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

Luiz Felipe de Oliveira Mendes

Implementação de Métricas para Estimativa de Tempo de Manutenção em Sistemas Orientados por Objetos

Monografia de Projeto Orientado em Computação I

Belo Horizonte – MG
2009 / 1º semestre

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

Implementação de Métricas para Estimativa de Tempo de Manutenção em Sistemas Orientados por Objetos

por

Luiz Felipe de Oliveira Mendes

Monografia de Projeto Orientado em Computação I

Apresentado como requisito da disciplina de Projeto Orientado em Computação I 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 / 1º semestre

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

Agradecimentos

Gostaria de agradecer a minha família e a minha namorada pelo apoio.
Agradeço também meus professores pelo conhecimento adquirido.
Finalmente gostaria de agradecer os grandes amigos que fiz no curso, sem eles não teria chegado até aqui.

"Any intelligent fool can make things bigger, more complex, and more violent. It takes a touch of genius and a lot of courage to move in the opposite direction."

Albert Einstein

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Siglas

Resumo

O presente trabalho tem como objetivo...

Palavras-chave: Métricas.

Abstract

This paper aims to...

Keywords: Metrics.

1 INTRODUÇÃO

1.1 Visão geral

Uma das questões mais importantes na produção de software é o custo, 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 da 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, já 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 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[**?**][**?**][**?**][**?**][1]. Dentre eles destacam-se dois: CONNECTA[1] e o Modelo de predição de tempo de manutenção de software orientado por objetos[**?**], por terem uma relação direta com o trabalho proposto 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[**?**]. 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.

Com base no problema exposto, Ferreira et al.[**?**], 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 de estabilização de um software quando um ou mais módulos

sofrem manutenção que altera suas interfaces.

1.2 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: Os objetivos propostos no início do trabalho foram:

- Implementar uma adaptação da métrica de estabilidade proposta inicialmente por Myers[2]. 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)[**?**]. Essa métrica será descrita na Seção 3.
- Incorporar a métrica implementada na ferramenta CONNECTA[1].
- Inserir na ferramenta CONNECTA a fórmula do modelo matemático proposto por Ferreira[**?**]
- 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.

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 que foram utilizados.

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, cada classe determina o comportamento (definido nos métodos) e 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 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 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.

2.2.1 Tipos de Coesão

Myers[**?**] 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 chamar o módulo 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.

2.3 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 Tipos de Acoplamento

Myers[**?**] 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 Domum

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 Estabilidade de Myers

Myers[2] 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 do sistema 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.

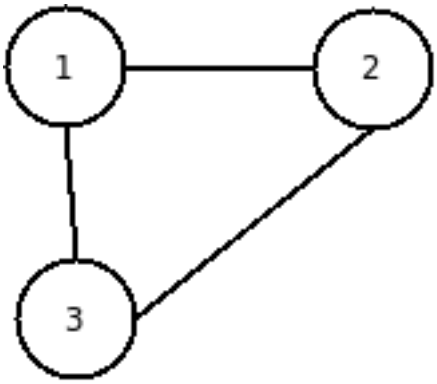

Figura 2.1: Exemplo de Grafo utilizado por Myers

2.4.1 Forma de Calcular

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.Isso resulta em um matriz *m* X *m* onde m é o total de módulos do sistema.

Para fazer os cálculos necessários Myers utiliza duas tabelas, com os valores de coesão e acoplamento, utilizando os tipos de coesão e acoplamento já explicados nesse capítulos.

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:

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

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

A matriz gerada fornece um número que representa quala probabilidade de haver mudanças no módlo j sendo 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 sendo 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 apena sum 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.

2.5 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 outro módulos serão alterados. Contudo Ferreira et al. descreve em [1] 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 dessas 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.

2.6 Índice de Manutenibilidade

O Índice de Manutenibilidade(MI)[**?**] 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.

2.7 CONNECTA

CONNECTA[1] 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[**?**], 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.

2.8 Algoritmo de Dijkstra

O algoritmo de Dijkstra[**?**], 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.

2.8.1 Pseudo Algoritmo

Em linhas gerais, o algoritmos funciona da seguinte forma:

- V[G] é o conjunto de vértices(v) que formam o Grafo G.

```
para todo v ∈ V[G]
        d[v] → ∞
        π[v] → nulo
 d[s] → 0
```

- 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 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 então 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.

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[1] e o estudo da ferramente CONNECTA. Além disso foram implamentadas mudanças na métrica de estabilidade de Myers dentro do software CONNECTA, e implementou-se também o modelo matemático proposto por Ferreira.

Com o objetivo de iniciar o trabalho do POCII foi feito um estudo da métrica "Índice de Manutenibilidade"(MI), que será de suma importância, uma vez que um dos objetivos do POCII é coletar as métricas MI e de Myers em um conjunto de softwares e analisar os resultados obtidos. Os outros objetivos do POCII inicialmente são: Estudo de métricas similares ao do modelo citado e a avaliação do Modelo de Ferreira.

4 RESULTADOS E DISCUSSÃO

4.1 Adaptação da métrica de Estabilidade de Myers

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

4.1.1 Diferenças

Nessa sessão explicarei quais foram as modificações feitas na métria de Myers.

O grafo que representa o sistema a ser analisado passa de não direcionado para direcionado, 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.

