UM MODELO DE PREDIÇÃO DE TEMPO DE MANUTENÇÃO DE SOFTWARE ORIENTADO POR OBJETOS

KECIA ALINE MARQUES FERREIRA

UM MODELO DE PREDIÇÃO DE TEMPO DE MANUTENÇÃO DE SOFTWARE ORIENTADO POR OBJETOS

Projeto de tese apresentado ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Ciências Exatas da Universidade Federal de Minas Gerais como requisito parcial para a obtenção do grau de Doutor em Ciência da Computação.

Orientador: Mariza Andrade da Silva Bigonha
Co-orientador: Roberto da Silva Bigonha
Colaborador: Bernardo Nunes Borges de Lima

Belo Horizonte

Agosto de 2009

Resumo

O objetivo deste trabalho é a definição de um modelo de predição de tempo de estabilização de software denominado KB3. Tempo de estabilização refere-se ao tempo médio necessário para realizar todas as manutenções decorrentes de manutenções realizadas em um determinado número de módulos no software. Este tempo é um indicador da manutenibilidade de software. O cálculo de KB3 é realizado com base em fatores de estrutura de software que têm impacto nessa característica: conectividade, acoplamento e coesão. Além de KB3, este trabalho propõe uma métrica para avaliação de coesão de classes denominada Coesão Contratual e apresenta os resultados de um estudo realizado para identificar valores referência para um conjunto de métrica de software orientado por objetos, em particular aquelas que podem ser utilizadas em KB3.

Palavras-chave: manutenibilidade de software, conectividade, orientação por objetos, métricas de software.

Abstract

The aim of this work is the definition of a prediction model of time to stabilization of software called KB3. Time to stabilization of software is the mean time needed to accomplish maintenances resulting from maintenances performed in a certain amount of modules. That time is a software maintainability indicator. KB3 computation is based on software structure factors that impact on maintainability: connectivity, coupling and cohesion. Besides KB3, this work proposes a cohesion metric called Cohesion by Contract and presents results of a study to identify reference values for a set of object-oriented software metrics, particularly those that can be used in KB3.

Keywords: software maintainability, connectivity, object-oriented software, software metrics.

Lista de Figuras

Lista de Tabelas

Sumário

Capítulo 1

Introdução

Software tornou-se um produto de grande importância social, cada vez mais complexo e sofisticado. Apesar da notável evolução tecnológica, desenvolver software de alta qualidade, com custo e prazo aceitáveis não é uma tarefa simples. Traduzindo a dimensão dos problemas envolvidos nessa atividade, Pressman [2006] os caracteriza como uma *aflição crônica*[1] vivenciada ainda hoje.

O produto *software* é particularmente complexo. A sua produção envolve uma gama de tecnologias, conceitos, métodos, técnicas, linguagens de programação, ferramentas e outros recursos disponíveis, fatores ambientais, como *hardware e outros softwares*, fatores econômicos, humanos, gerenciais etc. Trata-se de um produto não tangível, mas de profundo impacto nas atividades humanas atuais, que não se desgasta como os demais, porém deteriora-se. A Figura 1.1, clássica da literatura de Engenharia de Software, ilustra o processo de deterioração de software, que é decorrente das manutenções realizadas ao longo de sua vida. Cada manutenção realizada pode gerar efeitos colaterais ou necessidades de novas manutenções, aumentando a taxa de falhas do software. Antes de esse processo se estabilizar, o que significa que a taxa de falhas estabilizou-se, outra manutenção pode ser realizada, iniciando um novo aumento na taxa de falhas. Com isso, a taxa de falhas mínima do software tende a aumentar, o que caracteriza a sua deterioração. A complexidade da produção de software não se restringe ao seu processo de criação e à sua disponibilização para os usuário. Um grande desafio é amenizar sua deterioração.

Mais de de 70% do custo total de um sistema corresponde à atividade de manutenção [Meyer, 1997; Sommerville, 2003] apud Lientz e Swanson (1980)[2] e Nosek

[1] Pressman destaca o significado desta expressão: algo que causa dor ou sofrimento e que tem longa duração ou volta frequentemente.

[2] Lientz, B. P e Swanson, E. B. *Software maintenance management.* Reading, MA: Addison Wesley.

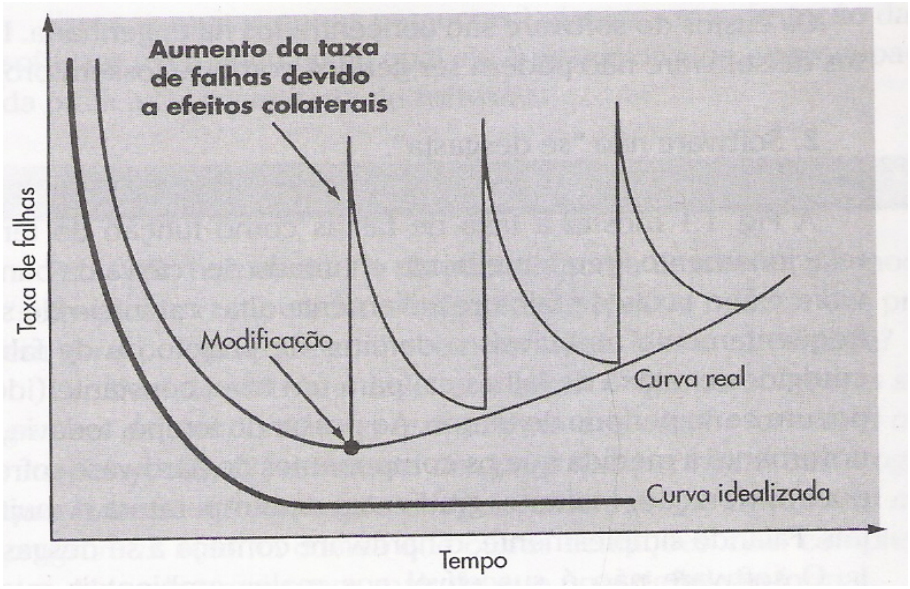


Figura 1.1. Curvas de falhas idealizada e real para software. Fonte: Pressman, 2002, p. 8.

e Palvia (1990)[3]. Desse custo, 17% corresponde a reparo de defeitos, 18% a adaptação do software a um novo ambiente e 65% a inclusão ou modificação de requisitos. Diante disso e do processo de deterioração potencial do software, é imperativo a construção de software com alto grau de *manutenibilidade*, definida como a facilidade de realizar manutenções em um software. Manutenções em software podem ser: corretivas, aquelas realizadas para corrigir defeitos no software; adaptativas, aquelas realizadas para adaptar o software a mudanças no seu ambiente, por exemplo, a suas regras de negócios; perfectivas, com o objetivo de aperfeiçoar o software; preventiva ou de reengenharia, com o objetivo de facilitar outras manutenções no software [Pressman, 2006; Filho, 2001].

Vários fatores influenciam a manutenibilidade de um software, por exemplo complexidade do domínio do problema, rotatividade de pessoal, qualidade da documentação, idade e estrutura do software [Pfleeger, 1998; Sommerville, 2003]. A complexidade do domínio do problema é inerente à aplicação, e a rotatividade de pessoal pode ser um problema comum nas equipes de desenvolvimento de software. Investir na qualidade de documentação e na qualidade estrutural do software são formas de se contornar ou amenizar as dificuldades de manutenção de software, inclusive no caso de dificuldade trazidas por rotatividade de pessoal e complexidade da aplicação. Sommerville [2003]

1980

[3]Nosek, J. T. e Palvia, P. (1990). *Software Maintenance management: changes in the last decade.* Software Maintenance: Reasearch and Practice, p. 157-174. 1990.

avalia que a atividade de manutenção de software é geralmente subestimada entre os engenheiros de software, que imaginam que esta atividade demanda menos habilidade e experiência do que a fase de desenvolvimento. O *Software Engineering Institute* (SEI), em seu *Software Technology Roadmap* [SEI, 2009], aponta como essencial que durante o desenvolvimento do software ocorra o planejamento e o preparo para a sua manutenção. O SEI identifica os seguintes pontos principais a serem considerados neste aspecto: a rastreabilidade de requisitos em código permite reduzir o tempo para se identificar o código que sofrerá manutenção; a documentação do software deve conter informações consistentes com o código para prover informações de suporte à manutenção; a possibilidade de medir diretamente a complexidade e a manutenibilidade de software permite a predição de custos e riscos futuros.

A medição, a avaliação e o controle da manutenibilidade do software são importantes durante as fases de desenvolvimento e operação para que problemas futuros graves sejam evitados. Muitos trabalhos têm essa questão como objeto de estudo [Chhabra & Aggarwal, 2006; Deissenboek et al., 2007; Ferreira, 2006; Gyimothy et al., 2005a; Heitlager et al., 2007; Hordijk & Wieringa, 2005; Nagappan et al., 2006; Santa'Anna et al., 2003; Schröter et al., 2006; Subrammanyam & Krishnan, 2003; Tsantalis et al., 2005; Zhou & Leung, 2006]. Myers [1975] propõe um modelo de estabilidade de programas que calcula o número médio de módulos de um software que sofrerão impacto em decorrência de uma manutenção em um módulos do software. Para isso, o modelo considera a coesão interna dos módulos e o grau de acoplamento, e utiliza um algoritmo aparentemente consistente, porém complicado. Ferreira [2006] adaptou esse modelo, inicialmente proposto para o paradigma estruturado, ao paradigma orientado por objetos. O modelo fornece uma informação importante, contudo necessita ser validado por meio de estudos experimentais. Um dos trabalhos mais importantes na medição de manutenibildiade é a métrica *Índice de Manutenibilidade* (MI) [Ash et al., 2006; Pearse & Oman, 9945]. Esta métrica provê um indicador do grau de manutenibilidade a partir de coleta de medidas de complexidade do código fonte do software. Foi proposta e validada com softwares estruturados, adotada pelo SEI e utilizada em organizações como Hewllet-Packard e Departamento de Defesa Americado (DoD) [SEI, 2009]. Porém, o uso dessa métrica tem sido criticado [Heitlager et al., 2007; Sarwar et al., 2008]. Uma observação que se pode fazer sobre MI é que o seu cálculo baseia-se em métricas como número de linhas código, número de linhas de comentários em código e complexidade. Essas métricas não capturam características de softwares orientados por objeto, para o qual há métricas específicas [Abreu & Carapuça, 1994; Chidamber & Kemerer, 1994]. Heitlager et al. [2007] afirmam que, com base em suas experiências com o uso de MI em um grande número de softwares de diversos propósi-

tos e tecnologias, observaram fortes limitações do uso desta métrica, dentre as quais destaca-se a dificuldade de se identificar o fator que causa determinado valor de MI, o que torna o seu uso difícil tanto em nível gerencial como técnico.

Apesar do grande investimento em estudos para identificar uma maneira de medir a manutenibilidade de software, ainda não há uma solução satisfatória para esse problema. Uma das causas que contribuem para este cenário é que a manutenibilidade é uma característica multidimensional, dificilmente capturada por uma única métrica. De acordo com a ISO 9126, essa característica envolve as seguintes subcaracterísticas: facilidade de identificação das partes do software que deverão ser modificadas e de realização de tais modificações, a capacidade do software evitar efeitos colaterais dessas modificações e a facilidade de testar o software modificado. Outra causa que pode ser observada é a dificuldade de validação das soluções propostas, devido à carência de dados experimentais de softwares reais.

Diante deste cenário, este trabalho tem por objetivo a definição de um modelo de predição de manutenibilidade de software. Assim como outras propostas nesta linha, a ideia desse modelo é que se possa obter, a partir da estrutura do software, por exemplo por meio de seu código, um número que indique a sua manutenbilidade e, como esse número, tanto gerentes quanto técnicos possam atuar no software de maneira a tornar a sua manutenção mais fácil. O modelo, denominado KB3, visa solucionar a seguinte questão: qual é o tempo médio de estabilização de um software com n módulos quando for necessário alterar i de seus módulos. Nesse modelo, um software é considerado estável quando todas as alterações decorrentes da primeira alteração foram concluídas. Neste modelo, o grau de interconexão entre os módulos do software, a sua *conectividade*, é considerada como o fator de principal impacto na manutenibilidade. O modelo proposto é genérico, podendo ser aplicado em diferentes paradigmas. Neste trabalho é enfatizada a orientação por objetos, por ser um dos paradigmas mais amplamente difundidos e utilizados atualmente. Além da conectividade, o modelo utiliza duas outras métricas de características classicamente reconhecidas como essenciais para a obtenção de boas estruturas de software: coesão interna de módulos e grau de acoplamento entre módulos. Para que o modelo possa ser efetivamente empregado, faz-se necessário identificar os valores referência das métricas utilizadas nele. Sem esta informação, a tomada de decisão no processo torna-se muito difícil. Neste contexto, identifica-se o seguinte problema: embora a importância de métricas na produção de software seja notória, elas não têm sido amplamente utilizadas na indústria. Tempero [2008] avalia que um dos motivos para esse fato é que para a maior parte das métricas de software ainda não se conhece o modelo de entidade da população (*entity population model*), o qual refere-se aos valores típicos de medidas que uma métrica assume em determinado

conjunto de entidades. Espera-se, com este trabalho, contornar estas dificuldades.

1.1 Objetivos

1.1.1 Objetivo Geral

A tese proposta neste trabalho tem por objetivo a definição de um modelo para predição de tempo médio t de estabilização de um software com n módulos quando i módulos sofrerem manutenção.

1.1.2 Objetivos Específicos

O trabalho proposto tem os seguintes objetivos específicos:

1. Proposta de um modelo probabilístico de predição de tempo de manutenção de software baseado em conectividade: partindo da idéia de que a conectividade é o fator que pode ser utilizado como indicador do grau de dificuldade de manutenção, será elaborado um Modelo Probabilístico de Predição de Tempo de Manutenção em Software, denominado KB3. O esforço de manutenção de software refere-se ao tempo de estabilização de um software a partir de uma alteração de um módulo deste software. Um software é considerado estável quando todas as alterações decorrentes da primeira alteração foram concluídas. O modelo a ser elaborado terá como entrada um software, representado por um conjunto de métricas coletadas nele. A saída do modelo é o tempo de estabilização do software.

2. Formalização do modelo proposto.

3. Avaliação do modelo proposto.

4. Validação da Métrica de Estabilidade de Myers: Myers [1975] propõe uma métrica para avaliar estabilidade de software estruturado. Ferreira [2006] adaptou esta métrica para o paradigma orientado por objetos. Esta métrica visa fornecer um indicador do impacto de uma alteração em módulo qualquer do software, porém não foi validada ainda. Os resultados obtidos com essa métrica serão utilizados, juntamente com os de MI, para comparação com os resultados fornecidos por KB3.

5. Proposta e validação de uma métrica para avaliação de coesão interna de módulos, denominada Coesão Contratual: coesão é um fator de grande importância na

manutenibilidade de software. Embora algumas métricas para este fator tenham sido propostas, ainda não há uma métrica que avalie confiavelmente a coesão. Dada a importância deste fator para a manutenibilidade, é essencial a existência de um indicador confiável para ele.

6. Evolução da ferramenta *Connecta* [Ferreira, 2006]: a ferramenta coleta um conjunto de métricas de software orientado por objetos em softwares desenvolvidos em Java. A ferramenta permite a gravação dos dados coletados em arquivos, o que dificulta a análise dos dados. Os dados passarão a ser armazenados também em banco de dados. Será implementada a coleta da métrica Coesão Contratual e o modelo proposto neste trabalho.

7. Identificação de valores referência para métricas de software orientado por objetos: esta ainda é uma questão em aberto na literatura, e sua solução é de grande valia no processo decisório na produção de software [Abreu & Carapuça, 1994; Gyimothy et al., 2005a; Li et al., 2006; Schröter et al., 2006]. A sua solução depende da análise de um largo número de softwares.

1.2 Contribuições

Os resultados do trabalho visam as seguintes contribuições principais:

1. Definição, formalização e avaliação de um modelo de predição de tempo de manutenção de software orientado por objetos.

2. Disponibilização da ferramenta *Connecta* para utilização do modelo proposto.

3. Identificação dos valores referência para as métricas de software orientado por objetos, em particular aquelas utilizadas no modelo proposto.

1.3 Metodologia

O trabalho proposto é de natureza teórica e experimental. Com base no raciocínio teórico sobre as relações entre os módulos de um software e sua manutenibilidade é possível chegar a três conclusões principais: a conectividade do software é determinante na sua manutenibilidade; há fatores da estrutura do software que contribuem para a facilidade de sua manutenção, por exemplo a coesão interna de seus módulos; há outros fatores que contribuem para a dificuldade de manutenção, por exemplo o grau de acoplamento entre seus módulos. A partir dessas conclusões e com base nos conceitos

de Probabilidades, em particular, Cadeias de Markov, foi definido o modelo KB3, que tem como entrada o software, representado por um conjunto de métricas, e como saída o tempo de estabilização do software frente ao número de módulos que sofrerão manutenção. A elaboração do modelo proposto envolve as seguinte etapas:

1. Definição formal do problema a ser solucionado pelo modelo.

2. Definição matemática do modelo: o modelo foi elaborado com base nos conceitos de Probabilidades. O processo de manutenção é modelado como uma Cadeia de Markov, na qual cada estado corresponde ao número de módulos em manutenção em determinado instante. No modelo, as probabilidades de mudança de estados são determinadas pela conectividade do software, por fatores que facilitam a manutenção e por fatores que a dificultam. Com isso, constroi-se uma matriz quadrada que representa as transições de estado na cadeia. A partir desta matriz e de um teorema que determina o tempo de estabilização de uma Cadeia de Markov com as características daquela utilizada no modelo, chegou-se a uma fórmula que envolve cálculos com variáveis simbólicas e matrizes, o que dificulta a sua compreensão e uso efetivo.

3. Implementação do modelo: com auxílio do software Matlab, a fórmula obtida foi implementada. Foram gerados e coletados resultados para softwares com número de módulos n variando de 4 a 25 módulos. Não foi possível gerar dados para softwares maiores porque o tempo de execução tornou essa tarefa inexequível. Como resultado, obteve-se, para cada software de tamanho n, para cada número de módulos i que sofrerão manutenção, um polinômio. Este polinômio fornece a informação buscada pelo modelo: $t = E(n, i)$, onde t é o tempo de estabilização, n é o número de módulos do software e i é o número de modulos nos quais serão inicialmente realizadas manutenções.

4. Análise preliminar dos polinômios obtidos e definição preliminar de KB3: com a análise preliminar dos polinômios obtidos, observou-se a existência de um padrão de formação entre eles. Contudo, a tentativa de identificação de um polinômio genérico não obteve sucesso em um primeiro momento. Desta forma, optou-se por representar o polinômio que fornece $t = E(n, i)$ por um de seus termos.

5. Implementação da métrica KB3 preliminar e experimentos: a implementação e experimentos preliminares com o polinômio obtido mostrou que essa fórmula não é eficaz, pois em muitos casos gerou números muito próximos de zero, de

difícil interpretação. Com isso, fez-se necessário uma nova análise dos polinômios obtidos.

6. Análise geral dos polinômios obtidos e definição de KB3: a análise geral dos polinômios resultou na identificação de uma fórmula genérica para $t = E(n, i)$.

7. Implementação da métrica genérica KB3: a métrica obtida será implementada na ferramenta *Connecta*.

8. Definição e implementação da métrica Coesão Contratual

9. Avaliação da métrica Coesão Contratual: a métrica Coesão Contratual será avaliada a partir de um estudo experimental que consistirá na coleta das medidas dessa métrica e de outra métrica amplamente conhecida para esse mesmo aspecto. Os resultados de ambas serão comparados à análise de um especialista sobre a qualidade estrutural das classes utilizadas no experimento.

10. Avaliação de KB3: o modelo proposto será avaliado por meio de experimentos com um conjunto de softwares.

11. Identificação de valores referência de métricas de software OO: foram coletadas medidas de um conjunto de métricas de software OO, especialmente aquelas potencialmente empregadas em KB3. Os dados utilizados neste estudo são de 40 softwares abertos desenvolvidos em Java. Os softwares foram obtidos de *www.sourceforge.net*. Foram utilizados softwares de 11 domínios de aplicações diferentes, classificados pelo próprio *site*, prefazendo o total de 26.202 classes. Para cada métrica, foi identifica a distribuição de probabilidades pela qual seus seus valores são modelados e, a partir disso e da análise dos dados obtidos, foram identificados valores que possam ser tomados com uma referência inicial.

1.4 Organização do trabalho

Além deste capítulo introdutório, este trabalho possui outras três partes: a primeira, que compreende os Capítulos 2, 3 e 4, apresenta uma revisão da literatura sobre assuntos relacionados ao tema desse trabalho; a segunda descreve nos Capítulos 5, 6 e 7 a solução proposta para atingir os objetivos traçados nesta tese; a terceira, referente ao Capítulo 8, apresenta o sumário da tese a ser elaborada e o cronograma de atividades a serem ainda desenvolvidas. Os conteúdos dos capítulos deste trabalho são:

- Parte 1: essa parte apresenta uma revisão da literatura relacionada ao assunto tratado neste trabalho. Inicialmente é apresentada uma revisão sobre métricas de software, abordando os seguinte aspectos: conceito e tipologia de métricas; principais métricas de software orientado por objetos; em particular, são aprofundados os estudos e análise de trabalhos sobre medição de coesão interna em software orientados por objetos; principais trabalhos e áreas de pesquisa em medição de software. Na sequência, é apresentado o conceito de manutenibilidade de software e suas subcaracterísticas. É realizada uma revisão da literatura sobre os principais trabalhos relacionados à avaliação da manutenibilidade de software. Essa parte abrange os seguinte capítulos:

  Capítulo 2: descreve os principais conceitos sobre a área de métricas de software, tais como sua relação com qualidade de software, classificações de métricas e princípios de medição.

  Capítulo 3: apresenta uma análise das métricas de software orientado por objetos e de ferramentas de coleta de medidas.

  Capítulo 4: avalia o estado da arte da área de medição de software, identificando os principais trabalhos realizados e necessidades de trabalhos futuros. Em particular, são estudados aqueles trabalhos relacionados à predição de tempo de manutenção de software.

- Parte 2: essa parte apresenta a solução proposta para alcançar os objetivos desta tese.

  Capítulo 5: apresenta a definição da métrica Coesão Contratual.

  Capítulo 6: apresenta um estudo realizado com o objetivo de identificar valores que possam ser considerados como referência para um conjunto de métricas de software orientado por objetos.

  Capítulo 7: define o modelo KB3 proposto nesta tese.

- Parte 3: essa parte apresenta o planejamento das atividades a serem realizadas para a conclusão desta tese.

  Capítulo 8: apresenta as atividades a serem desenvolvidas, os possíveis artigos a serem publicados como resultados dos estudos realizados neste trabalho, o sumário da tese final e o cronograma de atividades a serem realizadas.

Capítulo 2

Métricas de Software

Um dos primeiros livros publicados na área de métricas de software, escrito por Gilb [1977], data de meados da década de 70. Desde então, o número de métricas que têm sido propostas chega a algumas centenas. Como avaliam Fenton & Neill [2000], o desafio atual consiste em utilizar as métricas já propostas para construção de ferramentas de apoio à decisão. Este capítulo apresenta os principais conceitos relacionados à medição de software.

Três termos principais surgem na literatura sobre Métricas de Software: métrica, medida e medição. Embora estes termos sejam usados no mesmo sentido por alguns autores, eles têm significados diferentes, como definem von Staa [2000] e Pressman [2006]:

- *métrica (*metric*)*: é um padrão de medição para avaliar um atributo de algo relacionado a um software.

- *medida (*measure*)*: é o resultado da medição. Uma medida indica quantitativamente a presença de um atributo em determinado software.

- *medição (*measurement*)*: é o ato ou efeito de medir algo de acordo com uma métrica.

Métrica de software é o conjunto de regras que presidem a avaliação de um atributo de algo relacionado ao software. Uma métrica constitui um indicador que é usado pelos engenheiros de software para ajustar o produto, o projeto ou o processo.

Neste capítulo serão apresentados os seguintes conceitos relacionados à área de métricas de software: métrica e sua relação com qualidade de software, princípios de medição e categorias de métricas de software.

2.1 Métricas e Qualidade de Software

Pressman [2006] aponta como principal meta da Engenharia de Software a produção de software de alta qualidade e destaca o uso de medição como meio de avaliar a qualidade dos artefatos produzidos ao longo do ciclo de vida do software. A importância da medição de software para a sua qualidade é destacada também pela sua referência em modelos de qualidade mundialmente reconhecidos como o CMM *Capability Maturity Model* e a ISO 9000 [Kan, 2003].

O CMM [Humphrey, 1995] define cinco níveis de maturidade para uma organização desenvolvedora de software: o nível 1 é o *inicial*, caracterizado como caótico, no qual poucos processos são definidos e o sucesso depende de esforços individuais; no nível 2 os processos são *repetíveis* em projeto com aplicações similares; o nível 3, caracterizado como *definido*, o processo é padronizado, documentado e utilizado por toda a organização; o nível 4 é *gerenciável*, definido como quantitativo, havendo um controle estatístico da qualidade do produto; o nível 5 é o ideal, caracterizado como *otimizado*, no qual a base quantitativa é utilizada por exemplo para previsão de defeitos e investimento na automação de processos.

A ISO 9000 é um conjunto de padrões para a garantia da qualidade na gerência de sistemas. Ela define, entre outros requisitos, o uso de técnicas estatísticas para alcançar qualidade. Esse requisito abrange o uso de métricas de processo e de produto. No caso de métricas de processo, as medidas devem ser utilizadas para propósitos como identificar se o desenvolvimento do processo é efetivo na redução da probabilidade de ocorrência de falhas ou de não detecção dessas. No caso de métricas de produto, as medidas devem ser utilizadas para reportar falhas, agir de forma corretiva caso o nível da métrica seja considerado insatisfatório e estabelecer objetivos de melhoria em termos de métricas.

Qualidade de software é comumente definida como a conformidade do software como os seus requisitos. Esta definição, como aponta Kan [2003], é expressa numericamente por duas métricas principais: taxa de defeitos e confiabilidade. Um exemplo de métrica para taxa de defeitos é a quantidade de defeitos por quantidade de linhas de código. Um exemplo de métrica de confiabilidade é o número de falhas por horas de operação.

Entretanto a idéia da qualidade de software vista apenas como a conformidade com os seus requisitos não reflete de forma completa os seus aspectos. Na avaliação de Meyer [1997] a qualidade de software pode ser observada por fatores externos, aqueles do ponto de vista do usuário, e por fatores internos, aqueles do pontos de vista da estrutura do software. Outra classificaçao similar à de Meyer para fatores de qualidade

foi dada por McCall et al. [1] (1977 apud Pressman [2006]). Os fatores externos são:

- Correção: é a conformidade das tarefas realizadas pelo software com a sua especificação de requisitos.
- Robustez: é a característica de um software que realiza suas tarefas de forma correta mesmo quando submetido a situações anormais.
- Extensibilidade: é a característica de um software no qual pode-se facilmente incluir ou alterar requisitos.
- Reusabilidade: é a característica de um software que pode ser reutilizado ao todo ou em parte na construção de outros softwares.
- Compatibilidade: é a caracteríticas de um software que pode ser facilmente combinado com outros softwares.
- Eficiência: um software é dito eficiente se faz bom uso de recursos como memória e processadores.
- Portabilidade: refere-se à facilidade de se utilizar o software em diferentes ambientes de hardware e software.
- Verificabilidade: é a facilidade de se verificar se um software está em conformidade com a sua especificação de requisitos.
- Integridade: é a capacidade de proteger componentes como dados, programas e documentos contra acessos não autorizados.
- Facilidade de uso: é a facilidade com que o software pode ser utilizado.

Os fatores internos são determinantes para atingir os fatores externos. Dentre eles destaca-se a modularidade, que é a característica de um software construído por unidades elementares denominadas módulos. Um software é construído de forma modular quando seus módulos são o mais independente possível uns dos outros, de maneira 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. Para isso é necessário que o projeto de um software seja direcionado a manter baixo grau de relacionamento entre módulos, o que se denomina acoplamento, e alto grau de

[1] McCALL, J.; Richards, P.; Walters, G. *Factors in Software Quality.* three volumes, NTIS AD-A049-014, 015, 055, November 1977.

relacionamento entre os elementos internos de um módulo, o que se denomina coesão [Myers, 1975].

O que Betrand Meyer denomina fator de qualidade, Gilb[2] (1988 apud Pressman [2006]) denomina medida de qualidade de software. Gilb determina as seguintes medidas de qualidade para software:

- Correção: é o grau com o que o software desempenha sua função. Uma métrica para isso é o número de defeitos por KLOC (*kilo lines of code*, mil linhas de código). Defeito aqui refere-se a uma inconformidade de requisito.

- Manutenibilidade: é definida como a facilidade como que um software pode ser corrigido, adaptado ou aperfeiçoado. Uma métrica para manutenibilidade é MTTC (*mean-time-to-change*, tempo médio para modificação) que é dada pelo tempo gasto para analisar a manutenção a ser realizada, projetar adequadamente as alterações e adaptações necessárias, testar e distribuir o software para os usuários. Altos valores para MTTC indicam grande dificuldade de manutenção do software. Outra métrica usada para avaliar manutenibilidade é o custo para corrigir defeitos após a entrega do software, também denominada prejuízo.

- Integridade: é a capacidade do software resistir a ataques ao seus dados, programas e documentos. Sendo *ameaça* definida como a probabilidade de ocorrência de um ataque e *segurança* a probabilidade de um ataque ser repelido, a integridade é dada pela somatório por tipo de ataque de [(1 - ameaça) X (1 - segurança)].

- Utilização: é a quantificação da facilidade de uso de um software.

De acordo com Pressman [2006], há fatores que podem ser medidos diretamente e há fatores que podem ser medidos apenas indiretamente. Mas em todos os casos, medições devem ocorrer para que se possa chegar a uma indicação de qualidade do software. McCall et al. [3] (1977 apud Pressman [2006]) propõem que um conjunto de métricas sejam utilizadas para avaliar os fatores de qualidade de acordo com a seguinte relação: $F_q = c_1 X m_1 + c_2 X m_2 + ... + c_n X m_n$), onde F_q é um fator de qualidade, c_i é um peso dado a um métrica e m_i é o valor da métrica. As métricas que podem ser avaliadas apenas subjetivamente têm seus valores graduados em uma escala de 0 a 10, que indicam, respectivamente, valores baixos e altos para a métrica. Dentre as métricas propostas por McCall et al. destacamos a *modularidade*, definida como o grau de

[2]GILB, T. *Principles of Software Project Management.* Addison-Wesley, 1988.

[3]McCALL, J.; Richards, P.; Walters, G. *Factors in Software Quality.* three volumes, NTIS AD-A049-014, 015, 055, November 1977.

independência funcional entre os módulos do software, pois, de acordo com os autores, serve para avaliar indiretamente a maior parte dos fatores de qualidade: confiabilidade, manutenibilidade, flexibilidade, testabilidade, portabilidade, reutilização e interoperabilidade. Na definição de Meyer [1997], confiabilidade corresponde a robustez, flexibilidade corresponde a extensibilidade, testabilidade corresponde a verificabilidade, portabilidade tem o mesmo significado nas duas propostas, e interoperabilidade corresponde a compatibilidade. Manutenibilidade é definida genericamente como o esforço necessário para localizar e eliminar um erro no software. Além da modularidade, as seguintes métricas são indicadas para medir a manutenibilidade de um software:

- Autodocumentação: grau em que o código fonte possui documentação significativa.
- Concisão: indica quão compacto é um software em termos de número de linhas de código.
- Consistência: uso uniforme das técnicas de documentação e projeto ao longo de todo o desenvolvimento do software.
- Instrumentação: grau em que o software monitora sua própria operação e é capaz de identificar as falhas ocorridas.
- Simplicidade: grau de facilidade com que o software pode ser compreendido.

2.2 Princípios de Medição

Medição de software é realizada para a obtenção de dados para que possa: obter entendimento quantitativo; avaliar e controlar um produto, um processo ou uma organização; realizar estimativas e planejamentos [Humphrey, 1995]. Encontram-se na literatura duas classificações principais para medições: uma dada por Humphrey [1995] e a outra dada por Kan [2003]. A seguir estas duas classificações serão abordadas.

Humphrey [1995] define as seguinte categorias principais para medição:

Objetiva ou subjetiva: uma medida objetiva é quantitativa. Uma medida subjetiva baseia-se no julgamento humano, por exemplo *bom, ruim, baixo, alto, aceitável, satisfatório, etc.*

Absoluta ou relativa: uma medida absoluta não varia em decorrência da adição de novos itens. O contrário ocorre quando uma medida é relativa. O tamanho de

um programa é um exemplo de uma medida absoluta, já a média de tamanhos de programas é uma medida relativa.

Explícita ou derivada: uma medida explícita é obtida de forma direta, enquanto uma medida derivada é obtida a partir da combinação de outras medidas.

Dinâmica ou estática: uma medida dinâmica varia com o tempo, o que não ocorre com uma medida estática. Número de defeitos por mês é uma medida dinâmica, pois seu valor varia de acordo com o mês. LOC (número de linhas de código) e número total de defeitos são exemplos de medidas estáticas.

Preditiva ou explanatória: uma medida explanatória é obtida após a ocorrência de um fato, enquanto a obtenção de uma medida preditiva é adiantada, ou seja, ocorre antes do fato.

Kan [2003] classifica medições em quatro níveis: escala nominal, escala ordinal, escala de intervalo e escala de taxa.

Escala nominal: também denominada classificação, é a escala mais simples utilizada. Consiste em classificar um conjunto de elementos em categorias. As categorias são mutuamente exclusivas e o conjunto de todas as categorias deve cobrir todo o universo de discussão. Neste tipo de escala os nomes e as seqüência das categorias não definem relações entre as categorias.

Escala ordinal: este tipo de escala é um nível maior do que a escala nominal. Na escala ordinal, os elementos podem ser comparados de acordo com uma ordem. Por exemplo, uma organização pode ser classificada em nível no CMM; uma família pode ser classificada de acordo com sua situação sócio-econômica. Neste caso é possível não apenas categorizar um grupo de elementos mas também ordenar as categorias. A relação entre as categorias é simétrica, ou seja, se $A > B$ então B não é maior do que A, e transitiva, ou seja, se $A > B$ e $B > C$, então $A > C$.

