas, foi dividida em três outras classes de maior coesão interna. Os valores gerados para essa versão se encontram na Tabela 4.7.

Nessa terceira versão o resultado da métrica de estabilidade foi extremamente próximo, dando uma diferença ínfima de 0,013.

Métrica	Valor Anterior	Valor Atual
Número de Classes	8	8
Número de Conexões	9	9
Estabilidade	**2,1**	**2,113**
COF	0,16	0,16
CoCo Média	–	0,6875
Modelo Matemático	–	0,0001383

Tabela 4.7: Valores para Terceira Versão do Controle Acadêmico

Quarta Versão do Controle Acadêmico

Na quarta e última versão da aplicação a utilização de atributos públicos foi eliminada da maioria das classes, deixando apenas as classes depositórias de constantes com os dados visíveis. O resultado gerado para essa versão se encontram na Tabela 4.8.

Métrica	Valor Anterior	Valor Atual
Número de Classes	8	8
Número de Conexões	7	7
Estabilidade	**1,5**	**1,5**
COF	0,13	0,13
CoCo Média	–	0,666625
Modelo Matemático	–	0,0000244

Tabela 4.8: Valores para Quarta Versão do Controle Acadêmico

Na última versão da aplicação a nova métrica de estabilidade manteve o valor da antiga.

Conclusões

Os testes realizados nas quatros versões da aplicação para Controle Acadêmico abrem espaço para algumas conclusões. A Tabela 4.9 mostra alguns dados resultantes desses testes.

Métrica	V1 - V2	V2 - V3	V3 - V4
Estabilidade	Pequena melhora	Melhora	Grande Melhora
COF	Melhora	Grande Melhora	Melhora
CoCo Média	Pequena piora	Melhora	Melhora
Modelo Matemático	Melhora	Melhora	Melhora

Tabela 4.9: Diferença entre as versões

A Tabela 4.9 mostra se houve uma melhora ou piora no resultado obtido por cada métrica entre as versões. Com referência nos resultados da mesma pode-se perceber que com a melhora no código entre as versões houve também uma melhora na grande maioria dos resultados. É importante destacar que o modelo matemático(Seção 4.3) obteve um bom comportamento na diferença entre as versões, mostrando uma melhora em todas as etapas.

Versão	Diferença na Estabilidade
Versão 1	0,02
Versão 2	0,00
Versão 3	0,013
Versão 4	0,00

Tabela 4.10: Diferença entre as versões

Como pode ser visto na Tabela 4.10, a nova métrica de estabilidade gerou resultados iguais ou extremamente próximos à anterior. A mudança então pode ser considerada efetiva, já que resolveu o problema do tempo de execução exposto na Seção 4.1.2 e obteve resultados extremamente próximos ou iguais aos encontrados antes da mudança.

4.2 Índice de Manutenibilidade

A métrica MI(*Maintainability Index*), traduzida para Índice de Manutenibilidade, foi criada com objetivo de medir a manutenibilidade de sistemas(Seção 2.6).

A MI foi encontrada durante uma pesquisa sobre métricas similares às presentes no CONNECTA(Seção 2.7), seu estudo se faz necessàrio justamente pela similaridade no objetivo final que é a diminuição dos custo de manutenção de um software. Nesta seção demonstrarei como a MI é calculada e quais medidas ela utiliza.

4.2.1 Cálculo do Índice da Manutenibilidade

No caso da métrica MI, a manutenibilidade do programa é calculada utilizando uma combinação de medidas comuns e largamente utilizadas, como a métrica de Halstead 4.2.2 e a complexidade ciclomática de McCabe 4.2.3. O índice de manutenibilidade básico de um conjunto de programas é dado por um polinômio da seguinte forma:

$$171 - 5.2 * \ln(aveV) - 0.23 * aveV(g') - 16.2 * \ln(aveLOC) + 50 * \sin(\sqrt{2.4 perCM})$$

Tabela 4.11: Polinômio do Índice da Manutenibilidade

Sendo que:

- aveV é a média da medida de Halstead (4.2.2) por módulo.
- aveLOC é a média de linhas de código por módulo.

- aveV(g') é a média extendida da complexidade ciclomática(4.2.3) por módulo.
- perCM é a média do percentual de comentários por linha de código.

A medida de comentários por linha de código não é obrigatória, cabe ao analista verificar se ela é uma medida importante para aplicação em questao.

A métrica MI pode ser utilizada de diversas maneiras, como:

- O sistema pode ser verificado periodicamente para manutenabilidade, sendo uma maneira de calibrar as equações.
- A métrica pode ser incorporada no desenvolvimento para demonstrar qualidade do código enquanto este está sendo construido e modificado.
- Avaliando módulos selecionados para guiar a manutenção, procurando encontrar códigos de alto risco.
- A MI pode ser usada para comparar e avaliar sistemas: Comparando a MI de sistemas de alta qualidade com outros sistemas de terceiros pode provar informações para uma decisão de fazer ou comprar.