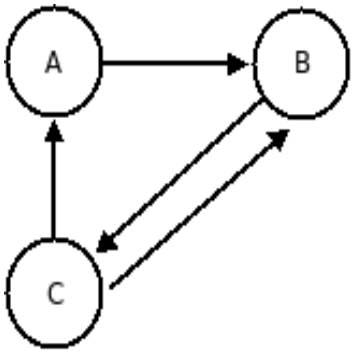

Figura 4.1: Exemplo de grafo utilizado na adaptação

Outra mudança que se faz necessária é a modificação nas escalas de acoplamento e coesão para Orientação a Objeto.

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 Contratual 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

Nos testes da adaptaçao 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.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.

De forma a contornar esse problema foi feita a seguinte modificação no algoritmo:

Algoritmo Antigo	**Algoritmo Novo**
Calcula-se os três caminhos de maior probabi lidade no grafo do sistema.	Calcula-se apenas o caminho de maior probabilidade no grafo do sistema.

Tabela 4.3: Modificação feita na métrica de Estabilidade

Antes da mudança o cálculo era feito da seguinte forma: Sendo :

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

Tabela 4.4: Cáculo de todos caminhos

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

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 fazia o calcula da probabilidade.

Resolução

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 de encontrar menor caminho em grafos: Algoritmo de Dijkstra.(Seção 2.8).

Apenas utilizar o Dijkstra não é o bastante para encontrar o caminho de maior probabilidade, foram necessárias algumas mudanças no algoritmo, que serão expostas a seguir.

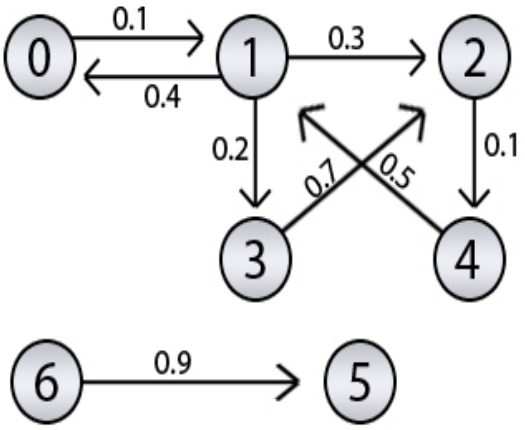

Figura 4.2: Grafo direcionado com ciclos e valores entre 0 e 1.

- 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 mudar todos os pesos das arestas de forma que todos pesos diferentes de 0 devem ficar negativos.
- No terceiro passo do pseudo-algoritmo exposto na Sessão 2.8 deve-se colocar uma multiplicação, ao invés de soma, mas tomando os cuidados necessários para manter o resultado negativo.
- 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.

Pseudo-Algoritmo

Inicializar $S' \leftarrow 2,3,....,n$,
$S \leftarrow 1$,
$\pi(1) \leftarrow 0$,
$\pi(j) \leftarrow -c_{1j}$ se $j \in Z^+$
$\pi(j) \leftarrow +\infty$ 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 4.5: Pseudo-Algoritmo do Dijkstra modificado

4.2 Índice de Manutenibilidade

O Índice de Manutenibilidade(MI)[**?**] é um conjunto de métricas polinomiais desenvolvidas pela Universidade de Idaho, que utiliza o módulo 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.

4.2.1 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[Bibliografia] 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"[**?**] até "uma das medidas mais fortes de manutenibilidade"[**?**], 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 4.6: 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 (4.2.2).Por esse motivo sua utilização deve ser feita juntamente com outras medidas, como pode se ver nos cáculos da MI(4.2.4) de forma a auxiliar a criação de um resultado mais fidedigno.

4.2.2 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 4.7: 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.

4.2.3 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.

4.2.4 Cálculo da MI

COISAS A COLOCAR:

Aqui deve fica a maioria da novidade, colocar aqui como cheguei nos resultados, qual a modificacao feita na adaptacao a métrica de meyers e porque ela foi feita (tempo), tentar colocar qual a extensao desse problema em notacao de O().

Dificuldades encontradas.

Mostrar aqui as modificacoes ao algoritmo de Dijkstra.

Mostrar aqui qual a notacao nova em O().

Aqui devo colocar uma tabela com os resultados da adaptacao da métrica de Estabilidade de Meyers.

Aqui dissertar sobre o modelo matemático (colocar formulas em apendice).

Falar sobre as tentativas de coloca-lo como estava planejado no Connecta, mostrar os problemas de tamanho com tabela que mostra numeros.

Falar sobre dicussao para nova tentativa do modelo.

COMPARAR OS RESULTADOS ESPERADOS AQUI OU NA CONCLUSAO?!?!?!?

ESSES ERAM OS RESULTADOS ESPERADOS

Tendo em vista os objetivos desse trabalho, espera-se que a adaptação da métrica de Estabilidade de Meyers seja correta e que com isso ela retrate de forma bastante aproximada a dificuldade da estabilização quando um ou mais módulos sofrem manutenção que alterem suas interfaces. Além disso espera-se que caso alguma métrica similar seja encontrada, que a adaptação feita nesse trabalho tenha melhores resultados.

O modelo matemático proposto por Ferreira[1] tem como objetivo, calcular o tempo de estabilização de um software, com isso espera-se que após a implementação da formúla, o CONNECTA seja capaz de prever de forma aproximada esse tempo de estabilização em um software escrito em JAVA.

5 CONCLUSÕES E TRABALHOS FUTUROS

Oque falar aqui?

Falar sobre o projeto para o POCII e oque os resultados do POC I influenciaram nele.

Referências Bibliográficas

[1] FERREIRA, K. A. M. e. a. *Modularidade em Sistemas Orientados por Objetos*. São Paulo, SP: Pioneira Thomson Learning, 2007.

[2] [S.l.: s.n.].