Escala de intervalo: uma escala de intervalo indica a diferença entre dois pontos de medida. A operações matemáticas aplicáveis a intervalos de dados são adição e subtração.

Escala taxa: é o maior nível de medição e todas as operação matemáticas são aplicaveis, inclusive multiplicação e divisão. Por exemplo, se um software A tem 4 defeitos por KLOC e B tem 2 defeitos por KLOC, pode-se dizer que A tem duas vezes mais defeitos que B.

O tipo de medição a ser utilizada em um contexto depende da aplicação desejada. Sobre a qualidade da medição, dois critérios mais importantes são identificados: validade e confiabilidade. A validade refere-se a se a métrica realmente mede o que ela pretende medir. Confiabilidade refere-se à consistência da medição e do método empregado na medição [Kan, 2003].

2.3 Categorias de Métricas de Software

Métricas de software são classificadas em três categorias: métricas de processo, métricas de projeto e métricas de produto [Pressman, 2006; Kan, 2003; Humphrey, 1995].

Métricas de processo: permitem a organização avaliar o processo empregado. Humphrey [1995] define processo como a seqüência de passos necessários no desenvolvimento ou na manutenção de software; o processo define as técninas, métodos gerenciais, ferramentas e as pessoas às tarefas relacionadas à produção do software. Métricas de processo são utilizadas para promover melhorias no desenvolvimento e na manutenção de software.

Métricas de projeto: métricas de projeto permitem avaliar o andamento de um projeto; descrevem as características de um projeto e de sua execução. Número de desenvolvedores, custo, cronograma e produtividade são exemplos de métricas de projeto. Estas métricas são denominadas por Humphrey [1995] como métricas de recursos.

Métricas de produto: descrevem as características de produtos de software. São exemplos dessas características tamanho, desempenho e complexidade.

Um subconjunto de métricas de software são as denominadas métricas de qualidade de software, que têm foco na qualidade de produto, processo e projeto. Métricas de qualidade de software subdividem-se em métricas de qualidade de produto final e métricas *in-process*. A *Engenharia de Qualidade de Software* tem por objetivo investigar a relação entre métricas *in-process*, características de projeto e métricas de qualidade de produto final, construindo melhorias em processo e produto com base nos dados obtidos. Além disso, uma vez que a qualidade deve ser avaliada por todo o ciclo de vida de um software, faz-se necessária a utilização de métricas que avaliem a qualidade do processo e de manutenção de software [Kan, 2003]. Estas categorias de métricas serão discutidas a seguir.

2.3.1 Métricas de Qualidade de Produto

A qualidade do produto de software é usualmente medida em função da quantidade de erros, defeitos e falhas do mesmo. Um erro é resultado de uma ação humana que resulta no funcionamento incorreto do software; um defeito é uma anomalia no produto; uma falha ocorre quando uma unidade do software não é capaz de desempenhar a sua função (IEEE/ANSI 982.2). Kan [2003] exemplifica as seguintes métricas de qualidade produto:

1. Tempo médio de falha (MTTF - *mean time to failure*): mede o tempo médio entre falhas de um software.

2. Densidade de defeitos: medida da quantidade de defeitos de um sofware em relação ao seu tamanho. O tamanho pode ser dado por exemplo por número de linhas de código, quantidade de pontos de função, etc.

3. Problemas de usuários: denotam os defeitos encontrados na utilização do software por seus usuários. É dada pelo *total de problemas reportados pelo usuário em período / total de número de licenças-mês durante o período*, onde *número de licenças-mês = número de licenças instaladas do software X número de meses no período.*

4. Satisfação de usuário: frequentemente medida via uma escala de cinco pontos: *muito satisfeito, satisfeito, neutro, insatisfeito, muito insatisfeito*. A partir desta escala, muitas outras métricas podem ser obtidas, por exemplo: percentual de usuários insatisfeitos, percentual de usuários satisfeitos, etc.

2.3.2 Métricas de Qualidade de Processo

Partindo da idéia de que os erros identificados durante a fase de teste indicam que erros foram introduzidos no software durante seu desenvolvimento, Kan [2003] exemplifica as seguintes métricas de qualidade processo (métricas *in-process*).

1. Densidade de defeitos durante testes: uma boa métrica para essa densidade é o número de defeitos por KLOC ou pontos de função.

2. Remoção de defeitos baseada em fases: é uma extensão da métrica de densidade de defeitos para as demais fases do ciclo de vida de sistema. Reflete a capacidade geral de remoção de defeitos por fases do processo de desenvolvimento.

2.3.3 Métricas para Custo de Manutenção de Software

A fase de manutenção do ciclo de vida de um software inicia-se quando o desenvolvimento de um software é finalizado e este entra em operação. Kan [2003] descreve métricas para a fase de manutenção que baseiam-se na identificação de defeitos por intervalo de tempo e chamados de problemas de usuários, que podem significar defeitos ou não. As métricas para a fase de manutenção descritas por Kan [2003] são apresentadas a seguir.

Acúmulo de consertos(*fix backlog*): é uma soma simples de problemas reportados que permanecem sem consertos no final de cada período, por exemplo, cada semana ou cada mês.

Índice gerencial de acúmulo de consertos: BMI (*backlog management index*) é dada pela razão entre *o número de problemas resolvidos durante o período* e *o número de problemas identificados durante o período* multiplicado por 100. Se BMI é maior do que 100, significa que o acúmulo de problemas foi reduzido no período, caso contrário, o acúmulo de problemas aumentou.

Tempo de resposta de consertos: é o tempo médio de resposta entre a abertura de um problema e o seu fechamento. A abertura do problema ocorre quando ele é identificado; o seu fechamento ocorre quando ele é solucionado.

Qualidade de consertos: do ponto de vista do usuário é ruim que um software apresente defeitos e é igualmente ruim, ou ainda pior, se uma manutenção em um software gera defeitos no mesmo. Uma manutenção gera defeitos se ela não corrige o problema inicialmente identificado ou se ela corrige o problema, mas introduz um novo defeito. Isso é denominado por Kan como conserto defeituoso. A métrica de qualidade de consertos é dada pelo percentual de todos os consertos defeituosos realizados em um intervalo de tempo.

Estas métricas são coletadas já na fase de manutenção do software. Entretanto, é importante a possibilidade de antecipação aos problemas de manutenção, pois esta fase é a mais cara no ciclo de vida do sistema. A identificação de uma métrica, ou de um conjunto de métricas, que possa indicar o esforço na fase de manutenção de um software em fases precoces é imperativo. Com isso, problemas que acarretariam dificuldades na fase de manutenção poderiam ser identificados e sanados precocemente, reduzindo o custo de manutenção. As Seções 4.4 e 4.5 descrevem estudos que têm sido realizados nesse sentido.

2.4 Panorama

Qualidade de software é uma das principais questões na produção de software. Métricas e qualidade de software são duas áreas que caminham juntas. Métricas têm papel fundamental na qualidade, pois fornecem aos engenheiros de software dados quantitativos que permitem avaliar processos, projetos e produtos, controlá-los, melhorá-los e planejá-los.

A importância que é dada ao assunto pode ser avaliada pela quantidade de trabalhos e publicações na área. As primeiras publicações datam de meados dos anos 70 [Gilb, 1977]. Um levantamento realizado por Xenos et al. [2000] conta dezenas de métricas propostas na literatura, dentre as quais a mais famosa talvez seja LOC (número de linhas de código). Fenton & Neill [2000] avalia que o desafio da área de medição de software é utilizar métricas simples, já propostas, para a construção de ferramentas de apoio à decisão no desenvolvimento de software.

Nos próximos capítulos são identificadas as principais métricas propostas na literatura para software orientado por objetos, bem como as principais linhas de trabalhos realizadas atualmente nesta área.

Capítulo 3

Métricas de Software Orientado por Objetos

Um software orientado por objetos é muito diferente de software construído no paradigma estruturado. Desta forma, métricas utilizadas para avaliar este tipo de software não são suficientes e algumas delas são inadequadas para avaliar software orientado por objetos. Como exemplo, podemos citar a métrica LOC (*lines of code*), que refere-se ao número de linhas de código de um software, uma das mais conhecidas métricas para indicar tamanho de software. No caso de um software OO, esta métrica fornece um nível de informação muito baixo. Métricas de software OO devem avaliar especificamente as características deste paradigma, tais como: coesão interna de classes, acoplamento entre classes, ocultação de informação e herança.

Dois dos conjuntos de métricas mais referenciados na literatura são aquele proposto por Chidamber e Kemerer [Chidamber & Kemerer, 1994], denominado conjunto CK, e aquele proposto por Abreu & Carapuça [1994], denominado MOOD. Ambos foram propostos em 1994, mas são independentes. Este capítulo apresenta estes dois conjuntos de métricas e duas métricas propostas para avaliação de estabilidade de sistema: a de Martin [1994] e a de Myers [1975] adaptada por Ferreira [2006], bem como um conjunto de ferramentas de coletas de métricas em sistemas OO.

3.1 Métricas CK

Chidamber & Kemerer [1994] propõem um conjunto de métricas para projeto orientado por objetos, referenciadas na literatura como métricas CK. Nesse trabalho, os autores apresentam seis métricas, validam-nas usando os critérios de avaliação de métricas propostos por Weyuker [1988] e relatam os resultados de experimentos reali-

zados com as métricas propostas. Para a realização dos experimentos, foram utilizados dois sistemas, um escrito em C++ e outro em Smalltalk. O conjunto CK é constituído pelas seguintes métricas: Métodos Ponderados por Classe (WMC - *Weighted Methods per Class*), Profundidade de Árvore de Herança (DIT - *Depth of Inheritance Tree*), Número de Filhos (NOC - *Number of Children*), Acoplamento entre Classes de Objetos (CBO - *Coupling between Object Class*), Resposta de Classe (RFC - *Response for a Class*) e Ausência de Coesão em Métodos (LCOM - *Lack of Cohesion in Methods*).

- *WMC* (Métodos Ponderados por Classe)*:* é uma métrica que representa a complexidade da classe por meio de seus métodos. O cálculo da métrica é dado pelo somatório das complexidades dos métodos que constituem a classe. Fica em aberto a definição para complexidade. Os autores não determinam um cálculo específico da complexidade dos métodos visando tornar a aplicação desta métrica mais geral. Segundo Chidamber e Kemerer, esta métrica é um indicador de custo de desenvolvimento e manutenção de uma classe, assim como do grau de reúso da classe. A quantidade de métodos de uma classe e a complexidade de tais métodos constituem um indicador do esforço de manutenção da classe. Além disso, classe com um grande número de métodos têm potencial de reúso limitado, pois tendem a ter um uso específico da aplicação da qual fazem parte.

- *DIT* (Profundidade de Árvore de Herança)*:* indica a posição de uma classe na árvore de herança de um software, que é dada pela distância máxima da classe até a raiz da árvore. Essa métrica é considerada um indicador da complexidade de desenho e de predição do comportamento de uma classe, visto que quanto maior a profundidade da classe na árvore de herança, mais classes, e portanto mais métodos e atributos, estarão envolvidos na análise.

- *NOC* (Número de filhos)*:* indica a quantidade de sub-classes imediatas de uma classe. É um indicador da importância que uma classe tem no sistema, pois quanto mais sub-classes possuir uma classe, maior a importância de seu teste no sistema.

- *CBO* (Acoplamento entre Classes de Objetos)*:* é um totalizador do número de classes às quais uma determinada classe está acoplada. Para Chidamber e Kemerer, o acoplamento entre duas classes existe quando métodos de uma delas usa métodos ou variáveis de instância da outra. A razão da existência desta métrica é justificada pelos autores pela necessidade de redução de acoplamento entre classes de objetos para atender fatores como melhoria de modularidade e aumento de reusabilidade.

- *RFC* (Resposta de Classe)*:* apresenta o resultado do número de métodos que podem ser executados em resposta a uma mensagem recebida por um objeto da classe. Este resultado é dado pela quantidade de métodos da classe somada à quantidade de métodos invocados por cada método da classe. Visto que RFC considera a ativação de métodos de outras classes, ela é, como CBO, um indicador de conectividade de uma classe. Enquanto CBO mostra a quantas outras classes uma classe está conectada, RFC é um detalhamento desta informação, pois apresenta por quantos caminhos uma classe está conectada a outras classes.

- *LCOM* (Ausência de Coesão em Métodos)*:* é uma métrica da ausência de coesão entre os métodos de uma classe. Chidamber e Kemerer consideram que a coesão entre os métodos de uma classe é definida pela similaridade entre eles. A avaliação da similaridade entre dois métodos é determinada pelo uso de variáveis de instância da classe por eles. Seja P o conjunto formado pelos pares de métodos que não possuem variáveis de instância em comum e Q o conjunto formado pelos pares de métodos que possuem variáveis de instância em comum. O cálculo de LCOM é dado por:

  $LCOM =\mid P \mid - \mid Q \mid$, se $\mid P \mid > \mid Q \mid$
  $LCOM = 0$, caso contrário

  Por exemplo, seja uma classe que possua dez pares de métodos sem variáveis de instância em comum e dois pares de métodos com variáveis de instância em comum. Assim, P = 10 e Q = 2. Então, LCOM = P - Q = 10 - 2 = 8. Uma classe que possua dez pares de métodos sem variáveis de instância em comum e quatro pares de métodos com variáveis de instância em comum. Assim, P = 10 e Q = 4. Então, LCOM = P - Q = 10 - 4 = 6. Em comparação com a classe do exemplo anterior, essa classe apresenta melhor grau de coesão entre os seus métodos, o que é indicado pelos valores da métrica LCOM obtidos para ambas.

  LCOM indica a diferença entre a quantidade de pares de métodos sem similaridade, isto é, pares de métodos que não possuem variáveis de instância em comum, e a quantidade de pares de métodos com similaridade. Baixos valores para essa métrica indicam bom nível de similaridade, portanto de coesão, entre os métodos da classe avaliada. Um valor alto para LCOM indica que a classe não provê uma funcionalidade bem específica.

3.2 Métricas MOOD

O conjunto de métricas MOOD (Metrics for Object Oriented Design) foi proposto por Abreu & Carapuça [1994]. As métricas MOOD avaliam os aspectos de herança, ocultação de informação, acoplamento, polimorfismo e reusabilidade em um software orientado por objetos. Compõem o conjunto MOOD as seguintes métricas: Fator Herança de Método (MIF - *Method Inheritance Factor*), Fator Herança de Atributo (AIF - *Attribute Inheritance Factor*), Fator Acoplamento (COF - *Coupling Factor*), Fator Agrupamento (CLF - *Clustering Factor*), Fator Polimorfismo (PF - *Polymorphism Factor*), Fator Ocultação de Método (MHF - *Method Hiding Factor*) , Fator Ocultação de Atributo (AHF - *Attribute Hiding Factor*), Fator Reúso (RF - *Reuse Factor*).

O cálculo de uma métrica MOOD é dado por uma razão na qual o numerador é o número de ocorrências encontradas no sistema para o aspecto avaliado e o denominador é o maior número possível de ocorrências no sistema para tal aspecto. Desta forma, o resultado de qualquer métrica MOOD é sempre um valor entre 0 e 1, o que representa o nível de ocorrência do aspecto avaliado no sistema. Esse tipo de resultado é apropriado porque fornece uma dimensão para a métrica independente do tamanho do sistema avaliado, o que torna possível comparar sistemas que possuam tamanhos e características distintos. A seguir, são apresentadas as principais métricas do conjunto MOOD.

1. *Métricas Para Avaliação de Herança*

   Herança é o recurso da orientação por objetos que permite criar classes a partir de classes já existentes. Por meio da herança obtém-se a reutilização de estruturas que já estão definidas e possivelmente depuradas e testadas, o que é um ponto considerável na redução do custo da produção do software. Porém, outro aspecto deve ser observado em relação a herança, como apontam Chidamber & Kemerer [1994]: árvores de herança muito profundas conferem maior complexidade ao software, o que é um fator negativo para a sua manutenção.

   Um indicador que permita a avaliação da herança em um sistema é, então, um instrumento útil para a predição do esforço de sua manutenção. As seguintes métricas para este aspecto fazem parte de MOOD: MIF (Fator Herança de Métodos) e AHF(Fator Herança de Atributos). Elas são descritas a seguir.

   - *MIF (Fator Herança de Método):* essa métrica indica o percentual de métodos herdados no sistema. Seja C_i uma classe do sistema a ser avaliado. Para a definição da métrica MIF, são considerados as seguintes métricas básicas:

- Métodos herdados: $M_h(C_i)$. São os métodos que uma classe possui em decorrência de herança e que não foram redefinidos na classe.
- Métodos novos: $M_n(C_i)$. São métodos criados na classe, que não foram herdados nem redefinidos.
- Métodos redefinidos: $M_r(C_i)$. São métodos herdados que têm uma redefinição na classe.
- Métodos definidos: $M_d(C_i)$. Englobam os métodos novos e os métodos redefinidos na classe.
- Métodos disponíveis: $M_{dis}(C_i)$. É a totalidade de métodos que uma classe possui, o que engloba métodos definidos nela e os métodos herdados por ela.
- Total de classes do sistema: TC.
- Total de métodos definidos no sistema: é o somatório do número de métodos definidos em cada classe do sistema, que englobam tanto os métodos novos das classes quanto os redefinidos nelas. Esse total é dado pela Equação 3.1.

$$TM_d = TM_n + TM_r = \sum_{k=1}^{TC} M_d(C_k) \tag{3.1}$$

- Total de métodos novos definidos no sistema: é o somatório do número de métodos novos de cada classe do sistema, dado pela Equação 3.2.

$$TM_n = \sum_{k=1}^{TC} M_n(C_k) \tag{3.2}$$

- Total de métodos novos redefinidos no sistema: é o somatório do número de métodos redefinidos em cada classe do sistema, dado pela Equação 3.3.

$$TM_r = \sum_{k=1}^{TC} M_r(C_k) \tag{3.3}$$

- Total de métodos herdados no sistema: é o somatório do número de métodos herdados em cada classe do sistema, dado pela Equação 3.4.

$$TM_h = \sum_{k=1}^{TC} M_h(C_k) \tag{3.4}$$

- Total de métodos disponíveis no sistema: é o somatório do número de métodos disponíveis em cada classe do sistema, dado pela Equação 3.5.

$$TM_{dis} = \sum_{k=1}^{TC} M_{dis}(C_k) \tag{3.5}$$

O cálculo de MIF é realizado da seguinte forma: para cada classe do sistema, verifica-se a quantidade de métodos herdados e a quantidade de métodos disponíveis. O valor de MIF é dado pela razão entre o somatório do número de métodos herdados de cada classe do sistema e o somatório do número de métodos disponíveis de cada classe do sistema. Assim, o cálculo de MIF é dado pela Equação 3.6

$$MIF = \frac{TM_h}{TM_{dis}} \tag{3.6}$$

MIF com valor igual a 0 indica que no sistema em questão não houve utilização efetiva do recurso de herança de métodos, o que significa que não existe relacionamento algum de herança entre as classes do sistema ou se existe, todos os métodos herdados foram redefinidos. Valores de MIF próximos de 1 indicam alta utilização do recurso de herança de métodos no sistema. Um valor igual a 1 para MIF indica que todos os métodos disponíveis em todas as classes do sistema são herdados. Esta situação parece estranha, mas sua ocorrência é possível, visto que uma métrica pode ser utilizada para avaliar um conjunto de classes particular de um sistema.

MIF indica, portanto, se o recurso de herança de métodos foi explorado amplamente no sistema, o que é um instrumento na predição no custo de manutenção do sistema. Quanto menor o valor de MIF, maior o custo de manutenção do sistema, pois valores baixos para esta métrica indicam que há pouca reutilização de métodos já definidos e possivelmente depurados e testados.

- *AIF (Fator Herança de Atributo):* indica o percentual de atributos herdados no sistema. Um raciocínio similar ao realizado no cálculo do fator herança de métodos é realizado para o fator herança de atributos AIF. O valor de AIF é dado pela razão entre o somatório do número de atributos herdados de cada classe do sistema e o somatório do número de atributos disponíveis de cada classe do sistema. Assim, o cálculo de AIF é dado pela Equação 3.7, onde TA_h é o total de atri- butos herdados no sistema e TA_{dis} é o total

de atributos disponíveis no sistema. O cálculo de TA_h e TA_{dis} são similares aos de TM_h e que são dados pelas Equações 3.4 e 3.5 respectivamente.

$$AIF = \frac{TA_h}{TA_{dis}} \tag{3.7}$$

As conclusões a cerca dos valores desta métrica são si- milares às referentes à métrica MIF: valores próximos de 0 indicam pouca utilização do recurso de herança de atributos e o oposto, valores próximos de 1, indicam boa utilização de tal recurso. Entretanto, a importância da métrica AIF é menos relevante do que a da métrica MIF, pois, como apontam Abreu & Carapuça [1994], o custo de manutenção dos métodos das classes que compõem o sistema tem peso muito maior no custo de manutenção do sistema do que os custos de manutenção decorrentes dos atributos das classes.

2. *Métricas Para Avaliação de Ocultação de Informação*

A ocultação de informação é um conceito importante relacionado à modularidade, pois a sua aplicação potencializa a independência de módulos. Quanto mais as informações e os serviços de uma classe estiverem confinados dentro dela, menor é a necessidade de as demais classes conhecerem sua organização interna e mais fraco é o nível de interdependência entre elas. Uma classe deve ser conhecida somente pelos serviços que ela disponibiliza. Na orientação por objetos, a ocultação de informação é obtida pelo uso de atributos e métodos privados nas classes.

Uma métrica de ocultação de informação em um sistema é um indicador que influencia a avaliação da modularidade do sistema porque reflete quão restritas estão as informações pertencentes aos módulos do software. As seguintes métricas para avaliação de ocultação de informação em sistemas orientados por objetos fazem parte de MOOD: MHF (Fator Ocultação de Métodos) e AHF (Fator Ocultação de Atributos). Elas são descritas a seguir.

- *MHF (Fator Ocultação de Método):* esta métrica representa o percentual de métodos ocultos no sistema. Para o seu cálculo, as seguintes métricas básicas são definidas, considerando-se C_i uma classe qualquer do sistema a ser avaliado.

  - Métodos visíveis: $M_v(C_i)$. São os métodos que constituem a interface da classe.
  - Métodos ocultos: $M_o(C_i)$. São os métodos privados da classe.

- Métodos definidos: $M_d(C_i)$. São os métodos visíveis mais os métodos ocultos da classe. Essa métrica é dada pela Equação 3.8.

$$M_d(C_i) = M_v(C_i) + M_o(C_i) \tag{3.8}$$

MHF é a razão entre o número de métodos ocultos em todas as classes e o número de métodos definidos em todas as classes, dado pela Equação 3.9

$$MHF = \frac{\sum_{k=1}^{TC} M_o(C_k)}{\sum_{k=1}^{TC} M_d(C_k)} \tag{3.9}$$

Valores próximos de 1 para a métrica MHF indicam um alto nível de ocultação de métodos das classes do sistema. Esse tipo de resultado reflete que, de uma forma geral, as classes do sistema exportam poucos serviços, o que deve propiciar baixa conectividade entre as classes do sistema. O contrário ocorre quando se obtém valores próximos a 0 para essa métrica, o que indica que as classes do sistema exportam muitos serviços, portanto favorecem alto grau de conectividade entre as classes do sistema.

- *AHF (Fator Ocultação de Atributo):* essa métrica é o percentual de atributos ocultos no sistema. Similarmente a MHF, o cálculo de AHF é dado pela razão entre o número de atributos ocultos em todas as classes e o número de atributos definidos em todas as classes, conforme a Equação 3.10. Nesta equação, A_o corresponde ao número de atributos ocultos na classe e A_d, ao número de atributos disponíveis na classe; o cálculo de A_d é similar ao de M_d, que é dado pela Equação 3.8.

$$AHF = \frac{\sum_{k=1}^{TC} A_o(C_k)}{\sum_{k=1}^{TC} A_d(C_k)} \tag{3.10}$$

A ocultação de atributos é característica de extrema importância para garantir a independência entre módulos, pois impossibilita a ocorrência dos tipos mais graves de acoplamento que podem existir entre duas classes. Quando uma classe torna público um atributo, outras classes do sistema podem alterar o valor desse dado e, então, perde-se a garantia da sua integridade e estabelece-se uma forte dependência entre todas as classes que fazem uso

de tal atributo. Conhecer o grau de ocultação de informação de atributos de um sistema é saber quão propenso é o surgimento de acoplamentos desse tipo no sistema.

Valores de AHF próximos a 1 indicam que poucos atributos no sistema em questão são públicos. A situação ideal é que nenhum atributo seja público, o que resulta em AHF igual a 1. O pior caso é um valor igual a 0 para esta métrica, que indica que todos os atributos de todas as classes do sistema são públicos.

3. *Métricas Para Acoplamento*

Métricas que possibilitam análise sobre os acoplamentos existentes em um sistema são úteis na predição do custo da manutenção do mesmo. MOOD contém as seguintes métricas para acoplamento: COF (Fator Acoplamento) e CLF(Fator Agrupamento). A métrica COF fornece um indicador do grau de conectividade do software.

- *COF (Fator Acoplamento):* para a avaliação de acoplamento, Abreu & Carapuça [1994] consideram o conceito de relação *cliente-servidor* entre as classes constituintes de um software. Segundo esse conceito, uma classe A é cliente de uma classe servidora B quando A referencia pelo menos um membro de B, seja este membro um atributo ou um método. Uma relação cliente-servidor entre duas classes corresponde à existência de uma conexão entre elas.

  Em um software com n classes, o maior número possível de conexões é $n^2 - n$. A métrica COF é dada pela razão entre o número total de conexões existentes entre as classes do software e o maior número possível de conexões para o software. Um software totalmente conectado possui COF = 1.

  COF é uma métrica importante, pois indica quão conectado é um software. Um software fortemente conectado possui estrutura rígida, baixo grau de independência entre os módulos e, conseqüentemente, o custo na sua manutenção é explosivo.

3.3 Análise das Métricas CK e MOOD

Os dois conjuntos de métricas, CK e MOOD, visam a avaliação quantitativa de sistemas orientados por objetos, porém com abordagens distintas. As métricas CK

avaliam fatores de classes no sistema, por exemplo, a falta de coesão da classe, o número de filhos de uma classe, a profundidade na árvores de herança de uma classe, etc. Já as métricas MOOD buscam avaliar não classes particulares no sistema, mas o sistema como um todo ou parte dele. O emprego das duas abordagens é complementar, pois na avaliação de um sistema estes dois níveis de avaliação - sistema e classe - são importantes. A primeira permite avaliar condições gerais do sistema, a segunda permite detalhar o nível da avaliação.

Nenhum dos dois conjuntos possui métricas para avaliação de aspectos como estabilidade de um software. Para este aspecto, destaca-se a métrica proposta por Martin [1994]. Uma outra forma de avaliar estabilidade de software foi proposta por Myers [1975], porém para o paradigma estruturado. Esta proposta foi adaptada ao paradigma orientado por objetos por Ferreira [2006]. A Seção 3.4 apresenta estas duas propostas.

Em especial, métricas para avaliação de coesão interna de classe apresentam maior grau de dificuldade em sua elaboração do que os demais fatores, pois a avaliação de coesão pode depender da compreensão do domínio do problema do sistema, cuja avaliação automática é muito difícil. Isso tem sido tema de trabalhos atuais nesta linha, conforme é discutido na Seção 4.1.

3.4 Métrica de Estabilidade

Gilb [1977] define a estabilidade como a capacidade de um sistema manter-se inalterado diante de uma alteração em seu ambiente. Para Martin [1994] também a estabilidade está relacionada a esta habilidade: quando um módulo é indepedente dos demais, alterações nestes não demandam alterações no primeiro, garantindo sua estabilidade. Pressman [2006] dá um significado mais amplo para estabilidade, definindo-a como a característica de um software que se recupera bem de falhas e no qual modificações não são frequentes, são controladas e não invalidam os testes realizados. Avaliamos que os conceitos não são distantes, mas complementares, pois a estabilidade, tal como é definida por Gilb e Martin, é que permite atingir as características apontadas por Pressman.

A seguir são apresentadas duas métricas para avaliação de estabilidade em sistemas orientados por objetos: a primeira, proposta por Martin [1994], mede o grau de *instabilidade* de um software, e a segunda, denominada *métrica de estabilidade*, proposta originalmente por Myers [1975] para avaliação de softwares no paradigma estruturado e adaptada por Ferreira [2006] para avaliar software orientado por objetos.

3.4.1 Métrica de Instabilidade

Martin [1994] alerta que quando a extensão de uma alteração em um sistema não é predizível, o seu impacto não pode ser estimado. Com base neste problema, ele propõe uma métrica de *instabilidade* que baseia-se na análise de dependências entre o que ele chama de categorias de classes. Uma categoria de classes é um grupo coeso de classes no qual: se uma classe da categoria é alterada, as demais classes da categoria têm grandes chances de serem alteradas; as classes são reutilizadas em conjunto; as classes têm um objetivo em comum ou realizam funções semelhantes.

Para Martin, a independência e a estabilidade de uma categoria de classes podem ser medidas a partir do número de conexões da categoria com classes externas a ela. Desta forma, a métrica de instabilidade I é definida da seguinte maneira: $I = Ce/(Ca + Ce)$, onde:

- Ca: são os acoplamentos aferentes (*afferent couplings*). Corresponde ao número de classes externas à categoria que estão conectadas às classes da categoria.
- Ce: são os acoplamentos eferentes (*efferent couplings*). Corresponde ao número de classes da categoria que estão conectadas a classes externas à categoria.

A métrica representa a razão entre os acoplamentos eferentes da categoria e total de acoplamentos nos quais a categoria está envolvida. Os valores da métrica de instabilidade vão de 0 a 1. Martin aponta que quando I é igual a zero, indica que a categoria está com a estabilidade máxima; um valor igual a 1, indica que a classe tem instabilidade máxima.

3.4.2 Métrica de Estabilidade

Myers [1975] propõe *um modelo de estabilidade de programas* que resulta em métricas que fornecem os seguintes indicadores: quantos módulos serão alterados em decorrência da alteração de um módulo qualquer no sistema; quantos módulos serão alterados em decorrência da alteração de um módulo específico do sistema. Para alcançar isso, o sistema é modelado como um grafo não direcionado no qual os vértices representam os módulos do sistema e as arestas representam as conexões entre os módulos. Para cada módulo, deve ser avaliado o grau de sua coesão, assim como para cada conexão avalia-se o grau do acoplamento envolvido entre os módulos. Estas avaliações seguem uma escala proposta pelo autor, como mostra a Tabela 3.1.

A ideia do cálculo desta métrica é obter para cada par de módulos a probabilidade de um ser alterado em decorrência da alteração do outro. O cálculo desta probabilidade

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

Acoplamento	*Valor*
Conteúdo	0,95
Dado comum	0,7
Externo	0,6
Controle	0,5
Dado local	0,35
Informação	0,2

Tabela 3.1. Pesos de Coesão e Acoplamento no Modelo de Myers

Acoplamento	*Valor*
Conteúdo	0,95
Dado comum	0,7
Inclusão	0,7
Externo	0,6
Controle	0,5
Referência	0,35
Informação	0,2

Coesão	*Valor*
Coincidental	0,95
Lógica	0,4
Temporal	0,6
Procedimental	0,4
Comunicacional	0,25
Contratual	0,2

Tabela 3.2. Pesos de Acoplamento e Coesão na OO

considera o grau de coesão dos módulos envolvidos, o grau de acoplamento direto e indireto entre eles. O algoritmo para o cômputo desta métrica [Myers, 1975; Ferreira, 2006] é complexo se comparado ao da métrica de Martin.

A métrica de estabilidade de Meyers foi adaptada por Ferreira [2006] ao paradigma OO. A adaptação consiste nos seguintes pontos:

- A métrica original modela o sistema como um grafo não direcionado. Ferreira considera que o modelo que melhor representa as conexões existentes entre módulos de um sistema é um grafo direcionado, pois o fato de uma alteração em um módulo A acarretar uma alteração em B, não implica que uma alteração em B também gera alteração em A.

- Ferreira revisou os tipos de acoplamento e coesão na orientação por objetos, o que resultou nos valores mostrados na Tabela 3.2.

- Na métrica original, não é definido como proceder no caso de um módulo estar acoplado a outro por mais de uma forma. Na adaptação de Ferreira, considera-se a conexão que envolve acoplamento mais forte, de acordo com a escala definida na Tabela 3.2

3.4.3 Avaliação das Métricas de Estabilidade

A métrica de Instabilidade de Martin fornece uma avaliação da instabilidade de categorias de classes de um sistema e não de classes isoladas, tampouco do sistema como um todo. No que tange ao nível de informação fornecida, a métrica de estabilidade de Myers adaptada por Ferreira é mais abrangente, pois avalia a estabilidade do sistema e de classes particulares. Ambas métricas têm base conceitual sólida. A primeira parte do princípio de que é possível avaliar a estabilidade de um conjunto de classes a partir da quantidade de conexões deste conjunto com as demais classes do sistema. A segunda métrica considera, além da quantidade de conexões, o grau de acoplamento entre os módulos bem como o grau de coesão interna dos módulos. Porém, estas métricas necessitam ser experimentalmente validadas.

3.5 Ferramentas de Coleta de Métricas

A aplicação de métricas de software requer o uso de ferramentas que as coletem. Algumas das ferramentas que possuem este propósito são apresentadas a seguir.

Understand [Understand, 2007] é uma ferramenta comercial que coleta métricas em softwares escritos em Ada, Delphi, Fortran, Java e C++. Entre as métricas coletadas pela ferramenta estão: número de classes, número de linhas de código, número de linhas de comentários e número de linhas de declarações.

Krakatau Essencial Metrics [Krakatau, 2006] é uma ferramenta comercial que coleta métricas em programas escritos em Java e C/C++. A sua principal funcionalidade é prover meios de comparações de versões de software. Baseia-se na coleta das seguintes métricas: linhas de código alteradas, adicionadas e excluídas. Apesar de ser uma ferramenta que vise a análise de programas OO, não provê métricas específicas a esse paradigma.