4.2.2 Halstead's effort/module

A medida de complexidade de Halstead(Halstead complexity) foi desenvolvida para medir a complexidade modular de um programa diretamente do código fonte, com ênfase na complexidade computacional.

Essa métrica foi desenvolvida por Maurice Halstead[12] com objetivo de determinar uma medida quantitativa para complexidade diretamente dos operadores e operandos de um módulo.Como é aplicada diretamente no código fonte, é normalmente utilizada como métrica de manutenção.

As opiniões sobre essa métrica variam muito, desde "não confiável"[13] até "uma das medidas mais fortes de manutenibilidade"[14], sua utilização pura é muito dependente da aplicação, mas sua utilização juntamente com outras métricas é com certeza algo a ser estudado.

Forma de Cálculo

As medidas de Halstead são baseadas em quatro valores derivados diretamente do código fonte, que são os seguintes:

- n1 = O número de operadores distintos
- n2 = O número de operandos distintos
- N1 = O número total de operadores
- N2 = O número total de operandos

Com esses valores podem ser calculadas as seguintes medidas:

Medida	**Símbolo**	**Fórmula**
Extensão do Programa	N	$N = N1 + N2$
Vocabulário do Programa	n	$n = n1 + n2$
Volume	V	$V = N * (LOG2n)$
Dificuldade	D	$D = (n1/2) * (N2/n2)$
Esforço	E	$E = D * V$

Tabela 4.12: Tabela das medidas de Halstead

Essas medidas são simples de serem calculadas, desde que se tenha as regras para identificar os operandos e operadores, contudo, essas regras podem ser difíceis de serem encontradas.A extração dos componentes dessas fórmulas requer um scanner sensitivo a linguagem, o que não é complicado na maioria das linguagens.

Avaliação

As medidas de Halstead foram introduzidas em 1977 e desde entao vem sendo usadas em variados experimentos. É uma da medidas mais antigas para complexidade de um programa.

As medidas de Halstead são aplicáveis em sistemas operacionais e no desenvolvimento, assim que o código tenha sido escrito já que depende do código fonte. Utilizar essas medidas durante o projeto pode ajudar no desenvolvimento, já que uma mudança drástica na medida de complexidade pode apontar para uma modificação de alto risco. Essa métrica vem sido criticada por diversas razões, entre elas deve-se ressaltar que as medidas de Halstead medem a complexidade léxica e/ou textual ao invés de medir a complexidade estrutural ou o fluxo lógico como a Complexidade Ciclomática (4.2.3). Por esse motivo sua utilização deve ser feita juntamente com outras medidas, como pode se ver nos cáculos da MI(Seção 4.2.1) de forma a auxiliar a criação de um resultado mais fidedigno.

4.2.3 McCabe's cyclomatic complexity

Complexidade ciclomática, também chamada simplesmente de complexidade do programa(*program complexity*) ou Complexidade de McCabe(*McCabe's complexity*) é a métrica mais utilizada dentro da classe de de métricas estáticas para softwares. Foi criada por Thomas McCabe em 1976, e mede o número de caminhos linearmente independentes dentro de um módulo.Essa medida gera um número que pode ser comparado a a complexidade de outros programas.

A complexidade ciclmática de McCabe é uma métrica de software largamente utilizada e foi planejada para ser independete da linguagem.

Forma de Cálculo

A complexidade ciclomática do módulo de um software é calculada utilizando o grafo conectado desse módulo (que mostra a topologia de controle de fluxo dentro do programa). O cálculo é feito da seguinte maneira:

- Complexidade ciclomática (CC) = E - N + p

  onde:

- E= número de arestas do grafo
- N= número de nodos do grafo
- p= número de componentes conectados

Para contar esses elementos, é necessário estabelecer convenções de contagem (ferramentas que coletam complexidade ciclomática possuem essas conveções).

Foram medidos inúmeros programas, e a complexidade encontrada nesses testes ajuda os engenheiros de softwares a determinar o risco e estabilidade inerentes. Estudos mostram uma correlação entre a complexidade ciclomática de um programa com sua frequência de erro. Uma baixa complexidade ciclomática contribui para inteligibilidade. A complexidade ciclomática de um programa é um grande fator na sua testabilidade.

Após muitos testes e experimentos, criou-se uma Tabela de valores para comparação:

Complexidade Ciclomática	Avaliação de risco
1-10	Um programa simples, sem muito risco
11-20	Um programa mais complexo, risco moderado
21-50	Complexo, programa de alto risco
Maior que 50	Programa altamente instável

Tabela 4.13: Valores de Complexidade Ciclomática

Avaliação

Considera-se que a complexidade ciclomática fornece uma medida mais forte da complexidade estrutural de um programa do que contar as linhas de código (LOC - será discutida adiante). A complexidade ciclomática pode ser utilizada em várias áreas, como:

- **Análise de Risco no desenvolvimento do código.** - Avaliar o risco enquanto o código está em desenvolvimento.
- **Planejamento de testes** - Análises matemáticas tem mostrado que a complexidade ciclomática fornece o número exato de testes necessários para testar todos pontos de decisão no programa para cada sáida. Um módulo muito complexo que necessitaria de um número extenso de testes poderia ser quebrado em módulos menores, menos complexos e mais facilmente testáveis.
- **Reengenharia** - A análise da complexidade ciclomática fornece conhecimentos da estrutura de um sistema operacional. O risco envolvido na reegenharia e um pedaço de código está relacionado com sua complexidade. Portanto, o custo e análise de risco podem se beneficiar de uma boa aplicação dessa métrica.