A ferramenta comercial *ObjectDetail* [ObjectDetail, 2006] coleta métricas específicas do paradigma OO em softwares desenvolvidos em C++. Dentre as métricas coletadas, destacam-se: profundidade da árvore de herança, acoplamento de classe, que é o número de classes que uma classe particular usa, e percentuais de atributos públicos, de atributos privados, de métodos públicos e de métodos privados.

MOODKIT [Abreu et al., 1995] é uma ferramenta desenvolvida pelo grupo de estudos relacionado à proposta do conjunto de métricas MOOD. Esta ferramenta coleta as métricas MOOD em softwares escritos em linguagens como C++, Eiffel e Java. A principal característica desta ferramenta é o uso de uma linguagem intermediária denominada GOODLY, proposta pelo mesmo grupo. A idéia basica é converter o

código fonte a ser analisado em um código equivalente GOODLY, sobre o qual ocorre a coleta das métricas.

Dependency Finder [DependencyFinder, 2007] é uma ferramenta *open source* que permite a análise de código compilado Java, sendo disponível para ambientes Windows, Unix e Web. Fornece o grafo de dependência entre classes e a coleta de um conjunto de métricas OO em nível de métodos, classes, grupos de classes e sistema. Dentre as métricas do conjunto CK e MOOD, coleta apenas a métrica de profundidade da árvore de herança.

JDepend [JDepend, 2007] é uma ferramenta *open source* que coleta métricas de pacotes de classes em Java, tais como número de classes e interfaces, acoplamentos aferentes, que são o número de classes externas ao pacote que usam as classes do pacote, e acoplamentos eferentes, que são o número de pacotes dos quais as classes do pacote dependem.

Metrics [Metrics, 2007] coleta métricas em software implementado em Java. É um *plugin* para o Eclipse. Coleta várias métricas orientadas por objetos, dentre as quais estão: acoplamento aferente, acoplamento eferente e instabilidade para pacotes, LCOM e DIT. Inclui um analizador gráfico de dependências entre pacotes de classes.

Connecta [Ferreira, 2006] é uma ferramenta que coleta métricas em software JAVA, a partir da análise de *bytecode*. A ferramenta coleta as seguintes métricas: estabilidade, COF, CBO, DIT e LCOM, além de apontar os graus de acoplamentos entre as classes do sistema. Os resultados são apresentados de forma que se possa identificar possíveis pontos de melhoria no sistema: inicialmente são reportados os valores da estabilidade e COF; a partir desses valores, pode-se detalhar o nível de avaliação por classes do sistema, obtendo-se os valores das métricas CBO, DIT, LCOM e grau de acoplamento entre classes. Uma das melhorias a serem realizadas na ferramenta é a inclusão de um recurso que permita a visualização dos resultados de forma gráfica.

Embora exista uma quantidade grande de ferramentas de coleta de métricas disponíveis no mercado e na academia, nenhuma delas apresenta todos os requisitos ideais de uma ferramenta desta categoria. Tais requisitos envolvem os seguintes aspectos, dentre outros:

- *Coleta de métricas de fato orientada por objetos*: algumas das ferramentas disponíveis, embora coletem métricas em software orientado por objetos, não coletam métricas relevantes específicas da orientação por objetos, como é o caso de Understand e Krakatau.

- *Integração com o ambiente de implementação*: este requisito é importante para que o próprio desenvolvedor possa avaliar a qualidade do código que está pro-

duzindo com facilidade. Metrics provê esta facilidade.

- *Possibilidade de execução fora do ambiente de implementação*: permite que o código seja avaliado de forma independente do ambiente de execução. Isso facilita a coleta e análise de métricas por gerentes, analistas, etc., sem a necessidade do uso do ambiente de implementação. Connecta e JDepend, por exemplo, funcionam desta forma.

- *Exportação de dados*: o recurso de exportação de dados coletados permite que estes sejam posteriormente manipulados e analisados. Metrics, por exemplo, permite exportação de dados em formato XML.

- *Armazenamento de histórico*: o armazenamento dos dados coletados é essencial para a tarefas como a análise comparativa entre resultados de versões distintas do mesmo software e entre softwares diferentes.

- *Multilinguagem*: um requisito desejável em uma ferramenta de coleta de métricas é que ela opere sobre softwares desenvolvidos em linguagens diferentes. Connecta, Metrics, JDepend, Dependency Finder funcionam somente para código Java.

- *Visualização gráfica das dependências entre os módulos do software*: uma das métricas mais importantes da orientação por objetos é a que mede o grau de conectividade do software. Das ferramentas listadas nesta seção, somente MOODKIT e Connecta coletam tal métrica. Um recurso facilitador para identificação de módulos críticos no sistema em relação ao fator conectividade é a visualização gráfica das dependências entre os módulos. Metrics possui este recurso, porém considera como módulos os pacotes do sistema e não as suas classes.

O ideal é que existisse uma ferramenta que agregasse todos esses requisitos. O que observa-se é que, ao realizar uma pesquisa que exija a coleta de métricas, os pesquisadores acabam desenvolvendo suas próprias ferramentas. A ausência desses requisitos nas ferramentas prejudica também o seu emprego na indústria.

Capítulo 4

Pesquisas em Medição de Software

A área de pesquisa em métricas de software orientado por objetos é um campo fértil. O interesse pela área abrange questões como a proposta de novas métricas de software, a identificação de necessidades de refatorações por meio de métricas, e a utilização de métricas como intrumentos de predição de falhas, esforço de manutenção e estabilidade de software. Este capítulo identifica os principais assuntos que têm sido discutidos na literatura, analisando alguns dos trabalhos realizados e identificando necessidades de trabalhos futuros na área.

4.1 Medição de Coesão

Coesão em software foi definida por Myers [1975] como o grau de intercomunicação entre os elementos internos de um módulo. Myers apresentou a escala de grau de coesão interna de módulos que desde então foi adotada e aceita pela comunidade[1]. Essa escala é qualitativa e baseia-se na observação e na análise do relacionamento entre os elementos de um módulo. Um módulo cujos elementos não apresentam relacionamento algum possui o pior grau de coesão, denominada coesão coincidental; um módulo que realiza uma única função bem definida possui coesão funcional, sendo o melhor tipo de coesão.

Avaliar quantitativamente a coesão de um módulo é difícil, pois dizer se os elementos de um módulo possuem relacionamento ou desempenham uma única função bem definida muitas vezes envolve conhecer o domínio do problema. Como apontam Counsell et al. [2004], esta tarefa tem sido tema de muitos trabalhos pelo fato de a coesão ser um fator de grande importância na compreensão e na manutenção de software;

[1] Alguns autores atribuem a Yourdon e Constantine essa definição de coesão e sua respectiva escala em 1979. Porém, esse conceito foi proposto quatro anos antes, por Glenford Myers.

em particular, a comparação matemática de propriedades de métricas de coesão está em curso. Esta seção apresenta discute trabalhos realizados para avaliação de coesão interna em software orientados por objetos.

A métrica mais referenciada para este fator é LCOM (1994) proposta por Chidamber & Kemerer [1994], descrita na Seção 3.1. A métrica baseia-se no conceito de que coesão de uma classe é avaliada pelo grau de similaridade entre seus métodos. Esta métrica foi revisada posteriormente por: Bieman e Ott (1994), Henderson-Sellers (1996) e Briand et al. (1998) [Counsell et al., 2004].

Bansiya[2](1999 apud Counsell et al. [2004]) propôs a métrica CAMC (*cohesion among methods in a class*), uma métrica de coesão baseada no fato de que uma classe é considerada coesa se seus métodos usam o mesmo conjunto de tipos de parâmetros. CAMC é dada pela média das entradas na chamada matriz de ocorrência de parâmetros. Esta matriz é uma tabela na qual as linhas correspondem aos métodos da classe e as colunas, aos tipos de parâmetros de todos os seus métodos. Uma entrada na tabela é dada da seguinte forma: para cada método, marca-se 1 nas posições correspondentes aos tipos de parâmetros que ele recebe. Counsell et al. [2004] avalia que esta abordagem tem a seguinte vantagem principal em relação àquela utilizada em LCOM: pode ser vista como uma métrica de projeto, podendo ser utilizada em estágios iniciais do desenvolvimento de software. Os autores de CAMC identificaram forte relação entre LCOM e CAMC, concluindo que, dado que esta pode ser usada na fase de projeto, é melhor utilizar CAMC do que utilizar LCOM.

Marcus & Poshyvanyk [2005] propõem medir a coesão de classes baseada na análise de informação semântica no código fonte, tais como comentários e identificadores. O cálculo da métrica baseia-se em técnicas de recuperação de informação avançadas. A métrica C3, denominada *coesão conceitual*, mede o grau em que os métodos de uma classe estão conceitualmente relacionados. Foi realizado um estudo de caso em que a coleta da métrica proposta em um software de código aberto foi comparado aos resultados de outras métricas de coesão já propostas na literatura, entre elas LCOM; Marcus & Poshyvanyk [2005] verificaram que os resultados obtidos pelas duas métricas são coerentes. Entrentanto, vale ressaltar que são necessários mais estudos para comprovar este fato. Como os próprios autores de C3 avaliam, o sucesso do cálculo da métrica é totalmente dependente da existência de comentários e nomes significativos no corpo do código fonte.

Counsell et al. [2004] avaliaram três métricas de coesão em software OO: CAMC (*cohesion among methods in a class*), e duas métricas propostas pelos autores, NHD

[2]Bansyia, J., Etzkorn, L. *A class cohesion metric for object-oriented designs.* Journal Object-Oriented Program. 11, 8, 47-52. 1999.

(*normalised Hamming distance*) e SNHD (*scaled normalised Hamming distance*). NHD (2002) e SNHD (2006) seguem o mesmo conceito de coesão utilizado em CAMC. A idéia do cálculo destas métricas é que uma classe cujos métodos compartilham uma proporção grande de tipos de parâmetros é mais coesa do que uma cujos métodos compartilham uma baixa proporção dos seus tipos de parâmetros. SNHD reflete quão perto o valor de NHD de uma classe está do valor máximo ou do mínimo possível de NHD. O estudo comparativo de Counsel et alli teve por objetivo investigar a eficiência das métricas SNHD e NHD em relação a CAMC. Para isso, foram coletadas as três métricas em três sistemas desenvolvidos em C++. Os resultados do estudo concluem que, assim como LCOM, as três métricas são dependentes do tamanho da classe. Com base no conceito dos autores sobre o que é coesão na OO, eles concluem que SNHD e NHD são mais úteis do que CAMC, mas alertam que é preciso refinar o que deve ser considerado como coesão na OO. Como trabalho futuros, são apontados: realização de testes extensivos em sistemas inteiros; estabelecimento dos valores referência das métricas.

Mäkelä & Leppänen [2007] propõem a medição de coesão externa de classe, com base no fato de que para o cliente de uma classe, a sua representação interna não é tão relevante quanto a sua interface. A métrica, denominada ELCOM (*external lack of cohesion in methods*) refere-se à forma como classes clientes de uma classe c, em um contexto C, usa os serviços de c. Formalmente, a métrica é definida como:

$$ELCOM(c) = 1 - (|CU(c)|/(|CM(c)||V(c)|)) \tag{4.1}$$

onde, $CM(c)$ é o conjunto de classes clientes de c, $V(c)$ é o conjunto de variáveis de instância de c, e $CU(c)$ é o conjunto de pares ordenados pertencente ao produto cartesiano entre $CM(c)$ e $V(c)$. Para avaliar a aplicabilidade da métrica, foi realizado um estudo de caso com 4 software de código aberto; no estudo foram coletadas a métricas ELCOM e LCOM. A métrica ELCOM fornece uma evidência que sugere o grau de coesão interna da classe, porém, como destacam os autores, é necessária uma avaliação mais aprofundada da classe para afirmar a qualidade de sua estrutura.

Avaliação

Há outras referências para métricas de coesão na literatura [Marcus & Poshyvanyk, 2005; Mäkelä & Leppänen, 2007]. Não há um consenso na literatura sobre uma métrica padrão para coesão. Algumas das necessidades de trabalhos a serem realizados nesta linha são: estudos experimentais em sistemas

maiores, a fim de verificar a aplicabilidade das métricas de coesão propostas em qualquer sistema; contraposição dos valores obtidos para as métricas com a análise realizada por desenvolvedores de software; definição de valores ou faixas de valores a serem considerados satisfatórios para o fator coesão.

4.2 Medição de Software Orientado por Aspectos

No caso da Programação Orientada por Aspectos (POA), tem sido discutido [Griswold et al., 2006; Kiczales & Mezini, 2005; Wand, 2003] o seu impacto na modularidade e no *raciocínio modular*, que refere-se à compreensão de um módulo e a tomada de decisão sobre ele conhecendo-se apenas sua implementação, sua interface e as interfaces dos módulos referenciados por ele [Kiczales & Mezini, 2005]. Os relatos nesses trabalhos baseiam-se em conceitos e aspectos qualitativos. Há uma carência de estudos que avaliem quantitativamente os impactos do uso da orientação por aspectos (OA) na manutenibilidade de software. No Brasil, uma das instituições que tem investido esforços neste sentido é a PUC-Rio. Esta seção discute dois dos trabalhos realizados por pesquisadores desta instituição.

Santa'Anna et al. [2003] propõem um modelo para avaliação de manutenibilidade e reusabilidade de software orientado por aspectos e realizam estudos experimentais. O modelo consiste em um conjunto de fatores considerados como determinantes da manutenibilidade e reusabilidade desse tipo de software, como separação de interesses, tamanho, coesão e acoplamento, bem como a definição de um conjunto de métricas para avaliação de tais fatores. O estudo experimental relatado nesse trabalho utiliza duas versões de um software: uma que utiliza padrões de projeto na orientação por objetos (OO) e outra que utiliza OA. No experimento, um grupo de implementadores realizaram sete tipos de alterações nos softwares em questão e, para as alterações realizadas, foram coletadas as seguintes medidas: quantidade de componentes de software adicionados, número de componentes alterados, número de relações entre componentes incluídas, número de linhas de código incluídas e alteradas. Santa'Anna et al. [2003] afirmam que os resultados dos experimento mostram que os fatores utilizados no modelo, como acoplamento, número de componentes e separação de interesses, tem impacto direto na manutenibilidade e na reusabilidade. Entretanto, esta relação não está claramente apresentada nos resultados. Os dados apresentados mostram que, em algumas alterações, o software OO demandou um número maior de inclusão de linhas de código, relacionamentos e de alterações de operações. Esse fato poderia indicar que alterar software OA é mais fácil do que alterar software OO, porém não evidencia que as

métricas de acoplamento, número de componentes e separação de interesses podem ser utilizadas para avaliar a manutenibilidade e reusabilidade de software OA. Sant'anna et al. destacam, ainda, os seguintes pontos que ficam em aberto:

1. o estudo que realizaram considera como esforço de manutenção a quantidade linhas de código incluídas ou alteradas. Faz-se necessário utilizar medidas mais representativas para o aspecto manuntenibilidade, por exemplo, o tempo para realizar modificações no software;

2. o experimento restringiu-se à comparação de dois softwares apenas: um desenvolvido com OA e outro com OO;

3. o tamanho e a complexidade limitada dos softwares analisados não permitem que os resultados alcançados no experimento sejam generalizados para outros softwares.

Kulesza et al. [2006] apresentam um estudo de caso que compara a manutenibilidade de software OA e OO. No estudo, foi utilizado um sistema real baseado na Web, sendo que uma versão utiliza OO e a outra utiliza OA. As métricas utilizadas na avaliação de manutenibilidade são as mesmas utilizadas por Santa'Anna et al. [2003]. As versões originais implementam 13 casos de uso. O experimento consiste em incluir 8 casos de uso ao sistema, que demandam alterações em classes das quatro camadas. As métricas foram coletadas nas versões originais e nas versões alteradas. Na avaliação de Kulesza et al. [2006], os resultados mostram superioridade da OA em relação à OO nos fatores acoplamento, tamanho e separação de interesses. O contrário ocorre no caso de coesão. Uma causa apontada para isso é que a métrica utilizada para avaliação de coesão, que é uma extensão da métrica LCOM, pode não aferir corretamente esse fator. De fato, conforme discutido na Seção 4.1, o problema de medição de coesão em software OO não está satisfatoriamente solucionado e isso vale para a OO também, já que a métrica para avaliação de coesão neste paradigma baseia-se em LCOM.

Avaliação

Os dois estudos discutidos nesta seção baseiam-se em métricas de fatores como acoplamento, coesão e tamanho para avaliar a manutenibilidade de software OA. Não avaliam, por exemplo, o tempo necessário para realizar as manutenções. Outro fato importante a ser destacado é que as métricas utilizadas avaliam componentes do software, tais como classes e aspectos, e não o software como um todo. Os valores das medidas possivelmente foram sumarizados, por exemplo, por meio de uma média. O

problema de se fazer isso é que as verdadeiras características do software podem ser mascaradas, caso os valores dessas métricas apresentem características de *power-laws*. Em distribuições de valores com essa característica, a média não é significativa para representar a população, conforme discutido na Seção 6.1.

4.3 Reestruturação de Software

Refatoração de software (do inglês *software refactoring*) foi introduzida por Fowler [1999] como a alteração da estrutura do software sem afetar o seu comportamento com o objetivo de melhorar a sua estrutura. O processo de realização de uma refatoração consiste em identificar qual parte do software será reestruturada, escolher uma refatoração apropriada como solução e aplicá-la. Identificar pontos de necessidade de refatoração em software, sobretudo nos de grande porte, pode ser uma tarefa inviável. Métricas de software têm sido utilizadas para este propósito.

Pizka [2004] descreve um estudo de caso em que utilizou métricas de software e ferramentas para auxiliar a reestruturação de um software comercial com o objetivo de melhorar sua estrutura. O ambiente de manutenção do software apresentava problemas tais como: instabilidade de requisitos, pressões sofridas pela equipe em relação a prazos para execução das tarefas de manutenção, ausência de documentação e pouca experiência dos desenvolvedores. De acordo com Pizka, o código tornou-se um *(hyper)spaghethi-code*. Motivado por isso, o responsável pelo software aderiu ao experimento, que foi realizado durante 5 meses. Foram realizadas as seguinte tarefas no experimento: indentificação de problemas no código utilizando cinco ferramentas de coleta de métricas; seleção e realização das reestruturações necessárias; avaliação dos benefícios das refatorações. Dentre as métricas utilizadas estão LCOM, RFC e CBO, do conjunto CK [Chidamber & Kemerer, 1994]. Um dos problemas identificados durante o experimento é a dificuldade de se utilizar as ferramentas e o grande tempo que elas demandam para realizar a coleta das métricas. As medidas obtidas para o software mostraram-se insuficientes para identificar necessidades de reestruturações, o que demandou análise subjetiva do código. Algumas das ferramentas utilizadas auxiliam na refatoração automática de software, porém, na avaliação de Pizka [2004], não são suficientemente amadurecidas para esta tarefa. Pizka descreve que, após o experimento, o gerente, os desenvolvedores e ele próprio consideram que não houve melhoria significativa na estrutura do software, porém esta avaliação não foi suportada por dados numéricos.

Marinescu[3] (2002 apud Munro [2005]) em sua tese de doutorado utilizou métricas de software para definir estratégias de identificar 14 tipos de problemas de desenho e classes que demandam refatorações em código fonte de software OO. Entretanto, como analisa Munro [2005], a escolha das métricas não foi satisfatoriamente justificada. O trabalho de Munro [2005] estende o de Marinescu. Munro utilizou métricas de software para identificar automaticamente *bad smell* em código fonte de software OO. Um *bad smell* é informalmente definido por Fowler como problemas de desenho em software, por exemplo, uma classe pouco coesa. Em seu trabalho, Munro seleciona um conjunto de métricas de software que podem ser utilizadas para identificar um sub-conjunto de *bad smells* em código fonte Java. O arcabouço proposto por ele é consituído por:

Nome do *bad smell*: a descrição informal do problema, definida por Fowler [1999].

Processo de medição: descrição das técninas de medição que podem identificar o problema em código Java.

Interpretação: indica um conjunto de regras a serem aplicadas para definir como as métricas podem ser utilizadas para identificar os candidatos à refatoração.

Munro [2005] apresenta a sua proposta para dois *bad smells*: classe *lazy*, caracterizada por ser uma classe que não realiza tarefas relevantes e por isso deve ser eliminada; campo temporário, caracterizado como uma variável de instância em um objeto que é atualizada somente em certas circunstâncias. A proposta foi avaliada a partir de dois estudos de caso: um software de 1500 linhas de código, 13 classes e 124 métodos; outro de 16000 linhas de código, 730 métodos e 84 classes. Embora a pesquisa tenha mostrado resultados que indicam a eficiência do uso de métricas com o propósito de identificar pontos de refatoração em sofware, ainda há pontos importantes a serem estudados, como:

- estender a proposta para os demais problemas de desenho de classe;
- realizar estudos de caso em sistemas de grande porte e reais;
- investigar se há uma métrica, ou um conjunto de métricas, que possa ser utilizado na identificação dos problemas de desenho de classes.

[3]Marinescu, R.. *Measurement and Quality in Object-Oriented Design.* Tese de doutorado. Universidade de Timisoara. Outubro de 2002.

Avaliação

Dado o fato da tendência à deterioraçao de softwares, reestruturação de software é essencial para amenizar essa situação. Um dos desafios nessa tarefa é identificar as partes do software que demandam melhorias. Métricas de software têm sido utilizadas para isso. Porém, esse parece ser um campo ainda pouco amadurecido. Dois pontos importantes, ressaltados pelo estudo de Pizka [2004] são: a dificuldade do uso efetivo de ferramentas durante o processo de refatoração; além disso, o estudo evidencia que o uso das métricas geralmente utilizadas com esse propósito, como as do conjunto CK [Chidamber & Kemerer, 1994], não são eficientes para identificar necessidades de refatoração, o que pode ser causado pelas próprias definições das métricas ou pela forma como as ferramentas as computam.

4.4 Predição de Falhas de Sistemas

Saber antecipadamente as chances que um sistema tem de falhar no futuro é essencial para avaliar a sua confiabilidade. Trabalhos nesta linha têm sido realizados, por exemplo o de Nagappan et al. [2006] e o de Gyimothy et al. [2005b], descritos a seguir.

O trabalho de Nagappan et al. [2006] tem por objetivo responder o que leva um software a falhar, identificando fatores que possam ser utilizados como instrumentos de predição de falha para um grupo amplo de softwares. O termo *falha*, aqui, refere-se a um erro observado no comportamento do software. Esse trabalho parte das seguintes hipóteses:

1. o aumento no valor das métricas de uma entidade do software, como um módulo, um arquivo ou algum outro componente, está relacionada ao número de falhas nesta entidade do software;

2. há um conjunto de métricas para o qual a hipótese 1 aplica-se a todos os softwares;

3. há uma combinação de métricas que indicam as falhas de novas entidades introduzidas no software;

4. indicadores obtidos a partir de tais métricas em um projeto podem ser utilizados na predição de falhas de entidades de outros softwares.

Visando atingir o objetivo do trabalho, Nagappan et al. realizaram um estudo empírico baseado na coleta de métricas em códigos fontes de cinco softwares produzidos pela Microsoft, desenvolvidos no paradigma orientado por objetos, nas linguagens C++

e C#. Foram coletadas métricas como: número de linhas executáveis em cada método, número de métodos em uma classe, número de superclasses de uma classe, número de classes acopladas a uma classe, entre outras. Os resultados foram comparados com uma base de dados históricos de falhas nos referidos softwares. Os resultados desse estudo foram:

1. para cada software analisado, foi encontrado um conjunto de métricas correlacionadas com as falhas no software, o que confirma a primeira hipótese;

2. a segunda hipótese não foi confirmada, pois os resultados dos experimentos não evidenciam um conjunto comum de métricas que possa ser utilizado como predição de falhas em todos os projetos;

3. a terceira hipótese foi confirmada, pois verificou-se que os indicadores obtidos de um componente principal podem ser utilizados na construção de modelos de regressão para estimar falhas em novas entidades;

4. a quarta hipótese foi parcialmente confirmada com os experimentos, pois verificou-se que os indicadores obtidos em determinado software podem ser utilizados apenas na predição de falhas em softwares similares e não em qualquer software.

De acordo com Nagappan et al. [2006], o trabalho deles avança o estado da arte nos seguintes pontos: é um dos primeiros trabalhos a mostrar como construir indicadores para falhas em software a partir da análise de dados históricos de falhas no software; investiga se as métricas de software orientado por objetos podem ser utilizadas como instrumento de predição de falhas em software; analisa se indicadores obtidos em um software podem ser utilizados para outros softwares; é um dos mais amplos estudos realizados com softwares comerciais, em relação a tamanho do código fonte, tamanho da equipe envolvida na produção de software e quantidade de usuários dos softwares. Ressaltamos que, dentre as métricas coletadas, não estão a de *conectividade*, tampouco métricas de coesão interna de módulos.

Gyimothy et al. [2005b] realizaram um estudo empírico sobre a predição de falhas em sistemas OO *open source*. O objetivo do trabalho é validar as métricas CK para predição de falhas em sistemas OO. Como estudo de caso, utilizaram um banco de dados de falhas detectadas no navegador Mozilla desde a sua versão 1.0 até a 1.7. Os valores das métricas CK foram comparados aos dados da base de falhas do Mozilla, o que levou à seguinte conclusão principal: a métrica CBO parece ser a melhor métrica para predição de falhas.

Avaliação

Prever a probabilidade de um software falhar a partir da análise da sua estrutura é um poderoso recurso para a obtenção de software mais confiável. O trabalho de Nagappan et al. [2006] é um dos mais significativos nesta linha, porque avalia softwares comerciais amplamente utilizados. Porém, ainda não se identificou uma métrica, ou um conjunto de métricas, que possa ser utilizada genericamente para a predição de falhas de software.

4.5 Manutenibilidade de Software

A norma ISO 9126 é um modelo para avaliação de qualidade de software que é tomado como padrão internacional. O modelo especifica seis características internas e externas de qualidade de software:

- Funcionalidade: característica do software que atende os seus requisitos.
- Confiabilidade: característica de um software que provê seus serviços em condições e período de tempo determinados.
- Usabilidade: facilidade de aprendizado, entendimento, utilização e atratividade de um software.
- Eficiência: refere-se ao bom uso que o software faz de recursos de sistema, tais como rede e memória.
- Portabilidade: facilidade com que o software é capaz de se adaptar à mudanças em seu ambiente.
- Manutenibilidade: facilidade de se realizar modificações em um software. Modificações compreendem alterações, correções, adaptações e inclusões de funcionalidades.

Para cada uma dessas características de qualidade, são definidas subcaracterísticas. Para manutenibilidade, a ISO 9126 define as seguintes subcaracterísticas:

- Analisabilidade: facilidade de se identificar as partes do software que deverão ser modificadas em decorrência de uma manutenção.
- Modificabilidade: facilidade de se realizar modificações no software.
- Estabilidade: capacidade de o software evitar efeitos não esperados em decorrência de uma modificação.
- Testabilidade: facilidade de se testar um software modificado.

A ISO 9126 indica métricas internas e externas para avaliar cada uma dessas subcaracterísticas. As métricas externas avaliam aspectos relacionados ao comportamento da equipe de manutenção, dos usuários e do sistema durante a atividade de manutenção. Um exemplo de métrica externa para *alterabilidade* é o *tempo necessário para implementação de alteração*, que indica a facilidade de se alterar o software para

solucionar um problema identificado pelo usuário. O tempo de alteração é dado pelo tempo decorrido entre a identificação das causas do problema e remoção dessas causas. A métrica é dada pela razão entre o somatório de tempos de alteração e o número total de falhas registradas e removidas. As métricas internas avaliam aspectos relacionados especificamente ao produto. Um exemplo de métrica interna para *estabilidade* é a *localização do impacto da modificação*, que indica a amplitude de impacto de uma modificação do produto, sendo dada pela razão entre o número de itens afetados pela modificação e o total de itens.

Avaliação

A norma ISO 9126 é um padrão de grande importância e disseminação internacional para a qualidade de software. Define, entre outros aspectos, as características, subcaracterísticas e suas respectivas métricas internas e externas. No caso da manutenibilidade de software, as métricas de avaliação abrangem aspectos relativos a existência ou não de fatores considerados importantes para a atividade de manutenção, por exemplo registro de *log* de operações, funções de teste, ou análise dos dados de registros de falhas e suas soluções. Não é abordado de forma específica nessa norma aspectos relacionados à avaliação da estrutura do software, tais como modularidade, coesão e acoplamento.

4.5.1 Modelos de Avaliação de Manutenibilidade

Esta seção discute trabalhos relacionados ao problema de predição de manutenibilidade de software. Os trabalhos estão agrupados segundo a abordagem utilizada por eles: utilização de métrica ou avaliação de características de manutenibilidade.

4.5.2 Modelos Baseados em Métrica

Índice de Manutenibilidade - MI

Uma das métricas mais conhecidas para avaliação de manutenbilidade de software é MI (*Maintainability Index*) [SEI, 2009]. A métrica é reconhecida e recomendada pelo *SEI - Software Engineering Institute*, que a considera como um recurso importante no ciclo de vida de software, uma vez que permite ao desenvolvedor avaliar a dificuldade de manutenção e, então, intervir na estrutura do software a fim de reduzi-la. A proposta inicial desta métrica foi realizada por Paul Oman e, posteriormente foi aperfeiçoada como resultado de um trabalho conjunto do SEI, da Hewlett-Packard e de outras organizações. A métrica MI é dada pelo seguinte polinômio:

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

onde $aveV$ é a média do Volume de Halstead por módulo, $aveV(g1)$ é a média da complexidade ciclomática por módulo, $aveLOC$ é a média do número de linhas de código por módulo e $perCM$ é a média do percentual de linhas de comentários por módulo. O termo $perCM$ é opcional na fórmula.

O *volume de Halstead* avalia a complexidade de módulos de programas por meio da análise de operandos e operadores. *Volume de Halstead* é dado por $V = N * log_2 N$, onde N é o tamanho do programa, o que corresponde ao total de operandos e de operadores.

A *complexidade ciclomática* é uma métrica para avaliar complexidade de código, que contabiliza o número de caminhos de execução em um módulo de um programa, cujo fluxo de execução é representado por um grafo. A métrica é dada por $CC = E - N + p$, onde E é o número de arestas no grafo, N é o número de nodos no grafo e p é o número de componente conectados no grafo.

No polinômio da métrica MI, os coeficientes foram calibrados de forma empírica a partir da análise de sistemas na Hewlett-Packard. A metodologia empregada para a

definição dos parâmetros da métrica foi o seu teste em softwares totalizando cerca de 50 KLOC e a verificação dos resultados a partir da comparação com dados subjetivos obtidos por meio de questionários. Os resultados desse primeiro estudo foram validados por um segundo, com um software de metade do tamanho do primeiro. Em um estudo posterior com softwares da US Air Force, escrito inicialmente em FORTRAN e traduzido para C, verificou-se que o uso de MI favoreceu a redução do esforço de manutenção.

Inicialmente, o autor da métrica desenvolveu um protótipo de uma ferramenta para coleta de medidas de MI em softwares escritos em Pascal e C. Como os resultados de seus estudos foram considerados sólidos, essa métrica passou a ser utilizada na construção de sistemas do Departamento de Defesa Americano. SEI recomenda as seguintes situações para o uso dessa métrica: verificação periódica da manutenibilidade de software; uso da métrica durante o processo de desenvolvimento do software; identificação de código que representa risco para a manutenção; comparação entre softwares.

O uso de métrica de software deve ser um instrumento de tomada de decisão. Diante de um valor não apropriado de uma métrica, deve ser possível intervir no software a fim de melhorar o aspecto avaliado. O uso da métrica MI não favorece isso, pois MI é computada a partir de valores de LOC, complexidade ciclomática e da métrica Halstead. É difícil para o gerente ou para o desenvolvedor intervir nestes aspectos e melhorar a manutenibilidade. Outro ponto importante é que esta métrica foi proposta e validada para software estruturados.

Um Estudo sobre Correlação entre um Conjunto de Métricas e Manutenibilidade

Li & Henry [1993] utilizaram métricas de software OO para avaliar manutenibilidade. O estudo foi realizado com software escrito em Ada. Os dados de esforço de manutenção e métricas OO foram coletados em dois softwares comerciais durante três anos. A métrica considerada como indicador de manutenbilidade foi o número de linhas de código alteradas por classe. O objetivo do estudo foi identificar relação entre esforço de manutenção e métricas de software. Dentre as métricas coletadas estão: DIT (profundidade na árvore de herança), NOC (número de filhos), LCOM, RFC (conjunto resposta de uma classe) e WMC (métodos ponderados por classe). O estudo teve como conclusões que a avaliação de manutenibilidade, tal como foi considerado, é possível a partir das métricas utilizadas. Contudo, uma observação a ser realizada sobre esse estudo é a métrica número de linhas de código alteradas pode não corresponder fielmente

à dificuldade de manutenção do software.

Um Estudo sobre Correlação entre Coesão e Alterabilidade

Kabaili et al. [2001] investigaram a correlação da coesão com a alterabilidade (do inglês *changeability*) de software orientado por objetos, partindo da afirmativa de Yourdon e Constantine que a coesão pode ser usada na predição de reusabilidade e manutenibilidade [4] (1979 apud Kabaili et al. [2001]). Alterabilidade é definida pelos autores como a capacidade de um software absorver uma alteração, e é determinada pelo impacto de uma alteração, ou seja, o conjunto de classes que necessitam de alteração ou correção em decorrência de uma dada alteração no sistema. Para validar a hipótese, foram utilizadas duas métricas de coesão em três sistemas industriais desenvolvidos em C++. Dos 66 tipos de alterações possíveis identificados pelos autores, foram analisados 6: alteração em tipo de variável, alteração de modificador de acesso de atributo de público para protegido, alteração da assinatura de método, modificador de acesso de classe de público para protegido, adição de classe abstrata na hierarquia de classes. Inicialmente foram coletadas as métricas de coesão nos sistemas. Para cada um dos tipos de alterações, e cada um dos sistemas, determinou-se o conjunto de classes às quais a alteração era aplicável. Para cada uma destas classes, calculou-se o número de classe que sofreram impacto. Com as métricas e os dados de impacto de alteração coletados, verificou-se a correlação entre coesão e impacto de alteração. Os resultados do experimento não comprovaram a hipótese do trabalho, ou seja, não foi identificada correlação entre coesão e alterabilidade. Os autores consideram que um dos seguinte fatores contribuiram para isso: ou as métricas escolhidas não avaliam coesão corretamente, ou de fato não existe correlação entre coesão e alterabilidade.

Um Modelo de Predição de Estabilidade de Classes

Grosser et al. [2003] apresentaram um modelo de predição de estabilidade de classes em software desenvolvido em Java. O modelo baseia-se em comparar versões de um software para aferir sua estabilidade. Métricas de software são utilizadas no modelo para identificar componentes de software similares. Os autores acreditam que dois componentes de software similares evoluirão da mesma maneira, ou seja, têm estabilidade

[4] Edward Yourdon and Lany L. Constantine. *Structured Design*. Prentice Hall, Englewood Cliffs, N.J., 1979.

similares. O conceito de estabilidade empregado no modelo é definido como o aumento de responsabilidades de uma classe ao longo da vida do software. Estabilidade é avaliada no nível de classe e entre versões do mesmo software. Para realizar esta avaliação, uma classe é representada por uma tupla $classe = (nome, m_1, m_2, ..., m_n, S_t, e)$, onde:

nome: é a identificação da classe;

m_i: é uma métrica de algum aspecto da classe. No trabalho, foram consideradas 14 métricas que avaliam os seguintes aspectos: coesão, conectividade, herança e complexidade.

S_t: é o chamado *fator de estresse* da classe, definido pelos autores como os fatores que têm impacto na estabilidade da classe. Foram identificados os seguintes fatores de estresse: modificações locais da classe em decorrência da definição de novos métodos; modificações nas classes ancestrais da classe; modificações nas classes descendentes da classe; modificações nas classes às quais a classe está conectada: aquelas que dependem da classe ou das quais a classe depende. O fator de estresse S_t é aproximado pelo percentual de métodos adicionados na classe. Formalmente: $S_t(c_i, c_{i+1}) = (I(c_{i+1}) - I(c_i))/(I(c_{i+1})$

e: é o indicador de estabilidade da classe. Seja $I(c_i)$ a interface da classe na versão i do software e $I(c_{i+1})$, a interface da classe na versão seguinte do software, sendo considerada interface como os métodos públicos e protegidos, locais ou herdados da classe. O nível de estabilidade da classe é medido pela razão entre $I(c_(i))$ e $I(c_{i+1})$, ou seja, é o percentual de $I(c_(i))$ em relação a $I(c_{i+1})$.

A hipótese do trabalho é que a estabilidade de uma classe depende de sua estrutura e do *estresse induzido* pela implementação de novos requisitos entre duas versões diferentes do software. A avaliação da estrutura da classe é realizada por meio de métricas de software. Para exemplificar o emprego do modelo proposto, foi realizado um estudo de caso com quatro versões do JDK.

O modelo de estabilidade proposto por Grosser et al. [2003] tem como base a idéia de que dois componentos de software similares evoluirão da mesma maneira. Isso é usado para predizer a estabilidade de classes em software orientado por objetos a partir da comparação com classes similares já avaliadas. Essa idéia parece ser sólida, porém o trabalho deixa em aberto ainda os seguintes pontos a serem explorados:

- O modelo avalia estabilidade de classes, mas não o de um sistema como um todo ou de um conjunto de classes. Ele poderia ser estendido para permitir esse nível de avaliação.

- Métricas são utilizadas como parâmetro de comparação entre classes. O trabalho não investiga correlação entre as métricas utilizadas.
- O trabalho propõe um modelo de estabilidade, mas não destaca a sua utilidade no processo de software; a estabilidade é avaliada, mas não há indicação sobre o que fazer com esta informação. Uma das possíveis utilizações desse resultado passível de investigação é a identificação de um conjunto de métricas correlacionadas à estabilidade e sua utilização para reestruturação de software.
- Dentre os resultados apontados pelos autores, está o de que provou-se que a estabilidade é predizível. Porém esta afirmativa não é sustentada pelo trabalho, uma vez que realizou-se apenas um estudo de caso. Para um comprovação do resultado, é necessário um estudo mais amplo.

Um Modelo de Avaliação de Manutenibilidade por meio da Conectividade do Software

Ferreira et al. [2008] abordam a conectividade como fator principal na determinação da manutenibilidade de software. O trabalho tem as seguintes contribuições principais:

1. Avaliação da relação entre conectividade e estabilidade de sistemas: uma análise sobre o impacto da conectividade na estabilidade de sistema é apresentada. Tal análise destaca a conectividade como fator principal na determinação da estabilidade de software, baseando-se no Modelo das Lâmpadas proposto por Myers [1975]. Neste modelo, Myers representa um software como um conjunto de lâmpadas conectadas entre si, no qual cada lâmpada representa um módulo. A mudança de estado de uma lâmpada de desligada para ligada representa atividade de manutenção no módulo correspondente e afeta o estado das demais lâmpadas conectadas a ela.
2. Análise crítica do Modelo de Estabilidade de Software proposto por Myers: Myers propõe uma métrica que indica a quantidade média de módulos que sofrerão impacto em decorrência de uma alteração em um módulo qualquer no software. No trabalho, é apresentado e analisado este modelo de Myers, e são propostas adaptações de tal métrica à orientação por objetos, visto que a métrica foi inicialmente descrita por Myers para o paradigma estruturado.
3. Adaptação dos conceitos de acoplamento e coesão à luz da orientação por objetos: visto que coesão e acoplamento são determinantes na conectividade de software,

realizou-se uma releitura desses conceitos para a orientação por objetos, classificando e exemplificando tipos de coesão e acoplamento em software orientados por objetos.

4. Identificação das métricas de software orientado por objetos impactantes no aspecto conectividade de sistema: é apresentada uma revisão bibliográfica detalhada para as principais métricas de software orientado por objetos propostas na literatura, identificando-se aquelas a serem utilizadas na avaliação do aspecto conectividade.

5. Proposta de duas métricas auxiliares na avaliação da conectividade: *conexões aferentes*, que indica o número de conexões que chegam a uma classe, e *peso de conexão aferente*, que corresponde ao grau de acoplamento envolvido em determinada conexão que chega a uma classe.

6. Proposta do Modelo de Conectividade em Sistemas Orientados por Objetos (MACSOO): é um método que visa a diminuição da conectividade em sistemas orientados por objetos a partir da avaliação e restruturação de sistemas sob os aspectos que determinam a conectividade.

7. Experimentos de avaliação de conectividade que resultaram em indícios de que a conectividade pode ser tomada como indicador principal na avaliação da manutenibilidade de software.

O trabalho deixa os seguintes pontos a serem explorados:

1. A Métrica de Estabilidade de Myers, utilizada em MACSOO, avalia o impacto de uma alteração qualquer em determinado módulo do software. É necessário categorizar os tipos de alterações e seus respectivos graus de impactos. Por exemplo, o impacto de alterar o nome de um método provavelmente não será o mesmo da alteração da lista de parâmetros do método.

2. Os resultados de nossa pesquisa apresentam indícios para comprovação da tese de que a conectividade é o fator preponderante na determinação da facilidade de manutenção de softwares orientados por objetos. Entretanto, faz-se necessária a realização de experimentos em larga escala para a comprovação dessa tese.

3. MACSOO constitui-se de avaliação de fatores de impacto na conectividade e da própria conectividade por meio de métricas. Indica os pontos nos quais tais fatores devem ser avaliados, porém não indica os valores a serem considerados como satisfatórios para os indicadores obtidos.

4.5.3 Modelos Baseados em Características de Manutenibilidade

Mapeamento de Propriedades de Código em Características de Manutenibilidade

Heitlager et al. [2007] definem um modelo de medição de manutenibilidade que baseia-se no mapeamento de propriedades do código, tais como volume e complexidade, em características do sistema relacionadas à manutenibilidade, como estabilidade e testabilidade. O modelo tem por objetivo favorecer um meio mais prático do que aquele obtido pela métrica MI. O modelo propõe que *analisabilidade* pode ser avaliada a partir das seguintes propriedades do código: volume, duplicação de código, tamanho da unidade e teste de unidade; *alterabilidade* pode ser avaliada a partir de complexidade por unidade e duplicação de código; *estabilidade* pode ser avaliada a partir de teste de unidade; e *testabilidade* pode ser avaliada a partir de complexidade por unidade, tamanho da unidade e teste de unidade. O conceito de *unidade* é definido pelos autores como a menor parte de um software que pode ser executada e testada individualmente. As propriedades do código podem avaliadas por meio de métricas. *Volume* corresponde ao tamanho do software e pode ser medido por métricas como LOC e pontos de função. *Complexidade* não é um conceito bem definido na produção de software; a métrica indicada para avaliar complexidade neste caso é *a complexidade ciclomática. Tamanho da unidade* é considerado um fator importante porque considera-se que unidades maiores tendem a ser mais difíceis de manter. A existência de *testes de unidade* é considerada um fator importante para a manutenibilidade do código. Algumas considerações devem ser realizadas a respeito desse modelo proposto por Heitlager et al. [2007]. O modelo é mais simples do que a métrica MI, mas eles têm abordagens diferentes: MI é uma métrica representada por uma única fórmula, enquanto o modelo proposto subdivide características relacionadas a manutenibilidade e as avalia de forma separada, por meio de várias métricas. Considerar que o tamanho do software ou de suas unidades é um fator determinantes para a manutenibilidade parece ser uma análise superficial sobre esse aspecto, pois pode ser que um software pequeno seja de difícil manutenção se, por exemplo, sua construção não tiver priorizado aspectos como baixo acoplamento entre módulos e alta coesão interna dos módulos. Além disso, a existência de testes de unidade é considerada no modelo uma propriedade de impacto na analisabilidade, estabilidade e testabilidade do software. Porém, diante de uma modificação no software, os seus testes de unidade poderão também sofrer alterações, e a facilidade de se

realizar tais alterações depende da forma como o código a ser testado foi construído.

Um Modelo Bidimensional de Manutenibilidade Baseado em Atividades

Deissenboek et al. [2007] propõem um modelo bidimensional de manutenibilidade baseado em atividades. O modelo considera que para alcançar manutenibilidade de software é preciso relacionar as propriedades do software e as atividades realizadas na manutenção. No modelo, a manutenção possui as seguintes atividades: conceito, análise de impacto, codificação e modificação. A atividade de análise de impactos, por exemplo, está relacionada às seguintes propriedades do código: concorrência, recursão, clonagem de código, e utilização de um depurador. Embora o modelo indique quais características do software estão relacionadas às atividades de manutenção identificadas, ele não determina como avaliar tais características. Além disso, o modelo não apresenta uma definição das propriedades do software. Por exemplo, o modelo define que a propriedade *formato do código* tem impacto na atividade de codificação, mas não define o que é considerado como formato do código.

4.5.4 Avaliação do Estado da Arte

Há alguns anos, vários trabalhos têm sido realizados com o objetivo de avaliar ou estimar a dificuldade de manutenção de software, o que evidencia que embora essa questão seja de grande importância na produção de software, ainda não foi solucionada. Esta seção realizou um estudo dos principais trabalho nesta linha.

Um dos trabalhos mais conhecidos para avaliação de manutenibilidade é a métrica MI [SEI, 2009], adotada pela SEI. Essa métrica foi proposta e validada utilizando-se softwares construídos no paradigma estruturado com linguagens como FORTRAN e C. Porém, softwares orientados por objetos têm características diferentes dos softwares construídos no paradigma estruturado. O trabalho de Li & Henry [1993] identificou correlação entre um conjunto de métricas OO e a manutenibilidade de software, porém a métrica utilizada para estimar esse fator, número de linhas de código alteradas, pode não corresponder fielmente à dificuldade real de manutenção. O trabalho de Kabaili et al. [2001] busca identificar correlação alterabilidade e coesão, utilizando duas métricas para este fator. Os resultados do trabalho não evidenciam tal correlação, segundo os autores, por um dos dois possíveis motivos: ou realmente a coesão não gera impacto na manutenção ou as métricas utilizadas não avaliam coesão satisfatoriamente. Grosser et al. [2003] definem um modelo de predição de estabilidade de classes baseado no pressuposto que duas classes com estrutura semelhantes tendem a evoluir da mesma maneira. Embora o trabalho tenha concluído que é possível realizar esse tipo de predição com a abordagem proposta, é necessária a realização de estudos mais amplos para confirmar essa conclusão. Além disso, o modelo avalia estabilidade de classes e não de um software como um todo. Ferreira et al. [2008] propõem a avaliação de manutenibilidade por meio de sua conectividade, e definem um método de reestruturação de software baseado nisso. Os resultados do estudo evidenciaram a eficiência do método, porém ainda é necessária a realização de experimentos mais significativos para a comprovação dessa conclusão. Alguns trabalhos para avaliação de manutenibilidade são baseados nas características desse aspecto. citeDEI2007 definem um modelo que associa as propriedades do código e as atividades realizadas na fase de manutenção. O modelo indica quais características devem ser avaliadas, mas não determina como avalia-las. Heitlager et al. [2007] definem um modelo que mapeia características de código, representadas por métricas, em características de manutenibilidade. A ideia do modelo é consistente porque considera os vários aspectos de manutenibilidade. Contudo, a sua aplicação pode não ser muito prática, pois utiliza uma série de métricas isoladas para avaliar a dificuldade de manutenção. Além disso, as métricas utilizadas podem não ser indicadores fieis. Por exemplo, o modelo utiliza o tamanho do soft-

ware como um dos indicadores, e não necessariamente essa métrica indica facilidade de manutenção. Embora esses dois últimos trabalhos sejam baseados em características de manutenbilidade, a utilização deles é dependente do uso de métricas.

O problema de avaliação de manutenibilidade de software está em aberto e parece ser de difícil solução. Uma das causas para esse cenário é que para realiza a predição de dificuldade de manutenção é preciso avaliar o software e suas características. Essa avaliação é idealmente realizada por meio de métricas. Para alguns fatores, como coesão, não há um consenso sobre uma métrica para sua avaliação. Ocorre que há uma grande quantidade de métricas propostas na literatura, mas ainda não se conhece os valores a serem considerados como satisfatórios para elas. A solução dessa questão tem papel central para o uso efetivo de métricas na produção de software. O segundo ponto é que, embora haja um número significativo de modelos de predição de dificuldade de manutenção de software, não há um modelo que realize isso satisfatoriamente no caso de softwares orientados por objetos. O modelo mais conhecido com o propósito de avaliar manutenibilidade corresponde à métrica MI. Essa métrica foi proposta e validada para o paradigma estruturado, mas não captura características do paradigma OO.

4.6 Conclusão

Este capítulo realiza uma revisão dos principais trabalhos na área de medição de software orientado por objetos. As principais linhas de pesquisas atuais em métricas de software orientado por objetos foram identificadas. Dentre elas, destacam-se: a pesquisa por uma métrica de coesão padrão, questão ainda em discussão na literatura; medição de software orientado por aspectos; o uso de métricas na reestruturação de software; o uso de métricas para predição de falhas e manutenibilidade de software. Este último assunto foi enfatizado por se tratar do tema discutido nesta tese.

A busca por uma métrica de coesão padrão justifica-se pela importância da avaliação deste fator para a qualidade estrutural de um software. A maior dificuldade para alcançar uma métrica que avalie fielmente a coesão interna de classe é decorrente da natureza do fator avaliado, pois como a coesão é o grau de relacionamento entre os elementos internos do módulo, a sua avaliação é fortemente dependente do entendimento do domínio do problema do software. Não há consenso na literatura sobre a métrica de coesão interna de módulos na OO. As principais abordagens utilizadas na definição de métricas para este fator são:

- a coesão interna de uma classe é definida em função do grau de similaridade entre

seus métodos, o que é analisado a partir do uso de variáveis de instâncias comuns entre os métodos. Esta abordagem é empregada na métrica LCOM;

- uma classe é considerada coesa se seus métodos usam o mesmo conjunto de tipos de parâmetros. Esta abordagem é utilizada no cálculo das métricas CAMC e NHD;
- a coesão é avaliada a partir da análise de informações semânticas no código fonte, tais como comentários. O cálculo da métrica utiliza técnicas de recuperação de informação. Esta abordagem é empregada na métrica C3.
- a coesão de uma classe é avaliada pela maneira como as suas classes clientes utilizam seus serviços. Esta abordagem é empregada na métrica ELCOM.

Nesta área de pesquisa é necessário definir melhor o que é coesão interna de uma classe. Dado o fato que a avaliação de coesão interna de módulos depende do entendimento do problema tratado pelo software, as métricas propostas para este aspecto devem ser validadas em relação à avaliação qualitativa de especialistas. Outra questão de grande relevância ainda não solucionada é o valor a ser considerado satisfatório para as métricas. Para estas duas questões em aberto, faz-se necessário a realização de experimentos extensivos.

Identificar pontos de necessidades de refatoração em sistemas de grande porte pode ser uma tarefa inviável. Com o objetivo de solucionar este problema, a utilização de métricas de software para a reestruturação de software tem sido investigada. Os trabalhos nesta linha baseiam-se em definir um conjunto de métricas que permite identificar automaticamente um *bad smell* em um código de software. Embora esses trabalhos tenham encontrado resultados significativos, as seguintes questões ainda precisam ser exploradas: os estudos precisam ainda ser validados em sistemas de grande porte; vale investigar se há um fator, ou um conjunto de fatores, que esteja relacionado a todos os problemas de desenho de classes.

Do ponto de vista de redução de custo de software, duas áreas de pesquisa correlatas em métricas de software estão em curso: predição de falhas e predição de esforço de manutenção. Saber antecipadamente a probabilidade de falhas de um sistema é essencial para avaliar a sua confiabilidade. Os trabalhos realizados nesta linha tentam identificar os fatores e respectivas métricas relacionados a falhas em sistemas. Para isso, métricas são eleitas, suas medidas são coletadas em softwares e o resultado desta coleta é comparado ao histórico de falhas dos softwares avaliados. Nesta área ainda não foi identificado um fator, ou um conjunto de fatores, que possa ser utilizado na predição de falhas de qualquer software. A solução deste problema é de grande importância, pois

isso significaria uma grande simplicidade e eficiência na predição de falhas de sistemas. Para alcançar este objetivo é necessária a realização de experimentos com um grande número de softwares e a lista de fatores avaliados deve ser revisada, pois, por exemplo, em um dos maiores estudos realizados nesta linha não foram utilizadas as métricas de *conectividade* e coesão interna de classes.

O alto custo da fase de manutenção de software é fato que motiva vários estudos que têm como objetivo encontrar soluções que reduzam custos e esforços nesta fase bem como meios de predição deste esforço. Métricas de software orientado por objetos têm sido investigadas como instrumento de avaliação de manutenibilidade de software. A principal proposta de solução desse problema é a métrica MI. Essa métrica foi proposta e validada para softwares implementados no paradigma estruturado. Os estudos realizados nesta área ainda não identificaram de forma conclusiva um fator, ou um conjunto de fatores, que possa ser utilizado na predição de dificuldade de manutenção em software orientado por objetos. A solução desta questão corresponde a um instrumento poderoso na área de produção de software, pois permitiria ao desenvolvedor realizar uma análise prévia do grau de esforço de manutenção de um software e, mediante esta análise, atuar na estrutura do software de maneira a reduzir seu custo de manutenção. Dentro desta área de investigação, a comprovação de que a conectividade entre os módulos de um sistemas é o fator principal na avaliação do esforço de manutenção está em aberto. As grandes dificuldades nessa linha de pesquisa são: não há um consenso sobre a melhor forma de se avaliar a manutenibilidade, por exemplo, há estudos que realizam esta avaliação a partir do número de linhas alteradas por classe e outros como o tempo necessário para realizar uma manutenção; a realização de um estudo conclusivo depende de experimentos em larga escala preferencialmente com sistemas reais.

Outra questão relevante ainda não solucionada é a identificação de valores a serem considerados satisfatórios para as métricas. Na literatura consta uma imensa quantidade de métricas de software, porém não há estudos que indiquem os valores considerados apropriados para elas. A indicação desses valores é essencial para o uso efetivo de métricas de software, pois sem esta informação não é possível a tomada de decisão apropriada a partir do uso das métricas.

Diante dos problemas identificados, esta tese tem por objetivos: a proposta de um modelo de avaliação de manutenibilidade de software orientado por objetos, a proposta de uma métrica de avaliação de coesão interna de classes e a realização de um estudo para identificar valores referência para as métricas utilizadas no modelo proposto.

Capítulo 5

Métrica de Coesão Contratual

Coesão interna de módulos é um dos conceitos mais conhecidos na área de software. Foi definida inicialmente por Myers [1975] como o grau de interrelacionamento entre os elementos internos de um módulo. O conceito foi introduzido no contexto do paradigma estruturado. Um módulo pode ser entendido como um conjunto de elementos delimitado. Em um software construído no paradigma estruturado, podem ser considerados como módulos: uma função, um procedimento ou um arquivo que pode ser compilado em separado. No paradigma orientado por objetos, podem ser considerados como módulos: uma classe, um método, um pacote. Alinhado com a definição de von Staa [2000] de que um módulo deve ser uma unidade de compilação de um programa, definimos módulo na orientação por objetos como uma classe. Classes possuem dois tipos de elementos: atributos e métodos. Desta forma, a coesão interna de uma classe é definida como o grau de interrelacionamento entre seus atributos e métodos.

A coesão, ao lado do acoplamento, são dois dos principais fatores a serem avaliados na qualidade de um software. Aceita-se de forma bastante disseminada que é necessário aumentar o grau de coesão interna de módulos e reduzir grau de acoplamento entre módulos. Visando este objetivo, várias formas de se avaliar coesão e outros aspectos estruturais de software, como acoplamento, têm sido propostas. A avaliação de um fator demanda o uso de uma métrica. A métrica mais popular de avaliação de coesão na orientação por objetos é LCOM, proposta por Chidamber & Kemerer [1994]. Alguns estudos [Marcus & Poshyvanyk, 2005; Counsell et al., 2004; Mäkelä & Leppänen, 2007] mostram que esta métrica não avalia fielmente coesão. Neste trabalho, propomos uma forma de avaliar coesão baseada em contratos da classe. O contrato de uma classe é constituído pelos serviços exportados por ela. Partindo-se da definição de que uma classe bem construída deve implementar um único contrato, a ideia da métrica é contar o número de contratos que uma classe implementa.

5.1 Descrição da Métrica Coesão Contratual

O grau de coesão interna de uma classe é dado pelo relacionamento entre seus atributos e métodos. Um conceito importante na avaliação de coesão interna de classes é o *contrato* [Meyer, 1997]. Um contrato define uma relação *cliente-fornecedor* entre duas classes regida por regras a serem seguidas tanto por quem usa quanto por quem fornece o serviço. A classe fornecedora compromete-se a fornecer um determinado serviço a partir de uma especificação do problema e publica a interface a ser utilizada por aquelas que precisarem utilizar o serviço. A classe cliente compromete-se a cumprir as pré-condições para utilizar o serviço fornecido. Para benefício da modularidade, é importante a criação de classes que implementam um único contrato. Por exemplo, uma classe que implementa os tipos abstratos de dados fila e pilha possui dois contratos. Uma classe que implementa uma fila tem apenas um contrato, tal como a classe que implementa uma pilha. Estas duas classes têm grau de coesão melhor do que aquela que implementa os dois contratos.

A ideia da métrica proposta baseia-se no número de contratos que uma classe implementa. A coesão contratual é dada por $1/t$, onde t é o total de contratos da classe. Por exemplo: se uma classe implementa dez contratos, o valor da métrica resulta em 0,1, se uma classe implementa dois contratos, resulta em 0,5 e se uma classe implementa um único contrato, resulta em 1. A métrica pode assumir valores de 0 a 1, e quanto mais próximo de 1 melhor a coesão contratual da classe. A seguir a métrica é definida formalmente.

5.2 Conceitos

1. *Contrato:* conjunto de métodos.
2. *Relacionamento:* dois métodos de uma classe c estão relacionados se utilizam pelo menos um atributo da classe c em comum, ou se um utiliza o outro, ou se usam direta ou indiretamente métodos em comum.
3. *Propriedade Transitiva de Relacionamento:* se a está relacionado com b e b está relacionado com c então a está relacionado com c.

5.2.1 Definição Formal da Métrica

- Número de métodos de uma classe: m.
- Número máximo de contratos de uma classe: $N = m$.

- C é o conjunto de conjuntos disjuntos formados por métodos com relacionamento entre si.
- Número de contratos coesos: $n = |C|$
- $CoesaoContratul = 1/n$, se $n > 0$
  Coesao Contratual = 0 , caso contrário

5.3 Implementação da métrica

A métrica proposta já foi implementada na ferramenta *Connecta*. Pretende-se utilizá-la no modelo de predição de esforço de manutenção proposto neste trabalho. Os seus valores foram coletados em um conjunto de aproximadamente 25.000 classes. Os resultados desse experimento serão ainda avaliados e publicados.

Capítulo 6

Valores Referência para Métricas de Software Orientado por Objetos

Embora a importância de métricas na produção de software seja notória, uma vez que, por meio delas, é possível medir, avaliar, controlar e melhorar os produtos e processos, este recurso não tem sido amplamente utilizado na indústria. Tempero [2008] avalia que um dos motivos para esse fato é que para a maior parte das métricas de software ainda não se conhece o modelo de entidade da população (*entity population model*), o qual refere-se aos valores típicos de medidas que uma métrica assume em determinado conjunto de entidades. Há uma carência de estudos empíricos sobre o assunto que gerem resultados que possam ser aplicados a software de forma geral [Tempero, 2008].

A caracterização de softwares OO é importante, porque pouco se conhece sobre as características de softwares reais produzidos neste paradigma. Não se conhece, por exemplo, informações simples como quantos métodos uma classe típica possui, ou quantas classes um pacote típico possui. Muitas métricas de software OO têm sido propostas [Abreu & Carapuça, 1994; Chidamber & Kemerer, 1994; Fenton & Neill, 2000; Martin, 1994; Xenos et al., 2000], porém pouco se sabe sobre as suas medidas em softwares reais. Além disso, ainda está em aberto a definição dos valores desejáveis para essas métricas, o que restringe o uso efetivo de métricas na produção de software.

Um problema recorrente na área de medição de software é a obtenção de dados para a realização dos experimentos, pois a coleta de medidas envolve muitas vezes a análise de código fonte ou compilado dos softwares. Uma solução para isso é analisar dados de software livre. Há hoje disponível uma grande quantidade de softwares desta natureza. Sourceforge (www.sourceforge.net), por exemplo, conta com mais de 176.000 softwares livres registrados.

Este trabalho visa realizar um estudo de caracterização de softwares OO. O propósito deste estudo é obter resultados que possam evidenciar as características deste tipo de software sob os aspectos de grau de conectividade do software, grau de coesão das classes, profundidade na árvore de herança das classes e número de conexões aferentes das classes, que corresponde ao total de classes que usam serviços de uma determinada classe, além do número de atributos e de métodos públicos. Busca-se identificar valores a serem considerados como referência para um conjunto de métricas de software OO que avaliam esses aspectos. Para isso, investigou-se se as métricas descritas na Seção 6.2.1 podem ser modeladas por alguma distribuição de probabilidades. É importante identificar o modelo probabilístico adequado a um fenômeno porque isso viabiliza o uso de probabilidades em fenômenos reais. Por exemplo, para determinar o número adequado de atendentes em uma central telefônica, é importante conhecer a distribuição de probabilidades que modela o número de clientes que ligam para a central em um determinado intervalo de tempo [Soares et al., 1991]. No caso de software, a identificação do modelo probabilístico das característica avaliadas neste trabalho auxilia, por exemplo, na determinação de emprego de recursos na manutenção do software. Além disso, a partir da identificação desse modelo, é possível identificar valores típicos da característica avaliada ou, pelo menos, concluir que não há um valor típico.

6.1 Trabalhos Relacionados

Uma crescente atenção tem sido dada por pesquisadores à análise da forma como os módulos de um software conectam-se entre si e tem-se chegado à conclusão que softwares parecem ser regidos pelas chamadas *power laws* [Baxter et al., 2006; Louridas et al., 2008; Potantin et al., 2005; Puppin & Silvestrini, 2006; R & Counsell, 2003]. Uma *power law* é uma função de distribuição de probabilidades na qual a probabilidade de uma variável randômica X assumir um valor x é proporcional a uma potência negativa de x. Essa relação é mostrada na Equação 6.1.

$$P(X = x) \propto cx^{-k} \tag{6.1}$$

Uma distribuição que segue uma *power law* é dita ser de cauda pesada (*heavy-tail distribution*), sendo caracterizada por existir uma quantidade não desprezível de ocorrências em que a variável randômica assume valores muito altos, mas na maior parte das ocorrências assume valores baixos. Outra característica desta distribuição é ter escala livre (*scale-free*), o que significa que a média não é um valor informativo, ou

seja, não há um valor que possa ser considerado típico para a característica avaliada. Uma grande variedade de fenômenos são modelados por *power laws*, por exemplo: frequência de uso de palavras, citações de artigos científicos, chamadas telefônicas, graus de entrada e saída dos nós na WWW e na Internet [Newman, 2003].

O relacionamento entre as classes ou entre os objetos de um sistema OO pode ser modelado como um grafo dirigido. Alguns pesquisadores têm utilizado esta modelagem e identificaram que esses grafos seguem *power laws* e, consequentemente, têm características de redes de livre escala, o que significa que não há um valor típico para determinada característica da rede. Potantin et al. [2005] identificaram que a geometria do grafo que representa o relacionamento entre os objetos de um software OO, em tempo de execução, tem escala livre. Um grafo com geometria em escala livre se diferencia de um grafo no qual as arestas são randomicamente distribuídas. Em um grafo randômico, os nodos têm aproximadamente o mesmo grau, ou seja, a média dos graus dos nodos é um valor significativo. A Web é um exemplo de um grafo em escala livre porque não há um valor médio para a conectividade de seus nodos. Foram analisados 60 grafos de objetos de 35 programas. De acordo com citePOT2005, as conclusões do trabalho são úteis para auxiliar a melhora de desempenho de coletores de lixo, a depuração de programas e o desempenho de programas.

R & Counsell [2003] identificaram *power laws* nos relacionamentos entre as classes de software OO desenvolvido em Java. A análise realizada por eles teve como amostra três softwares bem conhecidos: JDK (*Java Development Kit*), Apache Ant e Tomcat, totalizando 6.870 classes. O objetivo do trabalho é verificar *power laws* nos diferentes tipos de conexões que podem ocorrer em um software OO: herança, implementação de interface, agregação (classes que têm outras classes como tipos de atributos e classes que são tipos de atributos de outras classes), uma classe fazer parte da lista de parâmetros de métodos de outra classe, uma classe ser tipo de retorno de métodos de outra classe. Além disso, verificou-se que as seguintes características de classes também seguem *power laws*: número de campos, métodos e construtores em cada classe.

O trabalho de Louridas et al. [2008] analisa as distribuições estatísticas que os graus de entrada e de saída dos módulos de um sistema seguem. A amostra de dados analisada corresponde a softwares desenvolvidos em C, Perl, Java e Ruby. Foram analisados 11 softwares, dentre eles: J2SE SDK, Eclipse, OpenOffice e Ruby 1.8.2. O trabalho concluiu que, independente do paradigma de programação, os graus de entrada e saída são regidos por *power law*, ou seja, há uma grande número de módulos que possuem baixo grau de entrada e saída, e um número pequeno, mas não insignificante, que possui valores muito altos desses graus.

Os achados desses trabalhos têm grande importância, porém pode ser precipitado

afirmar categoricamente que os seus resultados são conclusivos, pois as amostras investigadas podem não representar significativamente o universo de softwares existentes. Por exemplo, a maior parte dos trabalhos baseia-se em softwares abertos, conhecidos e diretamente relacionados à area de desenvolvimento de software e pesquisa. Pode ser uma característica dos desenvolvedores destes softwares a grande preocupação com qualidade estrutural de software, o que pode levar a softwares com estas características. Além disso, a maior parte dos trabalhos têm investigado características no nível de sistema: os relacionamentos entre classes e objetos. Porém outras características interessantes em um nível de granularidade menor, como da classe e de métodos, devem ainda ser investigadas, por exemplo o grau de coesão interna dos módulos, grau de ocultação de informação e tamanho de métodos.

Dezenas de métricas têm sido propostas para avaliar diversos aspectos de software [Xenos et al., 2000], em particular para a OO, destacam-se as métricas do conjunto MOOD [Abreu & Carapuça, 1994] e do conjunto CK [Chidamber & Kemerer, 1994]. Tempero [2008] avalia a necessidade de se conhecer os valores típicos destas métricas para que elas tenham uso efetivo na produção de software e destaca que muito pouco se sabe sobre isso. Baxter et al. [2006] realizaram um estudo com o objetivo de investigar a estrutura de software desenvolvido em Java. Como os próprios autores salientam, este foi o primeiro trabalho a realizar um estudo desta natureza com um grande número de softwares. No estudo de Tempero [2008] foram utilizados 56 softwares abertos de tamanho pequeno a médio, e de diferentes domínios de aplicação. Os critérios para escolha dos softwares foram o fato de já terem sido utilizados em outros estudos ou serem amplamente utilizados, tais como tais como ArgoUML, Eclipse e NetBeans. Foram coletadas medidas das seguintes métricas para classes: número de métodos, campos, construtores, subclasses, interfaces implementadas, referências a uma classe como um membro, número de tipos dos campos, número de tipos usados como parâmetros nos métodos, referências à classe como parâmetro, referências à classe como retorno, número de tipos usados como retorno nos métodos, número de classes das quais se depende para compilar, número de classes que dependem da classe para compilar, número de métodos públicos; para interfaces: número de classes que as implementam; para método: número de instruções no *bytecode*, e para pacotes: número de classes no pacote. As medidas foram coletadas e analisadas para cada um dos softwares estudados. Os resultados desta análise mostram que algumas métricas têm distribuição que segue uma *power law* enquanto outras não. As métricas que têm distribuição *power law* incluem: referências a uma classe como um membro, referências à classe como parâmetro, referências à classe como retorno, número de subclasses. Dentre as métricas que não possuem distribuição *power law* estão: número de tipos

dos campos, número de tipos usados como parâmetros nos métodos, número de tipos usados como retorno nos métodos. As métricas número de campos, número de métodos e número de métodos públicos também não seguem *power law*. As aplicações são de domínios diferentes e talvez domínios diferentes tenham distribuições diferentes; outra explicação é que os projetos das aplicações podem ser muito diferentes entre si.