A complexidade ciclomática pode ser calculada manualmente para pequenos programas, mas é melhor possuir ferramentas que a calcule. Para gerar grafos automaticamente e calcular a complexidade é necessário uma tecnologia sensível a linguagem, precisa de um front-end(parser) para cada linguagem.

É preciso ressaltar no entanto que um módulo com alta complexidade ciclomática não representa necessariamente um excesso de risco, ou que deve ser refeito de maneira mais simples. É necessário entender melhor o seu funcionamento específico na aplicação antes de tirar qualquer tipo de conclusão sobre sua complexidade.

4.2.4 Outros Fatores

Além da Complexidade ciclomática e da complexidade de Halstead, a métrica Índice de Manutenibilidade(MI) também faz uso de duas outras métricas, a linhas de código (LOC) e comentários por módulo(opcional).

LOC

A maneira mais simples de medir o tamanho de um programa é contar as linhas de código. Esta é a mais antiga e amplamente utilizada métrica para calculo de tamanho, que logicamente possui muitos defeitos graças a sua simplicidade.

Os críticos dessa métrica dizem que as medidas LOC são dependentes da linguagem de programação utilizada na codificação do projeto, que elas penalizam programas bem projetados, porém mais curtos, que elas não podem acomodar facilmente linguagens não-procedurais e que seu uso em estimativas requer um nível de detalhes que pode ser difícil de conseguir , isto é, o planejador deve estimar as linhas de código a ser produzidas muito antes que a análise e o projeto tenham sido construídos.

Logicamente a utilização solitária das linhas de código não é uma boa estimativa para nada, mas juntamente com outras métricas como Complexidade ciclomática e complexidade de Halstead pode se tornar algo mais viável.

CPM

O número médio de comentários por módulo pode ser usado na fórmula da MI. Essa métrica tem o objetivo de medir a facilidade de entendimento do código que é um fator importante na manutenibilidade de um código.

4.3 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 etal [7] propõe uma maneira de investigar o esforço de manutenção em sofwares 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.

4.3.1 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 de 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[3], e sua difícil incorporação no sistema CONNECTA, decidimos por implementar uma simplificação da fórmula que parecia, em primeiro momento, corresponder ao resultado do modelo completo. A tabela 4.14 mostra esta fórmula.

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

Tabela 4.14: Fórmula do modelo matemático

sendo:

- n o número de módulos do sistema
- α 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 simplificações.

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

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

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 potência para grandes valores de n produziam números que tendiam a 0, e não foi possível armazenar tais números através de varivéis de ponto flutuante.

Para contornar os problemas descritos, utilizou-se trasformações matemáticas, por exemplo, logarítimos e a função LnGamma [15].

Para um inteiro positivo a, LnGamma retorna o logarítimo do produto $1*2*3*\ldots*(a-1) = ln((a-1)!)$. 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 [16].

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 [3].

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 numa diminuição de custos.

Diminuir a conectividade de um software resulta em uma melhora em sua manutebilidade. Tendo isso em vista, nesse trabalho foram implementadas modificações na métrica de Estabilidade de Myers, que é uma medida que indica a quantidade de módulos que devem ser modificados dada uma alteração no software, além disso, implementou-se também uma simplificação do modelo matemático proposto por Ferreira et al.[7] que objetiva calcular o tempo de estabilização de um software em manutenção. Ambas implementações ocorreram no software CONNECTA[7] que tem como objetivo medir a conectividade do software, além de auxiliar o projetista na tomada de decisão durante a contrução de software na tentativa de reduzir seu custo total.

Foi feito também um estudo sobre a métrica "Índice de Manutenibilidade"com objetivo de no futuro compará-la com a adaptação da métrica de Estabilidade de Myers.

Referências Bibliográficas

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

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

[3] 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.

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

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

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

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

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

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

[10] DIJKSTRA Algorithm. 2009. Disponível em: <http://en.wikipedia.org/wiki/Dijkstrasalgorithm/>.

[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] HALSTEAD, M. H. *H. Elements of Software Science, Operating, and Programming Systems Series Volume 7.* New York, NY: [s.n.], 1977.

[13] JONES, C. J. *Software Metrics: Good, Bad, and Missing.* Prentice Hall International Series in Computer Science: [s.n.], 1994.

[14] OMAN, P. *HP-MAS: A Tool for Software Maintainability, Software Engineering*. Test Laboratory, University of Idaho: [s.n.], 1991.

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

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