O trabalho apresentado neste capítulo avança o estudo das características de software aberto, avaliando métricas importantes como LCOM, COF e DIT, ainda não avaliadas desta forma em estudos precedentes. Detalhes dessas métricas são descritos na Seção 3. Os softwares analisados neste estudo são de vários domínios de aplicação, o que representa melhor a diversidade de softwares existente. O investimento em qualidade estrutural é fator crítico de sucesso de softwares abertos, pois esse tipo de software é caracterizado por escassez de documentação e por grande quantidade de atividade de manutenção. Samoladas et al. [0054] identificaram que a manutenibilidade de softwares abertos tende a ser melhor do que a de softwares fechados. Com base nisso, os resultados do estudo realizado neste trabalho possibilita identificar valores referência para as métricas estudadas. Esta é uma questão em aberto e sua solução constitui uma contribuição importante para viabilizar o emprego efetivo de métricas na produção de software.

6.2 Metodologia

Os dados utilizados neste estudo são de 40 softwares abertos desenvolvidos em Java. Os softwares foram obtidos de *www.sourceforge.net*. Foram utilizados softwares de 11 domínios de aplicações diferentes, classificados pelo próprio *site*. Os dados relativos aos softwares e seus respectivos domínios são mostrados na Tabela 6.1. No total foram analisadas 26.202 classes.

Foi utilizada a ferramenta *Connecta* [Ferreira, 2006] que coleta as medidas a partir do *byte code* das classes. Por esta razão, um dos critérios para a escolha dos softwares analisados foi a disponibilidade dos *bytecodes* das classes ou a facilidade de sua geração. Dentre os softwares analisados estão alguns de grande disseminação tais como Hibernate, *framework* Spring e a biblioteca JUnit, porém a escolha dos softwares não se baseou na sua popularidade, porque este critério poderia interferir nos resultados.

6.2.1 Métricas

Neste trabalho foram avaliadas as seguintes métricas:

COF (*Coupling Factor* - Fator acoplamento): esta métrica é um indicador do grau de conectividade do software.

LCOM (*Lack of Cohesion in Methods* - Ausência de coesão em métodos): é uma métrica do conjunto CK para a ausência de coesão entre os métodos de uma classe.

DIT (*Depth of Inheritance Tree* - Profundidade da árvore de herança): métrica do conjunto CK que indica a posição de uma classe na árvore de herança de um software.

Conexões Aferentes [Ferreira et al., 2008]: esta métrica é dada pelo total de classes que utilizam serviços de uma determinada classe. Do ponto de vista de manutenção, é importante conhecer as classes de maior impacto no sistema, ou seja, aquelas que têm grande número de conexões aferentes.

Foram analisados também as quantidades de atributos e métodos públicos das classes.

6.2.2 Ajuste dos Dados

Para o ajuste dos dados foi utilizada a ferramenta EasyFit versão 5.0. Esta ferramenta realiza automaticamente o ajuste dos dados para diversas distribuições de probabi- lidades conhecidas, indicandos os melhores ajustes, além de realizar análises estatísticas básicas, tais como cálculo de média e desvio padrão. Dentre as distribuições discretas avaliadas pela ferramenta estão Bernoulli, Binomial, Uniforme, Geométrica, Hipergeométrica, Logarítmica, Binomial Negativa e Poisson; dentre as distribuições contínuas estão Beta, Uniforme, Normal, t-Student, Chi-quadrado, Erland, Exponencial, Frechet, Gama, Lognormal, Pareto e Weibull. Cada distribuição tem uma *pdf* (*probability density function*) e uma *cdf* (*cumulative distribution function*) característica. A *pdf* corresponde a uma equação que indica a probabilidade da variável aleatória assumir determinado valor x. A *cdf* corresponde a uma equação que indica a probabilidade da variável aleatória assumir um valor menor ou igual a x. As distribuições identificadas nos experimentos foram: Geométrica, Poisson e Weibull.

A distribuição Geométrica tem sua *pdf* , $f_g(x)$, e *cdf*, $F_g(x)$, com parâmetro p, dadas pelas Equações 6.2 e 6.3 respectivamente.

$$f_g(x) = P(X = x) = p(1-p)^{x-1}, 0 < p < 1 \tag{6.2}$$

$$F_g(x) = P(X \leq x) = 1 - (1-p)^x, 0 < p < 1 \tag{6.3}$$

Tabela 6.1. Softwares utilizados no estudo e seus respectivos domínios

Domínio	*Software*	*#Classes*	*#Conexões*	*COF*
Clustering	Essence	182	543	0,016
	Gridsim	214	774	0,017
	JavaGroups	1061	3807	0,003
	Prevayler	90	137	0,017
	Super	246	1085	0,018
Database	DBUnit	289	911	0,011
	ERMaster	569	2187	0,007
	Hibernate	1359	5199	0,003
Desktop	Facilitator	2234	6565	0,001
	JavaGuiBuilder	60	126	0,036
	JavaX11Library	318	1146	0,011
	Jpilot	142	367	0,018
	Scope	214	535	0,012
Development	CodeGenerationLibrary	226	662	0,013
	DrJava	2766	9684	0,001
	FinfBugs	1019	3108	0,003
	JasperReports	1233	5610	0,004
	Junit	154	353	0,015
	SpringFramework	2116	7069	0,002
	BCEL	373	2111	0,015
Enterprise	Liferay	14	14	0,077
	Talend	2779	3567	0,000822
	Uengine	708	1774	0,004
	YAWL	382	1186	0,008
Financial	Jmoney	193	424	0,019
Games	JSpaceConquest	150	424	0,019
	Kolmafia	810	5106	0,008
	Robocode	29	16	0,02
Hardware	Jcapi	21	61	0,145
	LibUSBJava	35	90	0,076
	ServoMaster	55	117	0,039
Multimedia	CDK	3586	14711	0,001
	Jpedal	539	1533	0,005
	Pamguard	1503	5267	0,002
Networking	Bluecove	142	461	0,023
	DHCP	18	29	0,095
	JSLP	42	156	0,091
	WiKIDStrong Authentication	50	27	0,011
Security	JSCH	110	226	0,022
	OO Distributed Virtual Systems	171	325	0,011

A distribuição de Poisson é conhecida como a distribuição dos eventos raros, o que significa que a ocorrência de valores altos para a variável aleatória tem baixa probabilidade [Soares et al., 1991]. A distribuição de Poisson tem *pdf*, $f_p(x)$, e *cdf*, $F_p(x)$, definidas pelas Equações 6.4 e 6.5 respectivamente. O parâmetro λ da distribuição indica a média dos valores da variável aleatória.

$$f_p(x) = P(X = x) = \frac{e^{-\lambda}.\lambda^x}{x!} \tag{6.4}$$

$$F_p(x) = P(X \leq x_0) = \sum_{x=0}^{x=x_0} \frac{e^{-\lambda}.\lambda^x}{x!} \tag{6.5}$$

A distribuição Weibull tem sua *pdf*, $f_w(x)$, e sua *cdf*, $F_w(x)$, com parâmetros α e β, definidas pelas Equações 6.6 e 6.7 respectivamente. Essa distribuição aplica-se a situações em que a variável de interesse apresenta assimetria à esquerda, ou seja, quando há um número pequeno de ocorrências com valores altos para a variável de interesse e um número muito grande de ocorrências com valores baixos. Weibull é uma distribuição de cauda pesada. Conforme apresentado na Seção 6.1, neste tipo de distribuição a média não é significativa.

$$f_w(x) = P(X = x) = \frac{\alpha}{\beta}(\frac{x}{\beta})^{\alpha-1}e^{-(\frac{x}{\beta})^{\alpha}}, \alpha > 0 e \beta > 0 \tag{6.6}$$

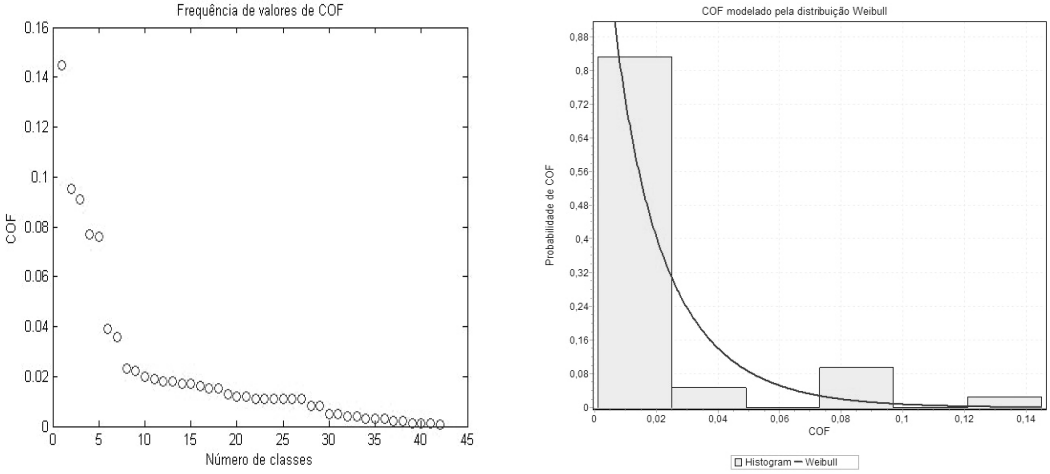
$$F_w(x) = P(X \leq x) = 1 - e^{-(\frac{x}{\beta})^{\alpha}}, \alpha > 0 e \beta > 0 \tag{6.7}$$

6.3 Resultados

Para cada uma das métricas analisadas neste trabalho, foram gerados gráficos de dispersão e realizados os ajustes de dados às distribuições de probabilidades disponíveis na ferramenta utilizada. Os resultados da análise dos dados são discutidos nesta seção.

6.3.1 Métrica COF

O gráfico de dispersão de COF, mostrado na Figura 6.1, indica que valores menores do que 0,20 são muito mais frequentes do que valores maiores do que isso. Os valores de COF podem ser modelados pela distribuição Weibull, com parâmetros $\alpha = 0,91927$ e $\beta = 0,01762$. O gráfico da Figura 6.1 mostra o ajuste da distribuição aos dados coletados. Isso indica que a probabilidade de uma software possuir um valor alto para COF é baixa. Pela análise dos gráficos, mais de 80% dos softwares têm COF



(a) (b)

Figura 6.1. Dispersão de COF e seu ajuste à distribuição Weibull.

em torno de 0,02, a pro- babilidade de COF estar entre 0,02 e 0,14 é relativamente baixa, e a probabilidade de ser maior do que 0,14 tende a zero.

6.3.2 Métricas de Classe

Conexões Aferentes

A análise do gráfico de dispersão dos valores da métrica Conexões Aferentes, mostrado na Figura 6.2, sugere que se trata de uma distribuição de cauda pesada, com um número pequeno de classes com uma quantidade muito alta conexões aferentes e há uma quantidade muito grande de classes com poucas conexões aferentes. Isso indica que a maior parte das classes têm poucas classes dependentes. De fato, como mostra a Figura 6.2, os valores desta métrica podem ser modelados por duas distribuições: Geométrica e Weibull. A distribuição discreta Geométrica com parâmetro $p = 0,24248$ modela os dados desta métrica. Os dados podem também ser modelados pela distribuição contínua Weibull, com parâmetros $\alpha = 0,78986$ e $\beta = 3,2228$. A análise gráfica mostra que cerca de 50% da classes tem 1 conexão aferente. A probabilidade da medida desta métrica estar entre 1 e 20 é relativamente baixa, e de ser maior do que 20 tende a zero.

LCOM

A distribuição dos valores da métrica LCOM também é de cauda pesada. Os gráficos da Figura 6.3 mostram as dispersões do conjunto completo e de valores menores

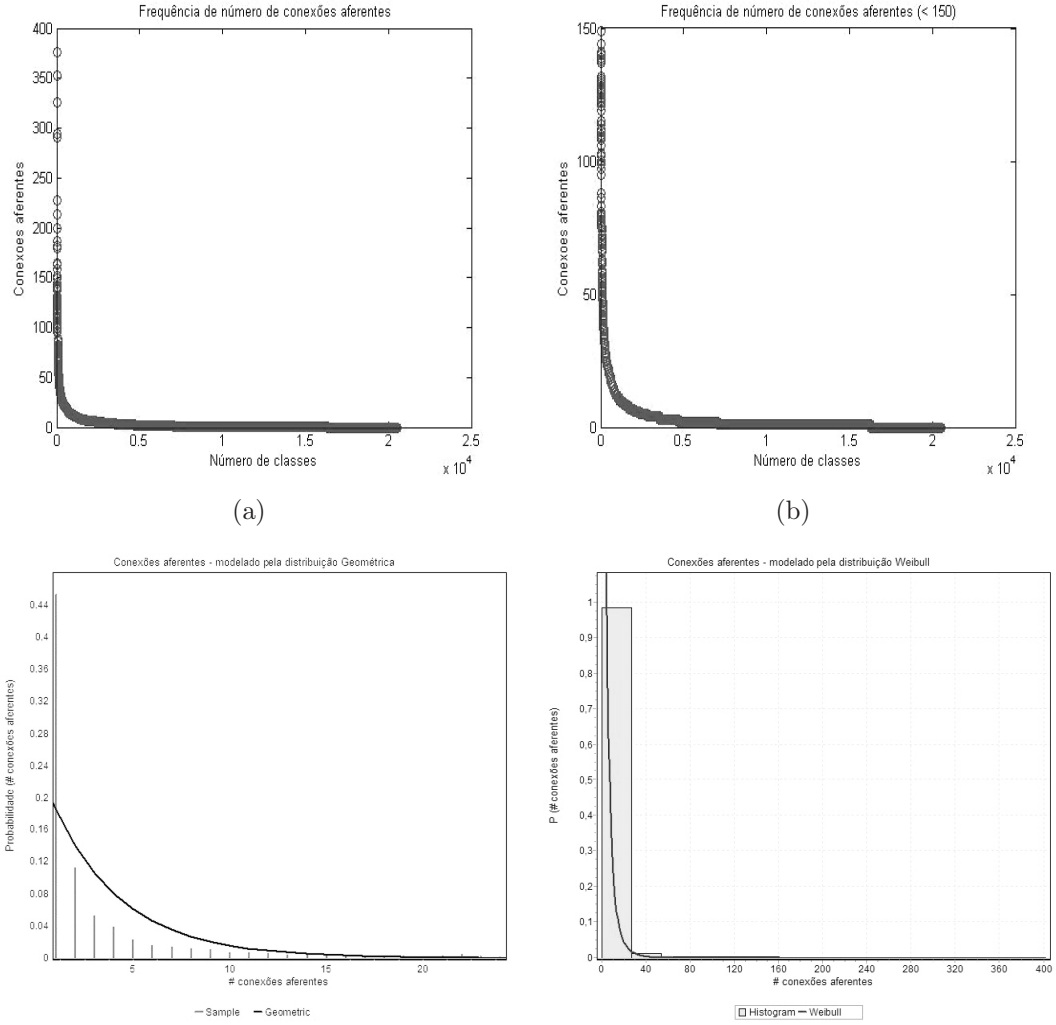


Figura 6.2. Conexões aferentes - gráficos de dispersão do conjunto completo e de valores menores do que 150. Ajustes às distribuições Geométrica e Weibull.

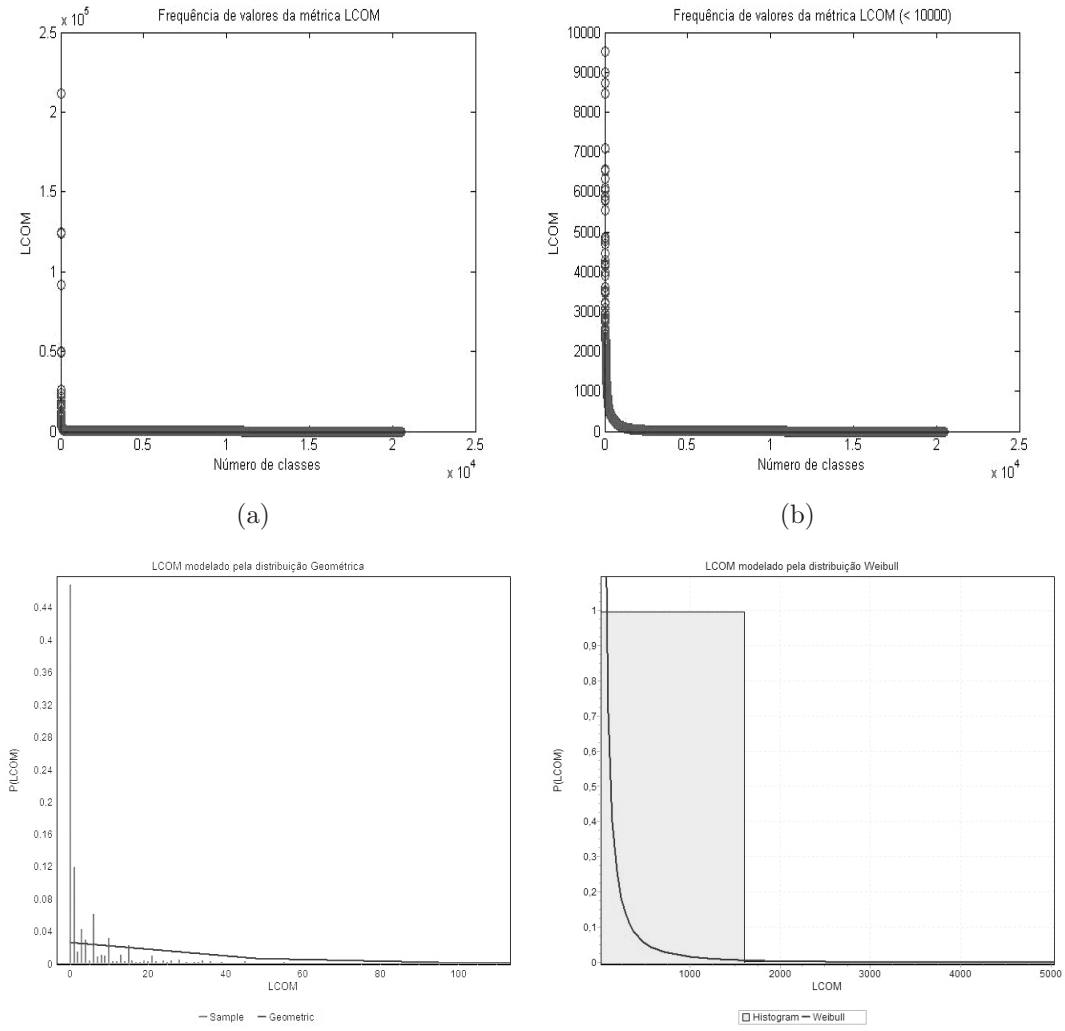
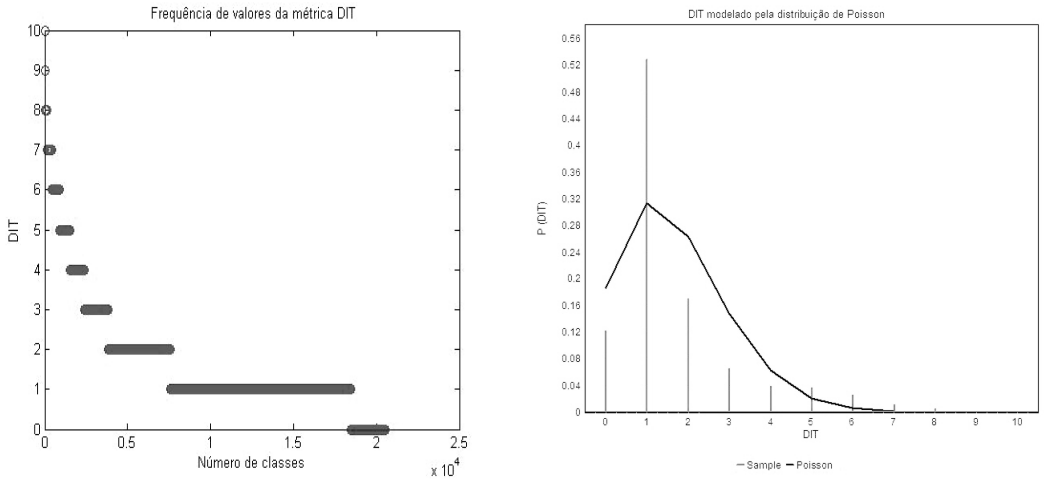


Figura 6.3. LCOM - gráfico de dispersão do conjunto completo e valores menores do que 10000. Ajustes às distribuições Geométrica e Weibull

do que 10000. O ajuste realizado mostra que os valores desta métrica podem ser modelados pela distribuição Geométrica com parâmetro $p = 0,02645$, porém, a inspeção do gráfico, da Figura 6.3, mostra que esta distribuição não é perfeitamente adequada aos dados. Os dados podem também ser modelados pela distribuição de Weibull, com parâmetros $\alpha = 0,23802$ e $\beta = 1,465$. Esta distribuição mostra-se mais adequada aos dados. Pela análise gráfica da distribuição de valores desta métrica, cerca de 50% das classes têm LCOM igual a zero, o que indica bom nível de coesão. Encontram-se classes com LCOM entre 0 e 20 com uma frequencia relativamente baixa, menor do que 12%. A probabilidade de ocorrência de classes com LCOM maior do 20 tende a zero.



(a) (b)

Figura 6.4. DIT - gráfico de dispersão e ajuste à distribuição de Poisson.

DIT

Os gráficos da Figura 6.4 mostram as dispersões dos valores da métrica DIT e o ajuste à distribuição Poisson, com parâmetros $\lambda = 1,6818$ e $\beta = 3,2228$. Na distribuição de Poisson, o parâmetro $\lambda = 1,6818$ indica a média dos valores da variável aleatória. O valor referência sugerido pela análise dos dados para DIT é 2.

Atributos públicos

O gráfico da Figura 6.5 apresenta a dispersão da quantidade de atributos públicos de uma classe. A análise gráfica da dispersão mostra que há uma pequena quantidade de classes com um número muito alto de atributos públicos e a maior parte tem um número muito pequeno de atributos públicos, sendo que este número tende bruscamente a zero. O mesmo ocorre quando se consideram, por exemplo, apenas as classes que têm menos de 100 atributos públicos. Esta métrica pode ser modelada pela distribuição de Geométrica, com parâmetro $p = 0,48941$, e pela Weibull, com parâmetros $\alpha = 0,71008$ e $\beta = 4,4001$, como mostra a Figura 6.5. A análise gráfica mostra que mais de 75% das classes não possuem atributos públicos. A ocorrência de classes que possuem entre 1 e 8 atributos públicos é muito baixa, sendo menor do que 8%. A probabilidade de ocorrências de classes com mais de 8 atributos públicos tende a zero.

Métodos públicos

A dispersão da quantidade de métodos públicos de uma classe é mostrada no gráfico da Figura 6.6. Esta métrica pode ser modelada pela distribuição de Geométrica,

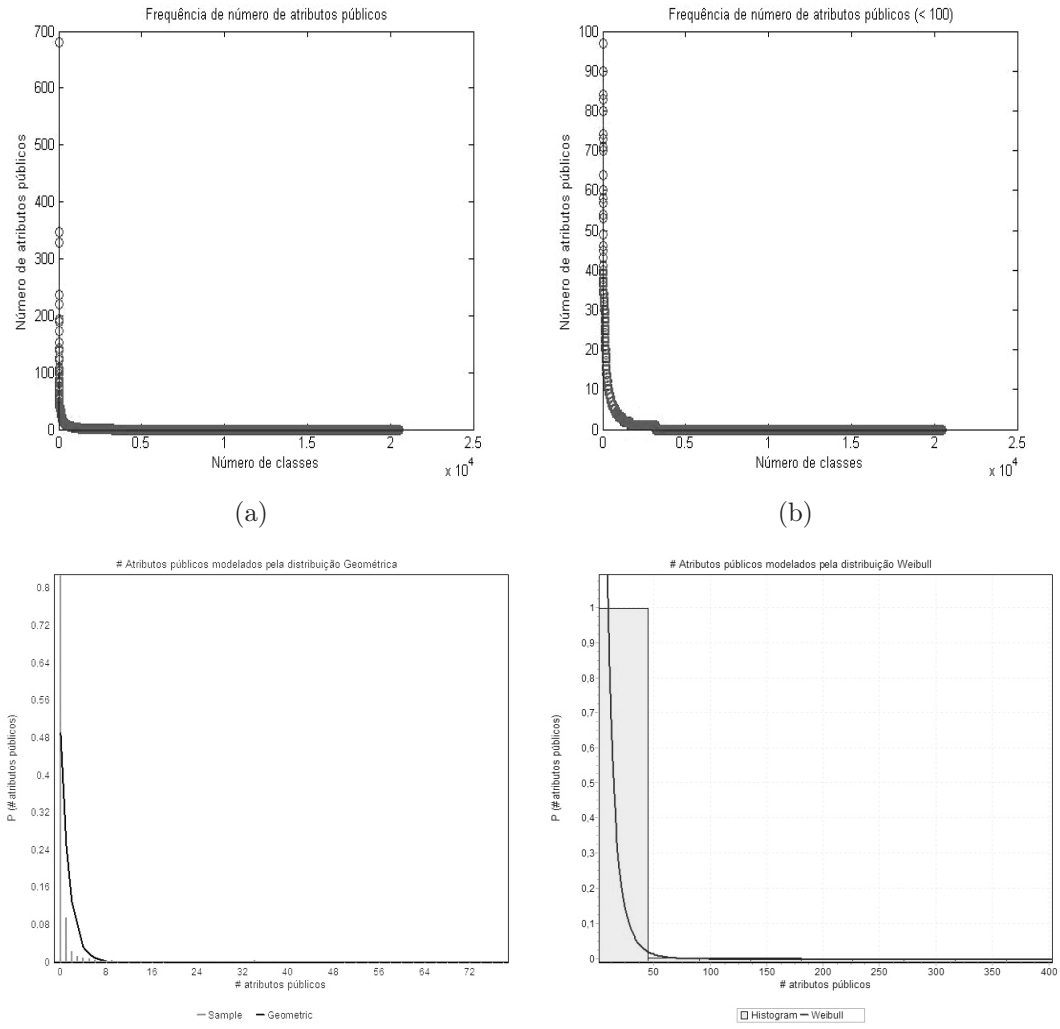


Figura 6.5. Atributos públicos - Gráfico de dispersão dos valores do conjunto completo e valores menores do que 100. Ajuste às distribuições Geométrica e Weibull.

com parâmetro $p = 0,14317$, e pela Weibull, com parâmetros $\alpha = 0,85938$ e $\beta = 5,6558$, como mostra a Figura 6.6. A análise gráfica da dispersão mostra que há uma pequena quantidade de classes com um número muito alto de métodos públicos e a maior parte tem um número muito pequeno de métodos públicos. O mesmo ocorre quando se consideram, por exemplo, apenas as classes que têm menos de 200 métodos públicos. A probabilidade de ocorrência de classes com mais de 40 métodos públicos tende a zero. Ocorrências de classe com número de métodos públicos entre 10 e 40 é relativamente baixa, e a maior parte das classes têm entre 0 e 10 métodos públicos.

6.3.3 Ajuste para os diversos domínios e para um software

O ajuste dos dados para os diversos domínios de aplicação mostrou que as mesmas distribuições de probabilidades encontradas para as métricas no experimento com o conjunto de dados completo são aplicáveis. Como são 11 domínios de aplicação e 5 métricas, e a análise foi realizada com distribuições discretas e contínuas, o que resulta em um número grande de gráficos, a título de exemplificação é mostrado o gráfico de ajuste da métrica Conexões aferentes para o domínio *Development* e para o software Talend na Figura 6.7.

6.3.4 Análise dos Resultados

A análise da distribuição dos valores da métrica COF mostra que não há um valor típico para esta métrica, e que a maior parte dos softwares OO abertos tem baixo grau de conectividade. COF é modelada pela distribuição Weibull. De acordo com o valores observados, sugere-se adotar os seguintes valores referência para COF: até 0,02 (bom), entre 0,02 e 0,14 (regular) e maior ou igual a 0,14 (ruim).

A métrica conexões aferentes indica quantas classes dependem da classe analisada. Esse valor é de grande importância para se avaliar o cuidado que se deve tomar ao alterar a classe. A distribuição de valores desta métrica mostra que para um determinada classe há um número pequeno de classes que dependem dela. Não há um valor típico para esta métrica. A partir dos resultados obtidos, sugere-se adotar os seguintes valores como referência: 1 (bom), entre 1 e 20 (regular) e maior ou igual a 20 (ruim).

LCOM também pode ser modelada pela distribuição de Weibull. Há poucos softwares com valores altos de LCOM, ou seja, há poucas classes com baixa coesão. A inspeção gráfica da distribuição de probabilidade desta métrica mostra que cerca de 50% das classes tem LCOM igual a 0, o que corresponde a um bom nível de coesão.

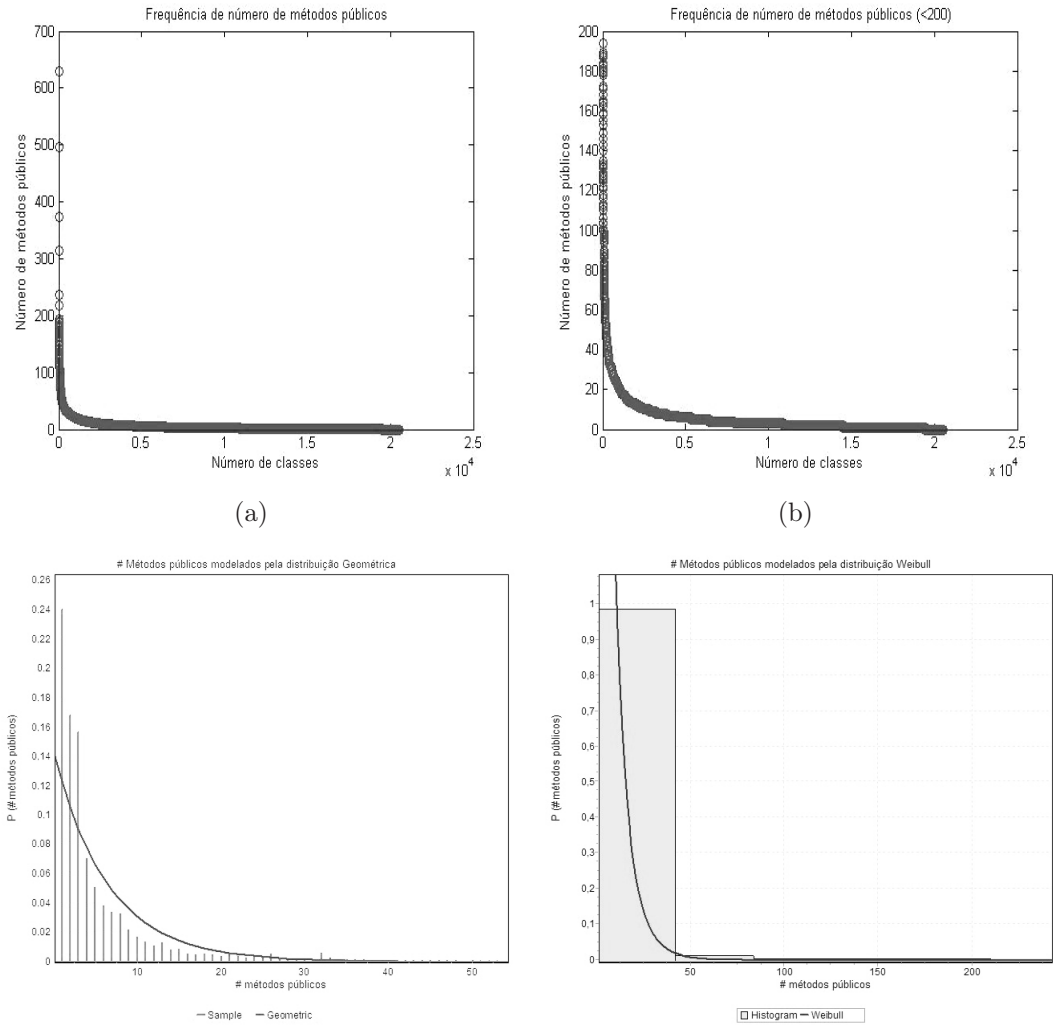
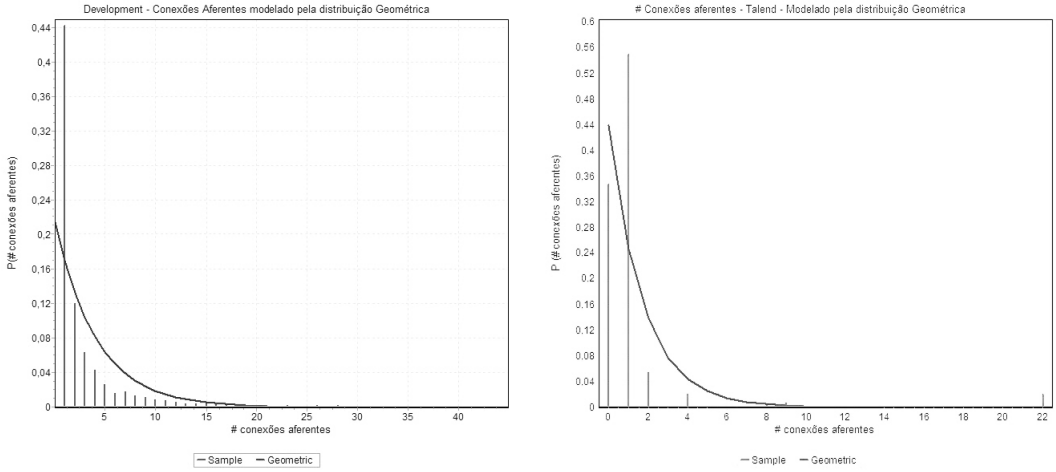


Figura 6.6. Métodos públicos - Gráfico de dispersão dos valores do conjunto completo e valores menores do que 200. Ajuste às distribuições Geométrica e Weibull.



(a) (b)

Figura 6.7. Conexões aferentes - Domínio *Development* e Talend. Modelagem pela distribuição Geométrica.

Sugere-se adotar a seguinte escala como referência para LCOM: 0 (bom), entre 0 e 20 (regular) e maior ou igual a 20 (ruim).

A distribuição de Poisson modela os valores das métricas DIT dos softwares OO desenvolvidos em Java. Desta forma, há uma probabilidade muito pequena de ocorrer valores altos para esta métrica. A média de DIT na amostra avaliada é de 1,68, aproximadamente 2. Esse valor pode ser tomado como referência para a construção de software OO, ou seja, pode-se considerar que é desejável que uma classe esteja no máximo no terceiro nível na árvore de herança.

As quantidades de atributos públicos de métodos públicos podem ser modeladas pela distribuição Geométrica e pela Weibull, o que indica que há poucas classes com número alto de atributos públicos e poucas classes com um número alto de métodos públicos. Isso indica que as classes seguem os princípios de ocultação de informação e têm interface pequena, ou seja, exportam poucos serviços. Isso é condizente com o fato de terem alta coesão interna, mostrado pelos baixos valores da métrica LCOM. A análise gráfica da distribuição das quantidade de atributos públicos mostra que mais de 75% das classes não possuem atributos públicos. Isso já tem sido uma diretriz para a construção de software OO. Pelos dados obtidos, sugerem-se os seguintes valores referência para a quantidade de atributos públicos de uma classe: 0 (bom), entre 0 e 8 (regular) e maior ou igual a 8 (ruim). A análise gráfica da distribuição de valores de métodos públicos mostra que a probabilidade de uma classe possuir mais de 40 métodos aproxima-se de zero. Os seguintes valores referência são sugeridos para essa

métrica: de 0 a 10 (bom), entre 10 e 40 (regular) e maior ou igual a 40 (ruim).

Os valores sugeridos como referência para as métricas COF, LCOM, DIT, conexões aferentes, quantidade de atributos públicos e de métodos públicos obtidos a partir da análise dos resultados do experimento são sumarizados na Tabela 6.2.

Tabela 6.2. Valores referência para métricas OO

Fator	*Nível*	*Métrica*	*Valor Referência*
Conectividade	Sistema	COF	Bom: até 0,02 - Regular: entre 0,02 e 0,14 - Ruim: maior ou igual a 0,14
	Classe	# conexões aferentes	Bom: 1 - Regular: entre 1 e 20 - Ruim: maior ou igual a 20
Ocultação de Informação	Classe	# atributos públicos	Bom: 0 - Regular: entre 0 e 8 - Ruim: maior ou igual a 8
Tamanho da Interface	Classe	# métodos públicos	Bom: 0 a 10 - Regular: entre 10 e 40 - Ruim: maior ou igual a 40
Herança	Classe	DIT (profundidade na árvore de herança)	Valor típico: 2
Coesão Interna	Classe	LCOM (ausência de coesão)	Bom : 0 - Regular: entre 0 e 20 - Ruim: maior ou igual a 20

6.4 Conclusão

Este trabalho apresenta um estudo sobre caracterização de software OO abertos. No estudo foram utilizados 40 softwares desenvolvidos em Java de 11 domínios de aplicação diferentes, em um total de 26.202 classes. Investigou-se se as métricas COF (conectividadade), LCOM (coesão), DIT (profundidade da árvore de herança), quantidade de conexões aferentes, quantidade de atributos públicos e quantidade de métodos públicos podem ser modeladas por uma distribuição de probabilidade. O estudo concluiu que a distribuição de Weibull modela os valores de: COF, o que indica que poucos softwares abertos apresentam alto grau de conectividade; conexões aferentes, o que indica que há poucas classes com um número alto de classes dependentes; LCOM, o que mostra que a maior parte das classes tem bom nível de coesão; quantidade de atributos públicos, o que evidencia que a ocultação de informação tem sido devidamente empregada; quantidade de métodos públicos, o que informa que a quantidade de serviços exportados por uma classe tende a ser pequena. A distribuição de Poisson modela os valores da métrica DIT, o que torna possível identificar um valor típico para essa métrica.

Com base nos resultados dos experimentos deste trabalho propõe-se utilizar os valores das métricas identificados nos softwares abertos como referência para a construção de software de qualidade. Este é um dos primeiros trabalhos na direção de se identificar esses valores. Isso é uma questão de grande relevância para o uso efetivo de métricas na construção de software. Neste trabalho foram investigados os valores de seis importantes métricas de software OO. Sugere-se utilizar o método aqui empregado para o estudo de outras métricas.

O estudo descrito neste capítulo compõe o artigo *Valores Referência para Métricas de Software Orientado por Objetos*, aceito para publicação no Simpósio Brasileiro de Engenharia de Software de 2009.

Capítulo 7

KB3 - Modelo de Predição de Tempo de Manutenção de Software

O objetivo principal deste trabalho é a definição de um modelo de predição de tempo de manutenção de software orientado por objetos. Neste capítulo é apresentado o modelo proposto.

7.1 Enunciado do Problema

Um software é constituído por módulos que interagem entre si. Na orientação por objetos, estes módulos são tipicamente as *classes*. Classes definem as características e o comportamento de seus *objetos*. A construção de uma classe envolve a definição de seu *contrato* [Meyer, 1997], definido pelo conjunto de serviços exportados pela classe, as pré e pós condições destes serviços. A comunicação entre objetos se dá pelos contratos definidos em suas classes: um objeto A que utiliza determinado serviço de outro objeto B deve satisfazer as pré-condições para o uso de tal serviço, e o objeto B deve, então, garantir a execução correta do serviço solicitado, gerando o resultado esperado, que corresponde às pós-condições do serviço. A base da construção de um software orientado por objetos é a ocultação de informação, ou seja, detalhes de implementação do serviço devem ser transparentes para o usuário do serviço. A interação entre objetos se dá apenas por meio da interface de sua classes, ou seja, dos serviços exportados por ela. Desta forma, se houver uma alteração na implementação de determinado serviço, mas não houver alteração no *contrato* da classe, o usuário do serviço não sofrerá impacto.

Em relação à atividade de manutenção, os módulos de um software podem assumir dois estados: *em manutenção* e *atualizado*. Um módulo assume o estado *em manutenção* se estiver passando por uma alteração que gere impacto na sua interface.

Atividades de manutenção em um módulo que não alterem a sua interface podem ser desprezadas neste modelo, pois módulos conectam-se uns aos outros por meio de suas interfaces. Desta forma, se um módulo sofreu manutenção mas a sua interface permanece a mesma, os outros módulos conectados a eles não sofrerão impacto desta alteração, ou seja, os estados dos demais módulos não serão afetados em decorrência da manutenção realizada. Ao término desta atividade de manutenção, o módulo passa para o estado *atualizado*. Isto posto, a questão investigada pelo modelo proposto é o tempo de estabilização de um software quando um ou mais módulos sofrem manutenção que alterem suas interfaces. O problema investigado é enunciado a seguir.

Seja um software com n módulos. Um módulo pode assumir dois estados: *em manutenção* e *atualizado*. Se um módulo está *em manutenção* em um dado momento, seja x a probabilidade de ele passar para o estado *atualizado* no próximo intervalo de tempo t; se um módulo está *atualizado*, seja y a probabilidade de ele passar para o estado *em manutenção* no próximo intervalo de tempo t se ele estiver conectado a algum outro módulo que esteja em manutenção, sendo que y é proporcional ao número de módulos em manutenção aos quais está conectado. Por outro lado, se um módulo estiver atualizado, permanecerá assim enquanto todos os módulos a ele conectados estiverem atualizados. Diz-se que o software atingiu o equilíbrio quando todos os módulos estiverem atualizados.

Uma estrutura provável para a organização de módulos em um software orientado por objetos é seu particionamento em agrupamentos de classes. Uma partição é um grupo de módulos conectados entre si. Em um sistema pode haver mais de uma partição e em cada partição há grande número de conexões entre suas classes. Idealmente deve haver poucas conexões entre classes de partições diferentes.

Dada uma situação inicial do software na qual a módulos estão em manutenção, deseja-se saber o tempo de equilíbrio E do software em função de n, t, q, a, g, c_i, onde:

- n: número de módulos no software;
- t: tempo da atividade de manutenção com impacto na interface do módulo;
- q: probabilidade de mudança de estado do módulo;
- a: número de módulos inicialmente no estado em manutenção;
- g: número de partições de módulos.
- c_i: cardinalidade da partição i tal que $\sum_{i=1}^{g} c_i = n$

São exemplos de formas como os módulos de um software podem estar conectados, mas não necessariamente correspondem a forma como módulos estão conectados em software reais:

1. Inexistência de conexões entre os módulos. Neste caso, g = n.

2. Software constituído por partições de m módulos, sendo que as partições são independentes entre si e cada uma delas é totalmente conectado. Neste caso, g = n/m.

3. Software totalmente conectado: nesta configuração, cada módulo possui uma conexão com todos os demais módulos do software. Neste caso, g = 1.

Myers [1975] modela de forma simplificada este problema como um circuito de lâmpadas. Neste modelo, as lâmpadas correspondem aos módulos. Uma lâmpada pode assumir dois estados: *acesa* e *apagada*. O estado da lâmpada corresponde à ocorrência de manutenção no módulo: *acesa* indica módulo em manutenção e *apagada* indica módulo em que não está ocorrendo manutenção. Se uma lâmpada está acesa, a probabilidade desta passar para o estado de apagada no próximo segundo é de 50%; se uma lâmpada está apagada, a probabilidade desta passar para o estado de acesa no próximo segundo é de 50% se estiver conectada a alguma outra lâmpada acesa. Por outro lado, se uma lâmpada estiver apagada, permanecerá assim enquanto todas as lâmpadas conectadas a ela estiverem apagadas. Diz-se que o circuito atingiu o equilíbrio quando todas as lâmpadas estiverem apagadas.

Para o caso em que inicialmente todas as lâmpadas do circuito estão acesas, Myers determina o tempo médio de equilíbrio do circuito nas seguintes situações:

1. Inexistência de conexões entre as lâmpadas: o tempo de equilíbrio do circuito é de 7 segundos.

2. Circuito constituído por agrupamentos de 10 lâmpadas, sendo que os agrupamentos são independentes entre si e cada um deles é totalmente conectado: neste caso, o tempo de equilíbrio do circuito é de 20 minutos.

3. Circuito totalmente conectado: nesta configuração, cada lâmpada possui uma conexão com todas as demais lâmpadas do circuito. Esse é o pior caso, pois o tempo para atingir o equilíbrio do circuito é de 10^{22} anos.

O modelo de Myers mostra a importância do grau de conectividade de um software na sua dificuldade de estabilização. Trata-se de um modelo informal e qualitativo,

para o qual não há qualquer demonstração. A abordagem proposta neste trabalho é estabelecer formalmente um meio de estimar o custo de manutenção de um software com maior acurácia. O modelo de Myers considera apenas o fator conectividade, que também consideramos ser o mais importante. Porém, para uma estimativa com melhor acurácia, outros fatores devem ser considerados. Por exemplo, dois softwares com o mesmo número de módulos e o mesmo número de conexões podem ter custos de manutenção distintos se, por exemplo, em um deles as classes apresentarem melhor coesão do que as do outro e se o grau de acoplamento entre suas classes forem menores do que no outro software. As seções seguintes apresentam a solução proposta para este problema.

7.2 Definição do Modelo de Estabilidade de Software

Esta seção descreve o modelo de estabilidade de software proposto neste trabalho. O modelo visa definir o tempo de manutenção de um software constituído por n módulos, no qual inicialmente i desses módulos sofrerão manutenção. O modelo não considera aspectos como o número de pessoas envolvidas na tarefa de manutenção ou a produtividade da equipe.

Seja um sistema constituído por n módulos, onde $S = \{0, 1, 2, ..., n\}$ denota o conjunto de estados possíveis para o sistema. Cada estado corresponde ao número de módulos em manutenção em um dado instante. Quando o número de módulos em manutenção for igual a zero, diz-se que o sistema alcançou estabilidade. Deseja-se saber o tempo necessário para que um sistema chegue à estabilidade dado o seu estado, ou seja, dado o número de módulos em manutenção.

No sistema, dado que há i ($i \neq 0$ e $i \neq n$) módulos em manutenção, no próximo instante só é possível ir para o estado em que haverá *i+1* módulos em manutenção ou para o estado em que haverá *i-1* módulos em manutenção. Este problema pode ser modelado como uma Cadeia de Markov[1] [Petrov & Mordecki., 2003]. Em uma Cadeia de Markov, se i é o estado presente, a probabilidade de passar para os estados futuros *i+1*, ... ,*i+m* depende somente do estado presente i e não dos estados anteriores a i. Em outras palavras, o número de módulos em manutenção nos instantes futuros depende somente do número de módulos em manutenção no instante presente. A Figura 7.1 ilustra o Passeio Aleatório [Petrov & Mordecki., 2003] para o problema. Dado um

[1] A elaboração deste modelo somente foi possível devido à preciosa colaboração do Prof. Bernardo de Lima, do Departamento de Matemática da UFMG

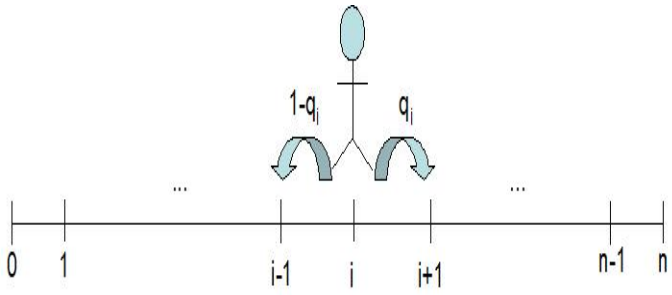


Figura 7.1. Passeio Aleatório

sistema no estado i, a probabilidade de passar para o estado *i+1* é q_i, e a probabilidade de no próximo instante passar para *i-1* módulos em manutenção é $1 - q_i$.

O Passeio Aleatório é representado por uma matriz de probabilidades $P = (p_{ij})$, na qual i referencia as linhas e j, as colunas, e cada posição ij na matriz representa a probabilidade de passar de i módulos em manutenção para j módulos em manutenção em uma unidade de tempo t, considerada constante para toda manutenção.

Para este problema temos as seguintes definições:

1. O estado em que não há módulos em manutenção é o *estado absorvente*, o estado em que o sistema atingiu estabilidade. A probabilidade de sair deste estado é 0, e a probabilidade de permanecer nele é 1. A matriz de probabilidades, na linha 0, tem valor 1 na coluna 0, e 0 nas demais colunas.

$$p_{0j} = \begin{cases} 1 & \text{se } j = 0 \\ 0 & \text{se } j \neq 0 \end{cases}$$

2. No estado em que todos os módulos estão em manutenção, só é possível passar para o estado em que haverá *n-1* módulos em manutenção. A probabilidade de ir para o estado *n+1* é 0, e a probabilidade de ir para o estado *n-1* é 1. A matriz de probabilidades, na linha n, tem valor 1 na coluna *n-1*, e valor 0 nas demais colunas.

$$p_{nj} = \begin{cases} 1 & \text{se } j = n-1 \\ 0 & \text{se } j \neq n-1 \end{cases}$$

3. Nos estados intermediários i, $i\epsilon\{1, 2, 3, ..., n-1\}$, nos quais há de *1* a *n-1* módulos em manutenção, não é possível passar para estados diferentes de *i+1* e *i-1*. A

probalidade de passar para o estado *i+1* é q_i, e a probabilidade de passar para *i-1* é $1 - q_i$.

$$p_{ij} = \begin{cases} 0 & \text{se } j \neq i+1, i-1 \\ q_i & \text{se } j = i+1 \\ 1 - q_i & \text{se } j = i-1 \end{cases}$$

Pelas características do problema de estabilidade de software e *supondo-se que no sistema cada um dos módulos possui conexão com todos os demais*, é razoável concluir que a partir do estado i a probabilidade de ir para o estado *i+1* é:

- proporcional ao número de módulos que não estão em manutenção (*n-i*): quanto maior o número de módulos que não estão em manutenção, maior a chance que pelo menos um deles entre em manutenção.
- proporcional ao número de módulos em manutenção (i): quanto mais módulos em manutenção, maior a probabilidade de que outros módulos do sistema entrem em manutenção.

Desta forma, a partir de um estado i, a probabilidade de ir para *i+1* é dada pela Equação 7.1. Utilizamos o termo "taxa de contaminação" para designar a quantidade de módulos nos quais será necessário realizar manutenção em decorrência do número de módulos em manutenção no instante corrente no sistema. Nesta equação, α é um parâmetro que indica a "taxa de contaminação" do sistema. Deve ser um fator que quanto maior, maior também a "taxa de contaminação". Um bom candidato a este fator é *grau de acoplamento* [Myers, 1975] entre os módulos do software, pois quanto maior o grau de acoplamento, maior o impacto de uma manutenção de um módulo nos demais.

$$p_{i,i+1} \propto k_1 \ \alpha(n-i)i \tag{7.1}$$

É possível concluir também que a partir do estado i, a probabilidade de se passar para o estado *i-1* é proporcional ao número de módulos que estão em manutenção. Esta probabilidade é dada pela Equação 7.2. Utilizamos o termo "taxa de melhora" para designar o número de módulos que terão a manutenção finalizada sem implicar na manutenção de outros módulos do sistema. Nesta equação, o parâmetro β indica a "taxa de melhora" do sistema. Quanto maior for β, maior a probabilidade das manutenções dos módulos serem finalizadas sem afetar outros módulos. Deve ser um fator que

	1	2	3	4	...	n-1	n	0
1	**0**	q_1	0	0	0	0	0	$1-q_1$
2	$1-q_2$	**0**	q_2	0	0	0	0	0
3	0	$1-q_3$	**0**	q_3	0	0	0	0
4	0	0	$1-q_4$	**0**	...	0	0	0
...	...	...	...	...	...	...	...	...
n-1	0	0	0	0	$1-q_{n-1}$	**0**	q_{n-1}	0
n	0	0	0	0	0	1	**0**	0
0	0	0	0	0	0	0	0	1

Figura 7.2. Matriz de probabilidades

quanto maior, maior também a "taxa de melhora". Um bom candidato a estimar este fator é grau de *coesão interna dos módulos* [Myers, 1975].

$$p_{i,i-1} \propto k_2 \ \beta i \tag{7.2}$$

Como a soma de $p_{i,i+1}$ e $p_{i,i-1}$, $i\epsilon\{1,2,3,...,n-1\}$, deve ser igual a 1, obtem-se as Equações 7.3 e 7.4.

$$p_{i,i+1} = \frac{\alpha(n-i)i}{\alpha(n-i)i + \beta i} \tag{7.3}$$

$$p_{i,i-1} = \frac{\beta i}{\alpha(n-i)i + \beta i} \tag{7.4}$$

O problema investigado consiste em saber quanto tempo o sistema levará para sair de um estado i e chegar ao *estado absorvente*. Ou seja, busca-se o tempo médio para o sistema estabilizar dado que há i módulos em manutenção, referenciado aqui como $E(t_i)$.

Há um teorema [Grinsteas & Snell., 1991] para resolver a seguinte questão em uma Cadeia de Markov: dado que a cadeia encontra-se no estado i, qual é o número esperado de passos para que a cadeia alcance o estado absorvente. Para que este teorema, que será definido posteriormente, possa ser aplicado ao problema, constrói-se uma matriz de probabilidades $(n+1)\times(n+1)$, onde n é o número total de módulos do

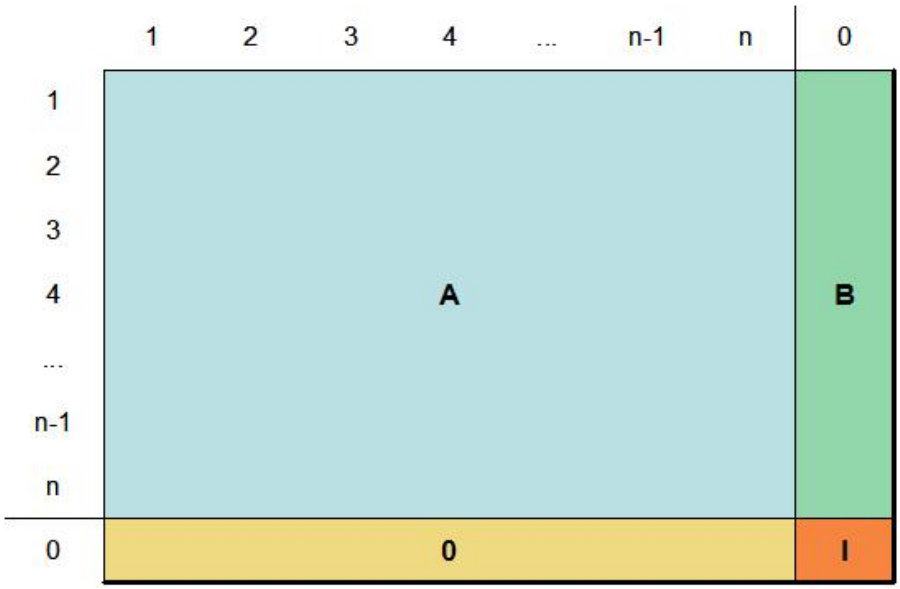


Figura 7.3. Forma da matriz de probabilidades

sistema, e esta matriz deve estar na forma canônica: o estado absorvente corresponde à última linha e à última coluna da matriz. A matriz de probabilidades é preenchida como mostra a Figura 7.2. Nesta matriz, para cada linha *i*, somente os elementos correspondentes às colunas *i+1* e *i-1* podem ter valores diferentes de 0. No caso da linha 0, que corresponde ao estado absorvente, a coluna 0 tem valor 1 e as demais têm valor 0.

A forma da matriz de probabilidades é mostrada na Figura 7.3. Nesta matriz: A é uma matriz $n \times n$; *0* é uma matriz linha de n elementos de valor 0; I é a matriz identidade; e B é uma matriz coluna de n elementos, na qual o elemento da linha 1 pode ter valor diferente de 0 e os demais têm valor igual 0.

O referido teorema define que:

Teorema: Seja $E(t_i)$ o número esperado de passos antes da cadeia ser absorvida, dado que a cadeia inicia no estado i, e seja t o vetor coluna cuja i-ésima entrada é $E(t_i)$. Então $t = Nc$, onde $N = (I - A)^{-1}$ e c é um vetor coluna no qual todas as entradas são iguais a 1.

Aplicando-se o Teorema ao problema, temos:

$$\begin{pmatrix} E(t_1) \\ E(t_2) \\ E(t_3) \\ \vdots \\ E(t_n) \end{pmatrix} = (\mathbf{I} - \mathbf{A})^{-1} \begin{pmatrix} 1 \\ 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix}$$

Os dois vetores desta igualdade têm n *elementos*. Cada posição do vetor resultante fornece $E(t_i)$, que corresponde ao tempo esperado para que o sistema chegue ao estado absorvente a partir do estado i.

Para saber o tempo esperado para que o sistema alcance a estabilidade quando todos os módulos estão em manutenção, deve-se calcular o valor do vetor resultante na linha n. Isso é dado pela multiplicação entre a última linha da matriz $(I - A)^{-1}$ e o vetor de valores 1, ou seja, é a soma dos elementos da última linha da matriz $(I - A)^{-1}$.

7.2.1 Resultados

Com o auxílio do software Matlab, o vetor E foi calculado para $n\epsilon\{4, 5, 6, ..., 11, 20, 25\}$. Os resultados deste cálculo são apresentados no Anexo A. A análise dos dados obtidos levou a concluir que o tempo de estabilidade de um software cresce fatorialmente em função do tamanho do software. A seguir esta análise é discutida.

O tempo de estabilidade de um software com n módulos dentre os quais i módulos passam por manutenção, $E(t_i)$, é dado por um polinômio de n termos, na variável α/β, com a formação mostrada na Equação 7.5. Por exemplo, o tempo de estabilidade do sistema com 5 módulos no qual há 1 módulo em manutenção é dado pelo polinômio mostrado na Equação 7.6.

$$Et_i = i + c_1 \frac{\alpha}{\beta} + c_2 \frac{\alpha^2}{\beta^2} + c_3 \frac{\alpha^3}{\beta^3} + ... + c_{n-1} \frac{\alpha^{n-1}}{\beta^{n-1}} \tag{7.5}$$

$$E1 = 1 + 8 \frac{\alpha}{\beta} + 24 \frac{\alpha^2}{\beta^2} + 48 \frac{\alpha^3}{\beta^3} + 48 \frac{\alpha^4}{\beta^4} \tag{7.6}$$

A inspeção dos polinômios gerados mostra que para um determinado valor de n os últimos termos dos polinômios $E(t_i)$ são iguais. Por exemplo, todos os polinômios obtidos para $n = 5$ têm o último termo igual a $48 \frac{\alpha^4}{\beta^4}$. Os valores dos últimos termos dos polinômios para $n\epsilon\{4, 5, 6, ..., 11, 20, 25\}$ são mostrados na tabela 7.1. Observa-se que a constante deste último termo é dada em função de n pela Equação 7.7. O último

termo é dado pela Equação 7.8.

O parâmetro α indica a "taxa de contaminação" do sistema, e β, a sua "taxa de melhoria". Quanto maior o valor de α, maior também o tempo que o sistema levará para estabilizar. Quanto menor o valor de β, maior o tempo de estabilização do sistema. Se, por exemplo, acoplamento médio entre os módulos do sistema for usado para o parâmetro α e coesão média dos módulos for usada para o parâmetro β, significaria dizer que quanto maior o acoplamento e menor a coesão, maior o tempo de estabilização, ou seja, maior a dificuldade de manutenção do sistema.

O tempo de estabilização de um sistema é da ordem de $n! \frac{\alpha^{n-1}}{\beta^{n-1}}$. Ou seja, tomando-se apenas o último termo dos polinômios já é possível concluir que o esforço de manutenção de software é explosivo. Na prática, isso traduz-se na impossibilidade de garantir a estabilidade de um software fortemente conectado.

n	Último termo
4	$12 \frac{\alpha^3}{\beta^3}$
5	$48 \frac{\alpha^4}{\beta^4}$
6	$240 \frac{\alpha^5}{\beta^5}$
7	$1440 \frac{\alpha^6}{\beta^6}$
8	$10080 \frac{\alpha^7}{\beta^7}$
9	$80640 \frac{\alpha^8}{\beta^8}$
10	$725760 \frac{\alpha^9}{\beta^9}$
11	$7257600 \frac{\alpha^{10}}{\beta^{10}}$
20	$2432902008176640000 \frac{\alpha^{19}}{\beta^{19}}$
25	$124089680346647887872000 0 \frac{\alpha^{24}}{\beta^{24}}$

Tabela 7.1. Último termo do polinômio $E(t_i)$

$$c = (n-2)!(2n-2) \tag{7.7}$$

$$E(n) = (n-2)!(2n-2)\,\frac{\alpha^{n-1}}{\beta^{n-1}} \tag{7.8}$$

7.3 Aproximação do Modelo Proposto a Software Real

No modelo proposto na Seção 7.2.1 considerou-se que *no sistema cada um dos módulos possui conexão com todos os demais*. Entretanto, um software real não possui esta configuração necessariamente. O trabalho proposto visa generalizar o modelo descrito anteriormente no sentido de obter um modelo mais próximo da configuração de softwares reais. Este modelo proposto é descrito a seguir.

A métrica COF (*Coupling Factor*) proposta por Abreu & Carapuça [1994] indica o grau de conectividade de um software orientado por objetos. Esta métrica é dada pela razão entre o número de conexões existentes no software e o maior número possível de conexões que podem existir no software. Como um software é modelado como um grafo direcionado, em um software com n classes, o maior número possível de conexões é $n^2 - n$. Um software totalmente conectado possui COF igual a 1, e um software sem conexões possui COF igual a 0. Embora não exista um estudo amplo sobre como os módulos de um software interagem entre si, o estudo realizado neste trabalho, descrito na Seção 6, com softwares implementados em Java mostra que o grau de conectividade médio dos sistemas é 0,021, ou seja, do total de conexões que poderiam existir no software, em média há 2,1% delas.

O grau de conectividade de um software, dado pela métrica COF, representa satisfatoriamente o formato do grafo referente ao software. Como os valores do grau de conectividade variam de 0 a 1, COF pode ser tomada como a probabilidade de uma conexão entre duas classes existir. Esta métrica será, então, introduzida no modelo proposto, sendo referenciada como ϕ. Com isso, o modelo passa a considerar as conexões que de fato existem no software.

Quanto maior o grau de conectividade de um software, maior a chance de uma alteração em um módulo qualquer afetar os demais módulos do sistema. Desta forma, a partir de um estado i, a probabilidade de ir para o estado *i+1* é proporcional a ϕ, conforme mostra a Equação 7.9.

$$p_{i,i+1} \propto k_1 \alpha \phi (n - i) i \tag{7.9}$$

A partir de um estado i, a probabilidade de ir para o estado *i-1* é dada pela Equação 7.10.

$$p_{i,i-1} \propto k_2 \beta i \tag{7.10}$$

Como a soma de $p_{i,i+1}$ e $p_{i,i-1}$, $i\epsilon\{1,2,3,...,n-1\}$, deve ser igual a 1, obtem-se as Equações 7.11 e 7.12.

$$p_{i,i+1} = \frac{\alpha\phi(n-i)i}{\alpha\phi(n-i)i + \beta i} \tag{7.11}$$

$$p_{i,i-1} = \frac{\beta i}{\alpha\phi(n-i)i + \beta i} \tag{7.12}$$

Com o auxílio do software Matlab, o vetor E foi calculado novamente considerando-se a inclusão do fator ϕ nos cálculos das probabilidades utilizadas no modelo. Os resultados obtidos, mostrados no Anexo A, são similares aos do modelo anterior, porém surge o fator ϕ nas fórmulas.

O tempo de estabilidade de um software com n módulos dentre os quais i módulos passam por manutenção, $E(t_i)$, é dado por um polinômio de n termos com a formação mostrada na Equação 7.13.

$$Et_i = i + c_1\frac{\alpha\phi}{\beta} + c_2\frac{\alpha^2\phi^2}{\beta^2} + c_3\frac{\alpha^3\phi^3}{\beta^3} + ... + c_{n-1}\frac{\alpha^{n-1}\phi^{n-1}}{\beta^{n-1}} \tag{7.13}$$

O tempo de estabilização de um software com n módulos, grau de conectividade ϕ, grau de acoplamento médio α e grau de coesão interna de módulos média β é da ordem de $n!\,\frac{\alpha^{n-1}\phi^{n-1}}{\beta^{n-1}}$.

7.4 KB3

A fórmula obtida na seção anterior foi implementada em *Connecta* e alguns experimentos preliminares foram realizados. Os valores obtidos não foram satisfatórios porque foram, em geral, muito próximos de zero, o que não fornece informação útil e impede o seu uso efetivo. Desta forma, fez-se necessário investir esforço na tentativa de identificar uma fórmula mais apurada que represente os dados obtidos. A dificuldade em se fazer isso é que não há um meio automatizado via software para análise dos dados.

Após uma análise minunciosa dos dados, observou-se a existência de um padrão de formação nos polinômios que fornece o tempo de estabilização de um software com n módulos dentre os quais i módulos passam por manutenção: há um padrão na formação das constantes que acompanham os termos de um polinômio e, além disso, um polinômio é definido recursivamente em relação ao polinômio que define o tempo de estabilização quando há *i-1* módulos em manutenção. A partir desse padrão de formação, deduziu-se as Equações 7.14 e 7.15, que definem o tempo de estabilização

$t = E(n, i)$. Nestas equações, α é a taxa de contaminação, β é a taxa de melhoria e ϕ é o grau de conectividade do software, sendo que $0 \leq \alpha \leq 1$, $0 < \beta \leq 1$ e $0 \leq \phi \leq 1$.

$$E(n,1) = 1+2(n-1)(n-2)!\,\frac{\alpha^{n-1}\phi^{n-1}}{\beta^{n-1}} + \sum_{k=1}^{k=n-2} \frac{2(n-1)(n-2)!}{k!}\,\frac{\alpha^{n-k-1}\phi^{n-k-1}}{\beta^{n-k-1}} \quad (7.14)$$

$$E(n,i) = 1 + E(n,i-1) + \sum_{k=i-1}^{k=n-2} 2(n-i)\,\frac{(n-i-1)!}{(k-i+1)!}\frac{\alpha^{n-k-1}\phi^{n-k-1}}{\beta^{n-k-1}} \quad (7.15)$$

Os fatores candidatos para α e β são, respectivamente, o grau de acoplamento médio entre os módulos do software e o grau de coesão interna médio deles, e ϕ corresponde ao grau de conectividade do software. Considerando-se isso, quando $\alpha = 0$ e $\phi = 0$, tem-se a situação em que não há conexões entre os módulos, ou seja, não há interação entre eles, então não há sistema. Pelo modelo, neste caso, o tempo de manutenção será igual a i. Em uma situação real, esse valor parece razoável, considerando-se o fato de que o modelo não considera o tamanho ou a produtividade da equipe envolvida na atividade de manutenção. Em um sistema, o melhor caso ocorre quando α e ϕ assumem valores próximos de zero e β, valores próximos de 1, o que corresponde a um software em que os módulos são pouco dependentes uns dos outros e possuem boa coesão interna. Neste caso, o tempo de manutenção tende para i. O pior caso ocorre quando $\alpha = 1$, $\phi = 1$ e β assume valores próximos de 0, o que corresponde a um software totalmente conectado, com forte grau de acoplamento entre os módulos e baixo grau de coesão interna de módulos. Neste caso, o tempo de manutenção é explosivo. O estudo relatado no Capítulo 6 mostra que para softwares abertos desenvolvidos em Java, os valores de conectividade são em geral pequenos, menores do que 0,02. O estudo mostra também que os graus de coesão são muito bons, ou seja, β é, em geral, próximo de 1. O estudo não avaliou grau de acoplamento entre os módulos. Com esses valores aplicados ao modelo, o tempo de manutenção de um software aberto é relativamente pequeno, o que, de fato, é uma característica essencial para esse tipo de software.

Não tem-se prova de que estas equações modelam situações diferentes daquelas apresentadas no Anexo A, mas após a implementação de KB3 em *Connecta*, serão realizados experimentos para avaliação do modelo. O objetivo desses experimentos é verificar a hipótese que KB3 pode ser utilizada para estimar o tempo de estabilização de software e ser, então, utilizada como um indicador de sua manutenibilidade.

7.4.1 Implementação

A fórmula KB3 possui termos que são fatoriais. A implementação de programas para cálculos de fatoriais é problemática porque ocorre *overflow* para números relativamente pequenos. Para solucionar esse problema, pode-se realizar uma transformação logarítmica em KB3. No caso da função fatorial, ela pode ser generalizada para números reais por uma outra função referenciada na literatura como *função gama* [Wikipedia, 2009]. A relação entre a função fatorial e a função gama é mostrada na Equação 7.16.

$$n! = \Gamma(n+1) = \int_0^\infty t^n e^{-t} dt \tag{7.16}$$

O logaritmo natural de *n!* equivale a $ln(\Gamma(n+1))$. O cálculo deste logaritmo, conhecido como *lngamma*, pode ser realizado de forma direta, sem o cálculo prévio de *n!*. Há implementações em Java disponíveis para o seu cálculo. Uma delas faz parte do projeto *Weka 3: Data Mining Software in Java* [Weka, 2009]. A transformação logarítmica de KB3 é demonstrada a seguir.

Sejam A e B os termos da expressão *E(n,1)*, da seguinte forma:

$$E(n,1) = 1 + \underbrace{2(n-1)(n-2)!\,\frac{\alpha^{n-1}\phi^{n-1}}{\beta^{n-1}}}_{A} + \sum_{k=1}^{k=n-2} \underbrace{\frac{2(n-1)(n-2)!}{k!}\,\frac{\alpha^{n-k-1}\phi^{n-k-1}}{\beta^{n-k-1}}}_{B}$$

$$A = 2(n-1)(n-2)!\,(\frac{\alpha\phi}{\beta})^{n-1}$$

$$B = \frac{2(n-1)(n-2)!}{k!}\,(\frac{\alpha\phi}{\beta})^{n-k-1}$$

O logaritmo natural de A é dado por:

$$ln(A) = ln(2) + ln(n-1) + lngamma(n-1) + (n-1)\,(ln(\phi) + ln(\alpha) - ln(\beta))$$

Com essa transformação logarítmica, A é dado por:

$$A = exp(ln(2) + ln(n-1) + lngamma(n-1) + (n-1)\,(ln(\phi) + ln(\alpha) - ln(\beta)))$$

O logaritmo natural de B é dado por:

$$ln(B) = ln(2)+ln(n-1)+lngamma(n-1)-lngamma(k+1)+(n-1)\,(ln(\phi)+ln(\alpha)-ln(\beta))$$

Com essa transformação logarítmica, B é dado por:

$$B = exp(ln(2)+ln(n-1)+lngamma(n-1)-lngamma(k+1)+(n-k-1)\,(ln(\phi)+ln(\alpha)-ln(\beta)))$$

Desta forma, o valor de *E(n,1)* é dado por:

$$E(n,1) = 1 + A + \sum_{k=1}^{k=n-2} B$$

Seja C o termo da expressão *E(n,i)*, da seguinte forma:

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

O logaritmo natural de C é dado por:

$$ln(C) = ln(2)+ln(n-i)+lngamma(n-i)-lngamma(k-i)+(n-k-1)\,(ln(\phi)+ln(\alpha)-ln(\beta))$$

Com essa transformação logarítmica, o C é dado por:

$$C = exp(ln(2)+ln(n-i)+lngamma(n-i)-lngamma(k-i)+(n-k-1)\,(ln(\phi)+ln(\alpha)-ln(\beta)))$$

Desta forma, o valor de *E(n,i)* é dado por:

$$E(n,1) = 1 + E(i-1,n) \sum_{k=i-1}^{k=n-2} C$$

7.4.2 Considerações

KB3 baseia-se em três parâmetros: α, que corresponde à *taxa de contaminação* do software, β, que corresponde à *taxa de melhora* e ϕ, que corresponde ao grau de conectividade. Um problema central na utilização de KB3 é identificar bons fatores a serem utilizados como α e β. Uma diretriz amplamente utilizada para construção de software de fácil manutenção é manter alta a coesão interna dos módulos e manter baixo o acoplamento entre eles. Com base nisso, os fatores inicialmente recomendados

para α e β são, respectivamente, graus de acoplamento e coesão. As métricas para coesão são definidas em nível de classe e as métricas de acoplamento são definidas para conexão entre duas classes. Os parâmetros α e β, porém, referem-se ao software. Neste contexto, identificam-se as seguintes questões:

- Conforme discutido na Seção 4.1, não há ainda um consenso sobre uma métrica para avaliação de coesão interna de classes. Neste trabalho, propõe-se a métrica Coesão Contratual, apresentada na Seção 5. É necessário avaliar a capacidade dessa métrica na avaliação da coesão interna de classes.

- O estudo realizado com uma outra métrica de coesão interna de classes, LCOM, apresentado na Seção 6 mostrou que os seus valores seguem uma *power law*, o que indica que a média dos valores não é significativa. Essa pode ser também uma característica da métrica Coesão Contratual. É necessário investigar essa hipótese, e caso se confirme, definir uma maneira diferente da média para sumarizar seus valores para nível de sistema.

- Conforme descrito na Seção 3.4, Myers [1975] define uma escala para graus de acoplamento e Ferreira [2006] revisou esta escala para a orientação por objetos. Entretanto, a automatização da identificação destes tipos de acoplamento é difícil. Por exemplo, é difícil diferenciar quando o uso de um atributo público ocasiona acoplamento por conteúdo, por dado comum ou por elemento externo. Na implementação da ferramenta *Connecta* utilizou-se uma simplificação desta escala: acoplamento por campo, quando uma classe utiliza diretamente atributo público da outra, por método, quando uma classe utiliza apenas métodos de outra, e por herança, quando uma classe é subclasse de outra. A esses acoplamentos foram atribuídos os valores 0,3, 0,1 e 0,4, respectivamente. Esses valores foram definidos de forma intuitiva, considerando-se os seus efeitos e a respectiva correspondência com a escala de Myers revisada por Ferreira.

Este trabalho se propõe a estudar um modelo de predição de tempo de manutenção de software orientado por objetos. Desta forma, os experimentos serão conduzidos utilizando-se softwares construídos neste paradigma. Porém, o modelo foi definido de forma genérica, podendo ser aplicado a outros paradigmas.

Capítulo 8

Planejamento de Atividades

Este capítulo apresenta o planejamento das atividades a serem realizadas na elaboração desta tese. A seguintes atividades já foram realizadas:

1. definição formal do problema a ser solucionado pelo modelo;
2. definição matemática do modelo;
3. implementação do modelo no software Matlab;
4. análise preliminar dos polinômios obtidos e definição preliminar de KB3;
5. implementação da métrica KB3 preliminar e experimentos;
6. análise geral dos polinômios obtidos e definição de KB3;
7. implementação de KB3;
8. definição e implementação da métrica Coesão Contratual;
9. identificação de valores referência de métricas de software OO: foram analisadas as seguintes métricas: conexões aferentes, LCOM, COF, DIT, número de atributos públicos e número de métodos públicos. Foram identifica as distribuições de probabilidades pelas quais os valores dessas métricas são modelados e, a partir disso e da análise dos dados obtidos, foram identificados valores que possam ser tomados com uma referência inicial.

As seguintes atividades ainda devem ser realizadas para atingir os objetivos traçados nesta tese:

1. Avaliação da métrica Coesão Contratual: a métrica Coesão Contratual será avaliada a partir de um estudo experimental que consistirá na coleta das medidas dessa métrica e de LCOM, outra métrica amplamente conhecida para esse mesmo aspecto. Os resultados de ambas serão comparados à análise sobre a qualidade estrutural das classes utilizadas no experimento.

2. Identificação de valores referência de métricas de software OO: outras métricas serão avaliadas dentro da mesma metodologia adotada para avaliação descrita no Capítulo 6. Dentre essas métricas estão: grau de acoplamento entre módulos e Coesão Contratual.

3. Avaliação de KB3: o modelo proposto será avaliado por meio de estudos de caso com um conjunto de softwares. A dificuldade deste tipo de abordagem é a dependência de disponibilidade de dados históricos de softwares reais. Em um primeiro momento, planeja-se a realização de três tipos de experimentos:

   Experimento 1 - software real de pequeno porte: será utilizado um software de pequeno porte, que possui 64 classes. O software apresenta alguns problemas estruturais, por exemplo, algumas classes implementam mais de um papél, como interface com usuário e implementação de regras de negócio. Melhorias na estrutura do software serão realizadas gradualmente e, a cada melhoria, os valores de KB3 e tempo utilizado para realização da melhoria serão coletados. Duas hipóteses serão investigadas neste experimento: a primeira é que os valores de KB3 diminuem à medida que as melhorias são realizadas; a segunda é que KB3 é um indicador do tempo necessário para realizar as manutenções de melhoria. Dois desenvolvedores participarão deste experimento e o software a ser utilizado é de conhecimento de ambos.

   Experimento 2 - software de alunos: um conjunto de software produzidos por alunos iniciantes em programação orientada por objetos serão utilizados neste experimento. Esse tipo de software é propício a essa avaliação porque, como não possuem formação em técnicas de produção de software de qualidade, alunos com essa característica tendem, por exemplo, a implemetar classes com problemas de coesão. A hipótese a ser investigada neste estudo é que a intervenção nos aspectos coesão interna, acoplamento entre módulos e conectividade gera impacto no valor de KB3.

   Experimento 3 - software real: o ideal é que KB3 fosse avaliado também em softwares produzidos por instituições particulares. Entretanto, realizar esse

tipo de estudo é relativamente difícil, por duas razões principais: dificilmente uma organização possui um processo de manutenção estabelecido e, mais dificilmente ainda, mantém dados históricos sobre as manutenções realizadas; além disso, software é um produto de grande valor para organizações, tanto para quem o utiliza quanto para quem o produz, o que gera grandes restrições em fornecê-los para estudos. Alternativamente a esse tipo de experimento, há a possibilidade de realizar o estudo com softwares abertos. A hipótese a ser investigada é que softwares com valores maiores de KB3 têm também maior tempo médio de manutenção.

Para a realização destes experimentos, as seguintes atividades devem ser realizadas:

- planejamento dos experimentos;
- identificação dos softwares a serem utilizados para avaliação;
- coleta dos dados;
- análise dos resultados.

Os seguintes artigos foram produzidos neste trabalho:

1. *Reestruturação de Software Dirigida por Conectividade para Redução de Custo de Manutenção* : artigo já publicado em 2008 no periódico nacional RITA (Revista de Informática Teórica e Aplicada, Volume 15, Número 2, páginas 155 a 179).
2. *Valores Referência para Métricas de Software OO*: um estudo que visa identificar os valores a serem tomados como referência para as principais métricas de software OO. Este artigo foi aceito para publicação no Simpósio Brasileiro de Engenharia de Software (SBES) de 2009.

Os seguintes artigos serão ainda elaborados com resultados futuros deste trabalho:

1. *Métrica de Coesão Contratual de Software Orientado por Objetos*: apresentação e avaliação da métrica de avaliação de coesão de sotware OO proposta nesta tese.
2. *KB3 - Um Modelo de Predição de Tempo de Manutenção de Software Orientado por Objetos*: artigo com descrição do modelo proposto nesta tese.
3. *Validação do Modelo de Predição de Tempo de Manutenção de Software Orientado por Objetos*: apresentação de estudos de caso realizados com o modelo proposto.

A Figura 8.1 apresenta o sumário proposto para o texto final da tese. As seções a serem ainda elaboradas estão em destaque.

Figura 8.1. Sumário da Tese

A Figura 8.2 apresenta o cronograma de atividades a serem realizadas.

Atividade	2009				
	Ago	Set	Out	Nov	Dez
Avaliação de Coesão Contratual					
Seções 5.4, 5.5., 5.6					
Artigo 3					
Planejamento Experimento 1					
Experimento 1					

Atividade	2010											
	Jan	Fev	Mar	Abr	Mai	Jun	Jul	Ago	Set	Out	Nov	Dez
Seção 8.1												
Artigo 4												
Planejamento Experimento 2												
Experimento 2												
Seção 8.2												
Planejamento Experimento 3												
Experimento 3												
Seção 8.3												
Artigo 5												

Atividade	2011		
	Jan	Fev	Mar
Conclusão e revisão/correção de texto			
Defesa de tese			

Figura 8.2. Cronograma de Atividades

Referências Bibliográficas

Abreu, F. B. & Carapuça, R. (1994). Object-oriented software engineering: Measuring and controlling the development process. In *Proceedings of 4th Int. Conf. of Software Quality*, McLean, VA, USA.

Abreu, F. B.; Ochoa, L. & Goulo, M. (1995). The goodly design language for mood metrics collection. In *ISEG/INESC ECOOP Workshop*, Portugal.

Ash, D.; Alderete, J.; Oman, P. W. & Lowther, B. (2006). Using software maintainability models to track code health. 15:123 – 149.

Baxter, G.; Frean, M.; Noble, J.; Rickerby, M.; Smith, H.; Visser, M.; Melton, H. & Tempero, E. (2006). Undertanding the shape of java software. In *OOPSLA'06*, Oregon, Portland, USA.

Chhabra, J. K. & Aggarwal, K. K. (2006). Measurement of intra-class and inter-class weakness for object-oriented software. In *Third International Conference on Information Technology: New Generations (ITNG'06)*, pp. 155–160.

Chidamber, S. R. & Kemerer, C. F. (1994). A metrics suite for object oriented design. pp. 476–493.

Counsell, S.; Swift, S. & Crampton, J. (2004). The interpretation and utility of three cohesion metrics for object-oriented design. 30(11):826–832.

Deissenboek, F.; Wagner, S. & Pizka, M. (2007). An activity-based quality model for maintainability. In *ICSM 2007*, pp. 184–193.

DependencyFinder (2007). *Dependency Finder*. Disponível em http://depfind.sourceforge.net/. Acesso em Abril de 2007.

Fenton, N. & Neill, M. (2000). Software metrics: Roadmap. In *Proceedings of the Conference on the Future of Software Engineering*.

Ferreira, K. A. M. (2006). *Avaliação de Conectividade em Sistemas Orientados por Objetos*. Belo Horizonte.

Ferreira, K. A. M.; Bigonha, M. A. S. & Bigonha, R. S. (2008). Reestruturação de software dirigida por conectividade para redução de custo de manutenção. *Revista de Informática Teórica e Aplicada*, 15(2):155–179.

Filho, W. P. P. (2001). *Engenharia de Software - Fundamentos, Métodos e Padrões*. LTC, Rio de Janeiro.

Fowler, M. (1999). *Refactoring - Improving the Design of Existing Code*. Addison Wesley.

Gilb, T. (1977). *Software Metrics*. Winthrop Publishers, Estados Unidos.

Grinsteas, C. M. & Snell., J. L. (1991). *Introduction to Probability*. America Mathematical Society.

Griswold, W. G.; Shonle, M.; Sullivan, K.; Song, Y.; ant Yuanfang Cai, N. T. & Rajan, H. (2006). Modular software design with crosscuting interfaces.

Grosser, D.; Sahraoui, H. A. & Valtchev, P. (2003). An analogy-based approach for predicting design stability of java classes. In *Ninth International Software Metrics Symposium*, pp. 252–262.

Gyimothy, T.; Ferenc, R. & Siket, I. (2005a). Empirical validation of object-oriented metrics on open source software for fault prediction. *IEEE Transactions on Software Engineering*, 31(10):897–910.

Gyimothy, T.; Ferenc, R. & Siket, I. (2005b). Empirical validation of object-oriented metrics on open source software for fault prediction. *IEEE Transactions on Software Engineering*, 31:897–910.

Heitlager, I.; Kuipers, T. & Visser, J. (2007). A practical model for measuring maintainability - a preliminary report. *IEEE*.

Hordijk, W. & Wieringa, R. (2005). Surveying the factors that influence maintainability: research design. In *Proceedings of the 10th European software engineering conference held jointly with 13th ACM SIGSOFT international symposium on Foundations of software engineering ESEC/FSE-13*, volume 30, pp. 385–388.

Humphrey, W. S. (1995). *A Discipline for Software Engineering*. Addison Wesley, 6 edição.

JDepend (2007). *JDepend*. Disponível em http://www.clarkware.com/software/JDepend.html. Acesso em Abril de 2007.

Kabaili, H.; Keller, R. K. & Lustman, F. (2001). Cohesion as changeability indicator in object-oriented systems. In *IEEE*, pp. 39–46.

Kan, S. (2003). *Metrics and Models in Software Quality Engineering*. Addison-Wesley, 2 edição.

Kiczales, G. & Mezini, M. (2005). Aspect-oriented programming and modular reasoning. In *ICSE'05*, St. Louis, Missouri, USA.

Krakatau (2006). *Krakatau Essencial Metrics*. Disponível em http://www.powersoftware.com/. Acesso em Maio de 2006.

Kulesza, U.; Sant'Anna, C.; Garcia, A.; Coelho, R.; Staa, A. v. & Lucena, C. (2006). Quantifying the effects of aspect-oriented programming: A maintenance study. In *ICSM06 - Proceedings of 9th International Conference on Software Engineering*.

Li, P. L.; Herbsleb, J.; Shaw, M. & Robinson, B. (2006). Experiences and results from initiating field defect prediction and product test prioritization efforts at abb inc. In *28th international conference on Software engineering ICSE '06*.

Li, W. & Henry, S. (1993). Maintenance metrics for object oriented paradigm. In *IEEE*.

Louridas, P.; Spinellis, D. & Vlachos, V. (2008). Power laws in software. 18(1).

Marcus, A. & Poshyvanyk, D. (2005). The conceptual cohesion of classes. In *Proceedings of the 21st IEEE International Conference on Software Maintenance*.

Martin, R. (1994). Oo design quality metrics - an analysis of dependencies.

Metrics (2007). *Metrics*. Disponível em http://www.metrics.sourceforge.net. Acesso em Outubro de 2007.

Meyer, B. (1997). *Object-oriented software construction*. Prentice Hall International Series, Estados Unidos, 2 edição.

Mäkelä, S. & Leppänen, V. (2007). Cliente bases object-oriented cohesion metrics. In *IEEE 31st Annual International Computer Software and Applications Conference*.

Munro, M. (2005). Product metrics for automatic identification of "bad smell"design problems in java source-code. In *11th IEEE International Symposium Software Metrics*.

Myers, G. J. (1975). *Reliable software through composite design*. Petrocelli/Charter,, Nova York, 2 edição.

Nagappan, N.; Ball, T. & Zeller, A. (2006). Mining metrics to predict component failures. In *ICSE'06*, pp. 452–461.

Newman, M. E. J. (2003). The structure and function of complex networks. In *SIAM Reviews*, volume 45, pp. 167–256.

ObjectDetail (2006). *Object Detail*. Disponível em http://www.obsoft.com/Product/DetailPaper.html. Acesso em Maio de 2006.

Pearse, T. & Oman, P. (19945). Maintainability measurements on industrial source code maintenance activities. 29:295–303.

Petrov, V. & Mordecki., E. (2003). *Teoría de Probabilidades*. Matematnka, Editorial URSS, Moscou.

Pfleeger, S. L. (1998). *Software engineering theory and practice*. Prentice-Hall, Upper Saddle River.

Pizka, M. (2004). Straightening spaghetti-code with refactoring? In *In Proceedings of the Int. Conf. on Software Engineering Research and Practice - SERP*.

Potantin, A.; Noble, J.; Frean, M. & Biddle, R. (2005). Scale-free geometry in oo programas. 48(5):99–103.

Pressman, R. S. (2006). *Engenharia de Software*. MacGraw Hill, Rio de Janeiro, 6 edição.

Puppin, D. & Silvestrini, F. (2006). The social network of java classes. In *SAC'06*, pp. 1409–1413, Dijon, França.

R, W. & Counsell, S. (2003). Power law distributions in class relationships. In *Proceedings of 3rd International Workshop on Source Code Analysis and Manipulation (SCAM)*.

Samoladas, I.; Stamelos, I.; Angelis, L. & Oikonomou, A. (20054). Open source software development should strive for even greater code maintainability. 47(10):83–87.

Santa'Anna, C. N.; Garcia, A. F.; Chaves, C. V. F. G.; Lucena, C. J. P. & Staa, A. V. (2003). On the reuse and maintenance of aspect-oriented software: an assessment framework. In *17º SBES*, Porto Alegre.

Sarwar, M. I.; WasifTanveer; Sarwar, I. & Mahmood, W. (2008). A comparative study of mi tools: Defining the roadmap to mi tools standardization. *IEEE*.

Schröter, A.; Zimmermann, T. & Zeller, A. (2006). Faults and failures: Predicting component failures at design time. In *ACM/IEEE International Symposium on Empirical Software Engineering ISESE '06*.

SEI, S. E. I. (2009). *Software Tecnology Roadmap*. Disponível em http://www.sei.cmu.edu/str/7/28/2008. Acesso em Abril de 2009.

Soares, J. F.; Farias, A. A. & Cesar, C. C. (1991). *Introdução à Estatística*. Guanabara, Rio de Janeiro.

Sommerville, I. (2003). *Engenharia de Software*. Addison Wesley, São Paulo, 6 edição.

Subrammanyam, R. & Krishnan, M. S. (2003). Empirical analysis of ck metrics for object-oriented design complexity: Implifications for software defects. 29(4):297–310.

Tempero, E. (2008). On measuring java software. In *ACSC2008 - Conferences in Research and Practice in Information Technology (CRPIT)*, volume 74, Wollongong, Australia.

Tsantalis, N.; Chatzigeorgiou, A. & Stephanides, G. (2005). Predicting the probability of change in object-oriented systems. In *IEEE Transactions on Software Engineering*, volume 31, pp. 601–614.

Understand (2007). *Understand for Java*. Disponível em http://www.scitools.com/uj.html. Acesso em Abril de 2007.

von Staa, A. (2000). *Programação Modular - Desenvolvendo programas complexos de forma organizada e segura*. Campus, Rio de Janeiro.

Wand, M. (2003). Understanding aspects (extended abstract). In *ICFP'03*.

Weka (2009). *Weka 3: Data Mining Software in Java*. Disponível em http://www.cs.waikato.ac.nz/ml/weka/index.html. Acesso em Julho de 2009.

Weyuker, E. (1988). Evaluating software complexity measures. *IEEE Transactions on Software Engineering*, 14:1357–1365.

Wikipedia (2009). *Factorial*. Disponível em http://pt.wikipedia.org/wiki/Factorial. Acesso em Julho de 2009.

Xenos, M.; Stavrinoudis, D.; Zikouli, K. & Christodoulakis, D. (2000). Object-oriented metrics - a survey. In *FESMA 2000*, Madrid, Spain.

Zhou, Y. & Leung, H. (2006). Empirical analysis of object-oriented design metrics for predicting high and low severity faults. *IEEE Transactions on Software Engineering*, 32(10):771–789.

Apêndice A

Polinômios Obtidos para Predição de Tempo de Manutenção

A.1 Resultados Preliminares

4 módulos

$$\begin{bmatrix} 1 + 6\,\frac{a}{b} + 12\,\frac{a^2}{b^2} + 12\,\frac{a^3}{b^3} \\ 2 + 10\,\frac{a}{b} + 16\,\frac{a^2}{b^2} + 12\,\frac{a^3}{b^3} \\ 3 + 12\,\frac{a}{b} + 16\,\frac{a^2}{b^2} + 12\,\frac{a^3}{b^3} \\ 4 + 12\,\frac{a}{b} + 16\,\frac{a^2}{b^2} + 12\,\frac{a^3}{b^3} \end{bmatrix}$$

5 módulos

$$\begin{bmatrix} 1 + 8\,\frac{a}{b} + 24\,\frac{a^2}{b^2} + 48\,\frac{a^3}{b^3} + 48\,\frac{a^4}{b^4} \\ 2 + 14\,\frac{a}{b} + 36\,\frac{a^2}{b^2} + 60\,\frac{a^3}{b^3} + 48\,\frac{a^4}{b^4} \\ 3 + 18\,\frac{a}{b} + 40\,\frac{a^2}{b^2} + 60\,\frac{a^3}{b^3} + 48\,\frac{a^4}{b^4} \\ 4 + 20\,\frac{a}{b} + 40\,\frac{a^2}{b^2} + 60\,\frac{a^3}{b^3} + 48\,\frac{a^4}{b^4} \\ 5 + 20\,\frac{a}{b} + 40\,\frac{a^2}{b^2} + 60\,\frac{a^3}{b^3} + 48\,\frac{a^4}{b^4} \end{bmatrix}$$

6 módulos

$$\left[\begin{array}{l} 1+10\,\frac{a}{b}+40\,\frac{a^2}{b^2}+120\,\frac{a^3}{b^3}+240\,\frac{a^4}{b^4}+240\,\frac{a^5}{b^5} \\ 2+18\,\frac{a}{b}+64\,\frac{a^2}{b^2}+168\,\frac{a^3}{b^3}+288\,\frac{a^4}{b^4}+240\,\frac{a^5}{b^5} \\ 3+24\,\frac{a}{b}+76\,\frac{a^2}{b^2}+180\,\frac{a^3}{b^3}+288\,\frac{a^4}{b^4}+240\,\frac{a^5}{b^5} \\ 4+28\,\frac{a}{b}+80\,\frac{a^2}{b^2}+180\,\frac{a^3}{b^3}+288\,\frac{a^4}{b^4}+240\,\frac{a^5}{b^5} \\ 5+30\,\frac{a}{b}+80\,\frac{a^2}{b^2}+180\,\frac{a^3}{b^3}+288\,\frac{a^4}{b^4}+240\,\frac{a^5}{b^5} \\ 6+30\,\frac{a}{b}+80\,\frac{a^2}{b^2}+180\,\frac{a^3}{b^3}+288\,\frac{a^4}{b^4}+240\,\frac{a^5}{b^5} \end{array}\right]$$

7 módulos

$$\left[\begin{array}{l} 1+12\,\frac{a}{b}+60\,\frac{a^2}{b^2}+240\,\frac{a^3}{b^3}+720\,\frac{a^4}{b^4}+1440\,\frac{a^5}{b^5}+1440\,\frac{a^6}{b^6} \\ 2+22\,\frac{a}{b}+100\,\frac{a^2}{b^2}+360\,\frac{a^3}{b^3}+960\,\frac{a^4}{b^4}+1680\,\frac{a^5}{b^5}+1440\,\frac{a^6}{b^6} \\ 3+30\,\frac{a}{b}+124\,\frac{a^2}{b^2}+408\,\frac{a^3}{b^3}+1008\,\frac{a^4}{b^4}+1680\,\frac{a^5}{b^5}+1440\,\frac{a^6}{b^6} \\ 4+36\,\frac{a}{b}+136\,\frac{a^2}{b^2}+420\,\frac{a^3}{b^3}+1008\,\frac{a^4}{b^4}+1680\,\frac{a^5}{b^5}+1440\,\frac{a^6}{b^6} \\ 5+40\,\frac{a}{b}+140\,\frac{a^2}{b^2}+420\,\frac{a^3}{b^3}+1008\,\frac{a^4}{b^4}+1680\,\frac{a^5}{b^5}+1440\,\frac{a^6}{b^6} \\ 6+42\,\frac{a}{b}+140\,\frac{a^2}{b^2}+420\,\frac{a^3}{b^3}+1008\,\frac{a^4}{b^4}+1680\,\frac{a^5}{b^5}+1440\,\frac{a^6}{b^6} \\ 7+42\,\frac{a}{b}+140\,\frac{a^2}{b^2}+420\,\frac{a^3}{b^3}+1008\,\frac{a^4}{b^4}+1680\,\frac{a^5}{b^5}+1440\,\frac{a^6}{b^6} \end{array}\right]$$

8 módulos

$$\left[\begin{array}{l} 1+14\,\frac{a}{b}+84\,\frac{a^2}{b^2}+420\,\frac{a^3}{b^3}+1680\,\frac{a^4}{b^4}+5040\,\frac{a^5}{b^5}+10080\,\frac{a^6}{b^6}+10080\,\frac{a^7}{b^7} \\ 2+26\,\frac{a}{b}+144\,\frac{a^2}{b^2}+660\,\frac{a^3}{b^3}+2400\,\frac{a^4}{b^4}+6480\,\frac{a^5}{b^5}+11520\,\frac{a^6}{b^6}+10080\,\frac{a^7}{b^7} \\ 3+36\,\frac{a}{b}+184\,\frac{a^2}{b^2}+780\,\frac{a^3}{b^3}+2640\,\frac{a^4}{b^4}+6720\,\frac{a^5}{b^5}+11520\,\frac{a^6}{b^6}+10080\,\frac{a^7}{b^7} \\ 4+44\,\frac{a}{b}+208\,\frac{a^2}{b^2}+828\,\frac{a^3}{b^3}+2688\,\frac{a^4}{b^4}+6720\,\frac{a^5}{b^5}+11520\,\frac{a^6}{b^6}+10080\,\frac{a^7}{b^7} \\ 5+50\,\frac{a}{b}+220\,\frac{a^2}{b^2}+840\,\frac{a^3}{b^3}+2688\,\frac{a^4}{b^4}+6720\,\frac{a^5}{b^5}+11520\,\frac{a^6}{b^6}+10080\,\frac{a^7}{b^7} \\ 6+54\,\frac{a}{b}+224\,\frac{a^2}{b^2}+840\,\frac{a^3}{b^3}+2688\,\frac{a^4}{b^4}+6720\,\frac{a^5}{b^5}+11520\,\frac{a^6}{b^6}+10080\,\frac{a^7}{b^7} \\ 7+56\,\frac{a}{b}+224\,\frac{a^2}{b^2}+840\,\frac{a^3}{b^3}+2688\,\frac{a^4}{b^4}+6720\,\frac{a^5}{b^5}+11520\,\frac{a^6}{b^6}+10080\,\frac{a^7}{b^7} \\ 8+56\,\frac{a}{b}+224\,\frac{a^2}{b^2}+840\,\frac{a^3}{b^3}+2688\,\frac{a^4}{b^4}+6720\,\frac{a^5}{b^5}+11520\,\frac{a^6}{b^6}+10080\,\frac{a^7}{b^7} \end{array}\right]$$

9 módulos

$$\left[\begin{array}{l}
1+16\,\frac{a}{b}+112\,\frac{a^2}{b^2}+672\,\frac{a^3}{b^3}+3360\,\frac{a^4}{b^4}+13440\,\frac{a^5}{b^5}+40320\,\frac{a^6}{b^6}+80640\,\frac{a^7}{b^7}+80640\,\frac{a^8}{b^8}\\
2+30\,\frac{a}{b}+196\,\frac{a^2}{b^2}+1092\,\frac{a^3}{b^3}+5040\,\frac{a^4}{b^4}+18480\,\frac{a^5}{b^5}+50400\,\frac{a^6}{b^6}+90720\,\frac{a^7}{b^7}+80640\,\frac{a^8}{b^8}\\
3+42\,\frac{a}{b}+256\,\frac{a^2}{b^2}+1332\,\frac{a^3}{b^3}+5760\,\frac{a^4}{b^4}+19920\,\frac{a^5}{b^5}+51840\,\frac{a^6}{b^6}+90720\,\frac{a^7}{b^7}+80640\,\frac{a^8}{b^8}\\
4+52\,\frac{a}{b}+296\,\frac{a^2}{b^2}+1452\,\frac{a^3}{b^3}+6000\,\frac{a^4}{b^4}+20160\,\frac{a^5}{b^5}+51840\,\frac{a^6}{b^6}+90720\,\frac{a^7}{b^7}+80640\,\frac{a^8}{b^8}\\
5+60\,\frac{a}{b}+320\,\frac{a^2}{b^2}+1500\,\frac{a^3}{b^3}+6048\,\frac{a^4}{b^4}+20160\,\frac{a^5}{b^5}+51840\,\frac{a^6}{b^6}+90720\,\frac{a^7}{b^7}+80640\,\frac{a^8}{b^8}\\
6+66\,\frac{a}{b}+332\,\frac{a^2}{b^2}+1512\,\frac{a^3}{b^3}+6048\,\frac{a^4}{b^4}+20160\,\frac{a^5}{b^5}+51840\,\frac{a^6}{b^6}+90720\,\frac{a^7}{b^7}+80640\,\frac{a^8}{b^8}\\
7+70\,\frac{a}{b}+336\,\frac{a^2}{b^2}+1512\,\frac{a^3}{b^3}+6048\,\frac{a^4}{b^4}+20160\,\frac{a^5}{b^5}+51840\,\frac{a^6}{b^6}+90720\,\frac{a^7}{b^7}+80640\,\frac{a^8}{b^8}\\
8+72\,\frac{a}{b}+336\,\frac{a^2}{b^2}+1512\,\frac{a^3}{b^3}+6048\,\frac{a^4}{b^4}+20160\,\frac{a^5}{b^5}+51840\,\frac{a^6}{b^6}+90720\,\frac{a^7}{b^7}+80640\,\frac{a^8}{b^8}\\
9+72\,\frac{a}{b}+336\,\frac{a^2}{b^2}+1512\,\frac{a^3}{b^3}+6048\,\frac{a^4}{b^4}+20160\,\frac{a^5}{b^5}+51840\,\frac{a^6}{b^6}+90720\,\frac{a^7}{b^7}+80640\,\frac{a^8}{b^8}
\end{array}\right]$$

10 módulos

$1+18\,\frac{a}{b}+144\,\frac{a^2}{b^2}+1008\,\frac{a^3}{b^3}+6048\,\frac{a^4}{b^4}+30240\,\frac{a^5}{b^5}+120960\,\frac{a^6}{b^6}+362880\,\frac{a^7}{b^7}+725760\,\frac{a^8}{b^8}+725760\,\frac{a^9}{b^9}$

$2+34\,\frac{a}{b}+256\,\frac{a^2}{b^2}+1680\,\frac{a^3}{b^3}+9408\,\frac{a^4}{b^4}+43680\,\frac{a^5}{b^5}+161280\,\frac{a^6}{b^6}+443520\,\frac{a^7}{b^7}+806400\,\frac{a^8}{b^8}+725760\,\frac{a^9}{b^9}$

$3+48\,\frac{a}{b}+340\,\frac{a^2}{b^2}+2100\,\frac{a^3}{b^3}+11088\,\frac{a^4}{b^4}+48720\,\frac{a^5}{b^5}+171360\,\frac{a^6}{b^6}+453600\,\frac{a^7}{b^7}+806400\,\frac{a^8}{b^8}+725760\,\frac{a^9}{b^9}$

$4+60\,\frac{a}{b}+400\,\frac{a^2}{b^2}+2340\,\frac{a^3}{b^3}+11808\,\frac{a^4}{b^4}+50160\,\frac{a^5}{b^5}+172800\,\frac{a^6}{b^6}+453600\,\frac{a^7}{b^7}+806400\,\frac{a^8}{b^8}+725760\,\frac{a^9}{b^9}$

$5+70\,\frac{a}{b}+440\,\frac{a^2}{b^2}+2460\,\frac{a^3}{b^3}+12048\,\frac{a^4}{b^4}+50400\,\frac{a^5}{b^5}+172800\,\frac{a^6}{b^6}+453600\,\frac{a^7}{b^7}+806400\,\frac{a^8}{b^8}+725760\,\frac{a^9}{b^9}$

$6+78\,\frac{a}{b}+464\,\frac{a^2}{b^2}+2508\,\frac{a^3}{b^3}+12096\,\frac{a^4}{b^4}+50400\,\frac{a^5}{b^5}+172800\,\frac{a^6}{b^6}+453600\,\frac{a^7}{b^7}+806400\,\frac{a^8}{b^8}+725760\,\frac{a^9}{b^9}$

$7+84\,\frac{a}{b}+476\,\frac{a^2}{b^2}+2520\,\frac{a^3}{b^3}+12096\,\frac{a^4}{b^4}+50400\,\frac{a^5}{b^5}+172800\,\frac{a^6}{b^6}+453600\,\frac{a^7}{b^7}+806400\,\frac{a^8}{b^8}+725760\,\frac{a^9}{b^9}$

$8+88\,\frac{a}{b}+480\,\frac{a^2}{b^2}+2520\,\frac{a^3}{b^3}+12096\,\frac{a^4}{b^4}+50400\,\frac{a^5}{b^5}+172800\,\frac{a^6}{b^6}+453600\,\frac{a^7}{b^7}+806400\,\frac{a^8}{b^8}+725760\,\frac{a^9}{b^9}$

$9+90\,\frac{a}{b}+480\,\frac{a^2}{b^2}+2520\,\frac{a^3}{b^3}+12096\,\frac{a^4}{b^4}+50400\,\frac{a^5}{b^5}+172800\,\frac{a^6}{b^6}+453600\,\frac{a^7}{b^7}+806400\,\frac{a^8}{b^8}+725760\,\frac{a^9}{b^9}$

$10+90\,\frac{a}{b}+480\,\frac{a^2}{b^2}+2520\,\frac{a^3}{b^3}+12096\,\frac{a^4}{b^4}+50400\,\frac{a^5}{b^5}+172800\,\frac{a^6}{b^6}+453600\,\frac{a^7}{b^7}+806400\,\frac{a^8}{b^8}+725760\,\frac{a^9}{b^9}$

11 módulos

$1 + 20\,\frac{a}{b} + 180\,\frac{a^2}{b^2} + 1440\,\frac{a^3}{b^3} + 10080\,\frac{a^4}{b^4} + 60480\,\frac{a^5}{b^5} + 302400\,\frac{a^6}{b^6} + 1209600\,\frac{a^7}{b^7} + 3628800\,\frac{a^8}{b^8} + 7257600\,\frac{a^9}{b^9} + 7257600\,\frac{a^{10}}{b^{10}}$

$1 + 20\,\frac{a}{b} + 180\,\frac{a^2}{b^2} + 1440\,\frac{a^3}{b^3} + 10080\,\frac{a^4}{b^4} + 60480\,\frac{a^5}{b^5} + 302400\,\frac{a^6}{b^6} + 1209600\,\frac{a^7}{b^7} + 3628800\,\frac{a^8}{b^8} + 7257600\,\frac{a^9}{b^9} + 7257600\,\frac{a^{10}}{b^{10}}$

$2 + 38\,\frac{a}{b} + 324\,\frac{a^2}{b^2} + 2448\,\frac{a^3}{b^3} + 16128\,\frac{a^4}{b^4} + 90720\,\frac{a^5}{b^5} + 423360\,\frac{a^6}{b^6} + 1572480\,\frac{a^7}{b^7} + 4354560\,\frac{a^8}{b^8} + 7983360\,\frac{a^9}{b^9} + 7257600\,\frac{a^{10}}{b^{10}}$

$3 + 54\,\frac{a}{b} + 436\,\frac{a^2}{b^2} + 3120\,\frac{a^3}{b^3} + 19488\,\frac{a^4}{b^4} + 104160\,\frac{a^5}{b^5} + 463680\,\frac{a^6}{b^6} + 1653120\,\frac{a^7}{b^7} + 4435200\,\frac{a^8}{b^8} + 7983360\,\frac{a^9}{b^9} + 7257600\,\frac{a^{10}}{b^{10}}$

$4 + 68\,\frac{a}{b} + 520\,\frac{a^2}{b^2} + 3540\,\frac{a^3}{b^3} + 21168\,\frac{a^4}{b^4} + 109200\,\frac{a^5}{b^5} + 473760\,\frac{a^6}{b^6} + 1663200\,\frac{a^7}{b^7} + 4435200\,\frac{a^8}{b^8} + 7983360\,\frac{a^9}{b^9} + 7257600\,\frac{a^{10}}{b^{10}}$

$5 + 80\,\frac{a}{b} + 580\,\frac{a^2}{b^2} + 3780\,\frac{a^3}{b^3} + 21888\,\frac{a^4}{b^4} + 110640\,\frac{a^5}{b^5} + 475200\,\frac{a^6}{b^6} + 1663200\,\frac{a^7}{b^7} + 4435200\,\frac{a^8}{b^8} + 7983360\,\frac{a^9}{b^9} + 7257600\,\frac{a^{10}}{b^{10}}$

$6 + 90\,\frac{a}{b} + 620\,\frac{a^2}{b^2} + 3900\,\frac{a^3}{b^3} + 22128\,\frac{a^4}{b^4} + 110880\,\frac{a^5}{b^5} + 475200\,\frac{a^6}{b^6} + 1663200\,\frac{a^7}{b^7} + 4435200\,\frac{a^8}{b^8} + 7983360\,\frac{a^9}{b^9} + 7257600\,\frac{a^{10}}{b^{10}}$

$7 + 98\,\frac{a}{b} + 644\,\frac{a^2}{b^2} + 3948\,\frac{a^3}{b^3} + 22176\,\frac{a^4}{b^4} + 110880\,\frac{a^5}{b^5} + 475200\,\frac{a^6}{b^6} + 1663200\,\frac{a^7}{b^7} + 4435200\,\frac{a^8}{b^8} + 7983360\,\frac{a^9}{b^9} + 7257600\,\frac{a^{10}}{b^{10}}$

$8 + 104\,\frac{a}{b} + 656\,\frac{a^2}{b^2} + 3960\,\frac{a^3}{b^3} + 22176\,\frac{a^4}{b^4} + 110880\,\frac{a^5}{b^5} + 475200\,\frac{a^6}{b^6} + 1663200\,\frac{a^7}{b^7} + 4435200\,\frac{a^8}{b^8} + 7983360\,\frac{a^9}{b^9} + 7257600\,\frac{a^{10}}{b^{10}}$

$9 + 108\,\frac{a}{b} + 660\,\frac{a^2}{b^2} + 3960\,\frac{a^3}{b^3} + 22176\,\frac{a^4}{b^4} + 110880\,\frac{a^5}{b^5} + 475200\,\frac{a^6}{b^6} + 1663200\,\frac{a^7}{b^7} + 4435200\,\frac{a^8}{b^8} + 7983360\,\frac{a^9}{b^9} + 7257600\,\frac{a^{10}}{b^{10}}$

$10 + 110\,\frac{a}{b} + 660\,\frac{a^2}{b^2} + 3960\,\frac{a^3}{b^3} + 22176\,\frac{a^4}{b^4} + 110880\,\frac{a^5}{b^5} + 475200\,\frac{a^6}{b^6} + 1663200\,\frac{a^7}{b^7} + 4435200\,\frac{a^8}{b^8} + 7983360\,\frac{a^9}{b^9} + 7257600\,\frac{a^{10}}{b^{10}}$

$11 + 110\,\frac{a}{b} + 660\,\frac{a^2}{b^2} + 3960\,\frac{a^3}{b^3} + 22176\,\frac{a^4}{b^4} + 110880\,\frac{a^5}{b^5} + 475200\,\frac{a^6}{b^6} + 1663200\,\frac{a^7}{b^7} + 4435200\,\frac{a^8}{b^8} + 7983360\,\frac{a^9}{b^9} + 7257600\,\frac{a^{10}}{b^{10}}$

20 módulos

$1+38\,\frac{a}{b}+684\,\frac{a^2}{b^2}+11628\,\frac{a^3}{b^3}+186048\,\frac{a^4}{b^4}+2790720\,\frac{a^5}{b^5}+39070080\,\frac{a^6}{b^6}+507911040\,\frac{a^7}{b^7}+6094932480\,\frac{a^8}{b^8} + 67044257280\,\frac{a^9}{b^9} + 670442572800\,\frac{a^{10}}{b^{10}} + 6033983155200\,\frac{a^{11}}{b^{11}} + 48271865241600\,\frac{a^{12}}{b^{12}} + 337903056691200\,\frac{a^{13}}{b^{13}} + 2027418340147200\,\frac{a^{14}}{b^{14}} + 10137091700736000\,\frac{a^{15}}{b^{15}} + 40548366802944000\,\frac{a^{16}}{b^{16}} + 121645100408832000\,\frac{a^{17}}{b^{17}} + 243290200817664000\,\frac{a^{18}}{b^{18}} + 243290200817664000\,\frac{a^{19}}{b^{19}}$

$2+74\,\frac{a}{b}+1296\,\frac{a^2}{b^2}+21420\,\frac{a^3}{b^3}+332928\,\frac{a^4}{b^4}+4847040\,\frac{a^5}{b^5}+65802240\,\frac{a^6}{b^6}+828696960\,\frac{a^7}{b^7}+9623577600\,\frac{a^8}{b^8} + 102330708480\,\frac{a^9}{b^9} + 988020633600\,\frac{a^{10}}{b^{10}} + 8574607641600\,\frac{a^{11}}{b^{11}} + 66056236646400\,\frac{a^{12}}{b^{12}} + 444609285120000\,\frac{a^{13}}{b^{13}} + 2560949482291200\,\frac{a^{14}}{b^{14}} + 12271216269312000\,\frac{a^{15}}{b^{15}} + 46950740508672000\,\frac{a^{16}}{b^{16}} + 134449847820288000\,\frac{a^{17}}{b^{17}} +$

$$256094948229120000\,\frac{a^{18}}{b^{18}} + 243290200817664000\,\frac{a^{19}}{b^{19}}$$

$$3 + 108\,\frac{a}{b} + 1840\,\frac{a^2}{b^2} + 29580\,\frac{a^3}{b^3} + 447168\,\frac{a^4}{b^4} + 6332160\,\frac{a^5}{b^5} + 83623680\,\frac{a^6}{b^6} + 1024732800\,\frac{a^7}{b^7} + 11583936000\,\frac{a^8}{b^8} + 119973934080\,\frac{a^9}{b^9} + 1129166438400\,\frac{a^{10}}{b^{10}} + 9562628275200\,\frac{a^{11}}{b^{11}} + 71984360448000\,\frac{a^{12}}{b^{12}} + 474249904128000\,\frac{a^{13}}{b^{13}} + 2679511958323200\,\frac{a^{14}}{b^{14}} + 12626903697408000\,\frac{a^{15}}{b^{15}} + 47662115364864000\,\frac{a^{16}}{b^{16}} + 135161222676480000\,\frac{a^{17}}{b^{17}} + 256094948229120000\,\frac{a^{18}}{b^{18}} + 243290200817664000\,\frac{a^{19}}{b^{19}}$$

$$4 + 140\,\frac{a}{b} + 2320\,\frac{a^2}{b^2} + 36300\,\frac{a^3}{b^3} + 534528\,\frac{a^4}{b^4} + 7380480\,\frac{a^5}{b^5} + 95155200\,\frac{a^6}{b^6} + 1140048000\,\frac{a^7}{b^7} + 12621772800\,\frac{a^8}{b^8} + 128276628480\,\frac{a^9}{b^9} + 1187285299200\,\frac{a^{10}}{b^{10}} + 9911341440000\,\frac{a^{11}}{b^{11}} + 73727926272000\,\frac{a^{12}}{b^{12}} + 481224167424000\,\frac{a^{13}}{b^{13}} + 2700434748211200\,\frac{a^{14}}{b^{14}} + 12668749277184000\,\frac{a^{15}}{b^{15}} + 47703960944640000\,\frac{a^{16}}{b^{16}} + 135161222676480000\,\frac{a^{17}}{b^{17}} + 256094948229120000\,\frac{a^{18}}{b^{18}} + 243290200817664000\,\frac{a^{19}}{b^{19}}$$

$$5 + 170\,\frac{a}{b} + 2740\,\frac{a^2}{b^2} + 41760\,\frac{a^3}{b^3} + 600048\,\frac{a^4}{b^4} + 8101200\,\frac{a^5}{b^5} + 102362400\,\frac{a^6}{b^6} + 1204912800\,\frac{a^7}{b^7} + 13140691200\,\frac{a^8}{b^8} + 131909057280\,\frac{a^9}{b^9} + 1209079872000\,\frac{a^{10}}{b^{10}} + 10020314304000\,\frac{a^{11}}{b^{11}} + 74163817728000\,\frac{a^{12}}{b^{12}} + 482531841792000\,\frac{a^{13}}{b^{13}} + 2703050096947200\,\frac{a^{14}}{b^{14}} + 12671364625920000\,\frac{a^{15}}{b^{15}} + 47703960944640000\,\frac{a^{16}}{b^{16}} + 135161222676480000\,\frac{a^{17}}{b^{17}} + 256094948229120000\,\frac{a^{18}}{b^{18}} + 243290200817664000\,\frac{a^{19}}{b^{19}}$$

$$6 + 198\,\frac{a}{b} + 3104\,\frac{a^2}{b^2} + 46128\,\frac{a^3}{b^3} + 648096\,\frac{a^4}{b^4} + 8581680\,\frac{a^5}{b^5} + 106686720\,\frac{a^6}{b^6} + 1239507360\,\frac{a^7}{b^7} + 13382853120\,\frac{a^8}{b^8} + 133362028800\,\frac{a^9}{b^9} + 1216344729600\,\frac{a^{10}}{b^{10}} + 10049373734400\,\frac{a^{11}}{b^{11}} + 74250996019200\,\frac{a^{12}}{b^{12}} + 482706198374400\,\frac{a^{13}}{b^{13}} + 2703224453529600\,\frac{a^{14}}{b^{14}} + 12671364625920000\,\frac{a^{15}}{b^{15}} + 47703960944640000\,\frac{a^{16}}{b^{16}} + 135161222676480000\,\frac{a^{17}}{b^{17}} + 256094948229120000\,\frac{a^{18}}{b^{18}} + 243290200817664000\,\frac{a^{19}}{b^{19}}$$

$$7 + 224\,\frac{a}{b} + 3416\,\frac{a^2}{b^2} + 49560\,\frac{a^3}{b^3} + 682416\,\frac{a^4}{b^4} + 8890560\,\frac{a^5}{b^5} + 109157760\,\frac{a^6}{b^6} + 1256804640\,\frac{a^7}{b^7} + 13486636800\,\frac{a^8}{b^8} + 133880947200\,\frac{a^9}{b^9} + 1218420403200\,\frac{a^{10}}{b^{10}} + 10055600755200\,\frac{a^{11}}{b^{11}} + 74263450060800\,\frac{a^{12}}{b^{12}} + 482718652416000\,\frac{a^{13}}{b^{13}} + 2703224453529600\,\frac{a^{14}}{b^{14}} + 12671364625920000\,\frac{a^{15}}{b^{15}} + 47703960944640000\,\frac{a^{16}}{b^{16}} + 135161222676480000\,\frac{a^{17}}{b^{17}} + 256094948229120000\,\frac{a^{18}}{b^{18}} + 243290200817664000\,\frac{a^{19}}{b^{19}}$$

$$8 + 248\,\frac{a}{b} + 3680\,\frac{a^2}{b^2} + 52200\,\frac{a^3}{b^3} + 706176\,\frac{a^4}{b^4} + 9080640\,\frac{a^5}{b^5} + 110488320\,\frac{a^6}{b^6} + 1264788000\,\frac{a^7}{b^7} + 13526553600\,\frac{a^8}{b^8} + 134040614400\,\frac{a^9}{b^9} + 1218899404800\,\frac{a^{10}}{b^{10}} + 10056558758400\,\frac{a^{11}}{b^{11}} + 74264408064000\,\frac{a^{12}}{b^{12}} + 482718652416000\,\frac{a^{13}}{b^{13}} + 2703224453529600\,\frac{a^{14}}{b^{14}} + 12671364625920000\,\frac{a^{15}}{b^{15}} + 47703960944640000\,\frac{a^{16}}{b^{16}} + 135161222676480000\,\frac{a^{17}}{b^{17}} + 256094948229120000\,\frac{a^{18}}{b^{18}} + 243290200817664000\,\frac{a^{19}}{b^{19}}$$

$$9 + 270\,\frac{a}{b} + 3900\,\frac{a^2}{b^2} + 54180\,\frac{a^3}{b^3} + 722016\,\frac{a^4}{b^4} + 9191520\,\frac{a^5}{b^5} + 111153600\,\frac{a^6}{b^6} + 1268114400\,\frac{a^7}{b^7} + 13539859200\,\frac{a^8}{b^8} + 134080531200\,\frac{a^9}{b^9} + 1218979238400\,\frac{a^{10}}{b^{10}} + 10056638592000\,\frac{a^{11}}{b^{11}} + 74264408064000\,\frac{a^{12}}{b^{12}} + 482718652416000\,\frac{a^{13}}{b^{13}} + 2703224453529600\,\frac{a^{14}}{b^{14}} + 12671364625920000\,\frac{a^{15}}{b^{15}} + 47703960944640000\,\frac{a^{16}}{b^{16}} + 135161222676480000\,\frac{a^{17}}{b^{17}} + 256094948229120000\,\frac{a^{18}}{b^{18}} + 243290200817664000\,\frac{a^{19}}{b^{19}}$$

$$10 + 290\,\frac{a}{b} + 4080\,\frac{a^2}{b^2} + 55620\,\frac{a^3}{b^3} + 732096\,\frac{a^4}{b^4} + 9252000\,\frac{a^5}{b^5} +$$

$$111456000\,\frac{a^6}{b^6} + 1269324000\,\frac{a^7}{b^7} + 13543488000\,\frac{a^8}{b^8} + 134087788800\,\frac{a^9}{b^9} + 1218986496000\,\frac{a^{10}}{b^{10}} + 10056638592000\,\frac{a^{11}}{b^{11}} + 74264408064000\,\frac{a^{12}}{b^{12}} + 482718652416000\,\frac{a^{13}}{b^{13}} + 2703224453529600\,\frac{a^{14}}{b^{14}} + 12671364625920000\,\frac{a^{15}}{b^{15}} + 47703960944640000\,\frac{a^{16}}{b^{16}} + 135161222676480000\,\frac{a^{17}}{b^{17}} + 256094948229120000\,\frac{a^{18}}{b^{18}} + 243290200817664000\,\frac{a^{19}}{b^{19}}$$

$$11 + 308\,\frac{a}{b} + 4224\,\frac{a^2}{b^2} + 56628\,\frac{a^3}{b^3} + 738144\,\frac{a^4}{b^4} + 9282240\,\frac{a^5}{b^5} + 111576960\,\frac{a^6}{b^6} + 1269686880\,\frac{a^7}{b^7} + 13544213760\,\frac{a^8}{b^8} + 134088514560\,\frac{a^9}{b^9} + 1218986496000\,\frac{a^{10}}{b^{10}} + 10056638592000\,\frac{a^{11}}{b^{11}} + 74264408064000\,\frac{a^{12}}{b^{12}} + 482718652416000\,\frac{a^{13}}{b^{13}} + 2703224453529600\,\frac{a^{14}}{b^{14}} + 12671364625920000\,\frac{a^{15}}{b^{15}} + 47703960944640000\,\frac{a^{16}}{b^{16}} + 135161222676480000\,\frac{a^{17}}{b^{17}} + 256094948229120000\,\frac{a^{18}}{b^{18}} + 243290200817664000\,\frac{a^{19}}{b^{19}}$$

$$12 + 324\,\frac{a}{b} + 4336\,\frac{a^2}{b^2} + 57300\,\frac{a^3}{b^3} + 741504\,\frac{a^4}{b^4} + 9295680\,\frac{a^5}{b^5} + 111617280\,\frac{a^6}{b^6} + 1269767520\,\frac{a^7}{b^7} + 13544294400\,\frac{a^8}{b^8} + 134088514560\,\frac{a^9}{b^9} + 1218986496000\,\frac{a^{10}}{b^{10}} + 10056638592000\,\frac{a^{11}}{b^{11}} + 74264408064000\,\frac{a^{12}}{b^{12}} + 482718652416000\,\frac{a^{13}}{b^{13}} + 2703224453529600\,\frac{a^{14}}{b^{14}} + 12671364625920000\,\frac{a^{15}}{b^{15}} + 47703960944640000\,\frac{a^{16}}{b^{16}} + 135161222676480000\,\frac{a^{17}}{b^{17}} + 256094948229120000\,\frac{a^{18}}{b^{18}} + 243290200817664000\,\frac{a^{19}}{b^{19}}$$

$$13 + 338\,\frac{a}{b} + 4420\,\frac{a^2}{b^2} + 57720\,\frac{a^3}{b^3} + 743184\,\frac{a^4}{b^4} + 9300720\,\frac{a^5}{b^5} + 111627360\,\frac{a^6}{b^6} + 1269777600\,\frac{a^7}{b^7} + 13544294400\,\frac{a^8}{b^8} + 134088514560\,\frac{a^9}{b^9} + 1218986496000\,\frac{a^{10}}{b^{10}} + 10056638592000\,\frac{a^{11}}{b^{11}} + 74264408064000\,\frac{a^{12}}{b^{12}} + 482718652416000\,\frac{a^{13}}{b^{13}} + 2703224453529600\,\frac{a^{14}}{b^{14}} + 12671364625920000\,\frac{a^{15}}{b^{15}} + 47703960944640000\,\frac{a^{16}}{b^{16}} + 135161222676480000\,\frac{a^{17}}{b^{17}} + 256094948229120000\,\frac{a^{18}}{b^{18}} + 243290200817664000\,\frac{a^{19}}{b^{19}}$$

$$14 + 350\,\frac{a}{b} + 4480\,\frac{a^2}{b^2} + 57960\,\frac{a^3}{b^3} + 743904\,\frac{a^4}{b^4} + 9302160\,\frac{a^5}{b^5} + 111628800\,\frac{a^6}{b^6} + 1269777600\,\frac{a^7}{b^7} + 13544294400\,\frac{a^8}{b^8} + 134088514560\,\frac{a^9}{b^9} + 1218986496000\,\frac{a^{10}}{b^{10}} + 10056638592000\,\frac{a^{11}}{b^{11}} + 74264408064000\,\frac{a^{12}}{b^{12}} + 482718652416000\,\frac{a^{13}}{b^{13}} + 2703224453529600\,\frac{a^{14}}{b^{14}} + 12671364625920000\,\frac{a^{15}}{b^{15}} + 47703960944640000\,\frac{a^{16}}{b^{16}} + 135161222676480000\,\frac{a^{17}}{b^{17}} + 256094948229120000\,\frac{a^{18}}{b^{18}} + 243290200817664000\,\frac{a^{19}}{b^{19}}$$

$$15 + 360\,\frac{a}{b} + 4520\,\frac{a^2}{b^2} + 58080\,\frac{a^3}{b^3} + 744144\,\frac{a^4}{b^4} + 9302400\,\frac{a^5}{b^5} + 111628800\,\frac{a^6}{b^6} + 1269777600\,\frac{a^7}{b^7} + 13544294400\,\frac{a^8}{b^8} + 134088514560\,\frac{a^9}{b^9} + 1218986496000\,\frac{a^{10}}{b^{10}} + 10056638592000\,\frac{a^{11}}{b^{11}} + 74264408064000\,\frac{a^{12}}{b^{12}} + 482718652416000\,\frac{a^{13}}{b^{13}} + 2703224453529600\,\frac{a^{14}}{b^{14}} + 12671364625920000\,\frac{a^{15}}{b^{15}} + 47703960944640000\,\frac{a^{16}}{b^{16}} + 135161222676480000\,\frac{a^{17}}{b^{17}} + 256094948229120000\,\frac{a^{18}}{b^{18}} + 243290200817664000\,\frac{a^{19}}{b^{19}}$$

$$16 + 368\,\frac{a}{b} + 4544\,\frac{a^2}{b^2} + 58128\,\frac{a^3}{b^3} + 744192\,\frac{a^4}{b^4} + 9302400\,\frac{a^5}{b^5} + 111628800\,\frac{a^6}{b^6} + 1269777600\,\frac{a^7}{b^7} + 13544294400\,\frac{a^8}{b^8} + 134088514560\,\frac{a^9}{b^9} + 1218986496000\,\frac{a^{10}}{b^{10}} + 10056638592000\,\frac{a^{11}}{b^{11}} + 74264408064000\,\frac{a^{12}}{b^{12}} + 482718652416000\,\frac{a^{13}}{b^{13}} + 2703224453529600\,\frac{a^{14}}{b^{14}} + 12671364625920000\,\frac{a^{15}}{b^{15}} + 47703960944640000\,\frac{a^{16}}{b^{16}} + 135161222676480000\,\frac{a^{17}}{b^{17}} + 256094948229120000\,\frac{a^{18}}{b^{18}} + 243290200817664000\,\frac{a^{19}}{b^{19}}$$

$$17 + 374\,\frac{a}{b} + 4556\,\frac{a^2}{b^2} + 58140\,\frac{a^3}{b^3} + 744192\,\frac{a^4}{b^4} + 9302400\,\frac{a^5}{b^5} + 111628800\,\frac{a^6}{b^6} + 1269777600\,\frac{a^7}{b^7} + 13544294400\,\frac{a^8}{b^8} + 134088514560\,\frac{a^9}{b^9} + 1218986496000\,\frac{a^{10}}{b^{10}} + 10056638592000\,\frac{a^{11}}{b^{11}} + 74264408064000\,\frac{a^{12}}{b^{12}} + 482718652416000\,\frac{a^{13}}{b^{13}} +$$

$2703224453529600\,\frac{a^{14}}{b^{14}} + 12671364625920000\,\frac{a^{15}}{b^{15}} + 47703960944640000\,\frac{a^{16}}{b^{16}} + 135161222676480000\,\frac{a^{17}}{b^{17}} + 256094948229120000\,\frac{a^{18}}{b^{18}} + 243290200817664000\,\frac{a^{19}}{b^{19}}$

$18 + 378\,\frac{a}{b} + 4560\,\frac{a^2}{b^2} + 58140\,\frac{a^3}{b^3} + 744192\,\frac{a^4}{b^4} + 9302400\,\frac{a^5}{b^5} + 111628800\,\frac{a^6}{b^6} + 1269777600\,\frac{a^7}{b^7} + 13544294400\,\frac{a^8}{b^8} + 134088514560\,\frac{a^9}{b^9} + 1218986496000\,\frac{a^{10}}{b^{10}} + 10056638592000\,\frac{a^{11}}{b^{11}} + 74264408064000\,\frac{a^{12}}{b^{12}} + 482718652416000\,\frac{a^{13}}{b^{13}} + 2703224453529600\,\frac{a^{14}}{b^{14}} + 12671364625920000\,\frac{a^{15}}{b^{15}} + 47703960944640000\,\frac{a^{16}}{b^{16}} + 135161222676480000\,\frac{a^{17}}{b^{17}} + 256094948229120000\,\frac{a^{18}}{b^{18}} + 243290200817664000\,\frac{a^{19}}{b^{19}}$

$19 + 380\,\frac{a}{b} + 4560\,\frac{a^2}{b^2} + 58140\,\frac{a^3}{b^3} + 744192\,\frac{a^4}{b^4} + 9302400\,\frac{a^5}{b^5} + 111628800\,\frac{a^6}{b^6} + 1269777600\,\frac{a^7}{b^7} + 13544294400\,\frac{a^8}{b^8} + 134088514560\,\frac{a^9}{b^9} + 1218986496000\,\frac{a^{10}}{b^{10}} + 10056638592000\,\frac{a^{11}}{b^{11}} + 74264408064000\,\frac{a^{12}}{b^{12}} + 482718652416000\,\frac{a^{13}}{b^{13}} + 2703224453529600\,\frac{a^{14}}{b^{14}} + 12671364625920000\,\frac{a^{15}}{b^{15}} + 47703960944640000\,\frac{a^{16}}{b^{16}} + 135161222676480000\,\frac{a^{17}}{b^{17}} + 256094948229120000\,\frac{a^{18}}{b^{18}} + 243290200817664000\,\frac{a^{19}}{b^{19}}$

$20 + 380\,\frac{a}{b} + 4560\,\frac{a^2}{b^2} + 58140\,\frac{a^3}{b^3} + 744192\,\frac{a^4}{b^4} + 9302400\,\frac{a^5}{b^5} + 111628800\,\frac{a^6}{b^6} + 1269777600\,\frac{a^7}{b^7} + 13544294400\,\frac{a^8}{b^8} + 134088514560\,\frac{a^9}{b^9} + 1218986496000\,\frac{a^{10}}{b^{10}} + 10056638592000\,\frac{a^{11}}{b^{11}} + 74264408064000\,\frac{a^{12}}{b^{12}} + 482718652416000\,\frac{a^{13}}{b^{13}} + 2703224453529600\,\frac{a^{14}}{b^{14}} + 12671364625920000\,\frac{a^{15}}{b^{15}} + 47703960944640000\,\frac{a^{16}}{b^{16}} + 135161222676480000\,\frac{a^{17}}{b^{17}} + 256094948229120000\,\frac{a^{18}}{b^{18}} + 243290200817664000\,\frac{a^{19}}{b^{19}}$

25 módulos

$1 + 48\,\frac{a}{b} + 1104\,\frac{a^2}{b^2} + 24288\,\frac{a^3}{b^3} + 510048\,\frac{a^4}{b^4} + 10200960\,\frac{a^5}{b^5} + 193818240\,\frac{a^6}{b^6} + 3488728320\,\frac{a^7}{b^7} + 59308381440\,\frac{a^8}{b^8} + 948934103040\,\frac{a^9}{b^9} + 14234011545600\,\frac{a^{10}}{b^{10}} + 199276161638400\,\frac{a^{11}}{b^{11}} + 2590590101299200\,\frac{a^{12}}{b^{12}} + 31087081215590400\,\frac{a^{13}}{b^{13}} + 341957893371494400\,\frac{a^{14}}{b^{14}} + 3419578933714944000\,\frac{a^{15}}{b^{15}} + 30776210403434496000\,\frac{a^{16}}{b^{16}} + 246209683227475968000\,\frac{a^{17}}{b^{17}} + 1723467782592331776000\,\frac{a^{18}}{b^{18}} + 10340806695553990656000\,\frac{a^{19}}{b^{19}} + 51704033477769953280000\,\frac{a^{20}}{b^{20}} + 206816133911079813120000\,\frac{a^{21}}{b^{21}} + 620448401733239439360000\,\frac{a^{22}}{b^{22}} + 1240896803466478878720000\,\frac{a^{23}}{b^{23}} + 1240896803466478878720000\,\frac{a^{24}}{b^{24}}$

$2 + 94\,\frac{a}{b} + 2116\,\frac{a^2}{b^2} + 45540\,\frac{a^3}{b^3} + 935088\,\frac{a^4}{b^4} + 18276720\,\frac{a^5}{b^5} + 339181920\,\frac{a^6}{b^6} + 5959910880\,\frac{a^7}{b^7} + 98847302400\,\frac{a^8}{b^8} + 1542017917440\,\frac{a^9}{b^9} + 22537184947200\,\frac{a^{10}}{b^{10}} + 307217415859200\,\frac{a^{11}}{b^{11}} + 3885885151948800\,\frac{a^{12}}{b^{12}} + 45335326772736000\,\frac{a^{13}}{b^{13}} + 484440348942950400\,\frac{a^{14}}{b^{14}} + 4701921033858048000\,\frac{a^{15}}{b^{15}} + 41034947204579328000\,\frac{a^{16}}{b^{16}} + 318020840835489792000\,\frac{a^{17}}{b^{17}} + 2154334728240414720000\,\frac{a^{18}}{b^{18}} + 12495141423794405376000\,\frac{a^{19}}{b^{19}} + 60321372390731612160000\,\frac{a^{20}}{b^{20}} + 232668150649964789760000\,\frac{a^{21}}{b^{21}} + 672152435211009392640000\,\frac{a^{22}}{b^{22}} + 1292600836944248832000000\,\frac{a^{23}}{b^{23}} + 1240896803466478878720000\,\frac{a^{24}}{b^{24}}$

$3 + 138\,\frac{a}{b} + 3040\,\frac{a^2}{b^2} + 64020\,\frac{a^3}{b^3} + 1286208\,\frac{a^4}{b^4} + 24596880\,\frac{a^5}{b^5} + 446624640\,\frac{a^6}{b^6} + 7678994400\,\frac{a^7}{b^7} + 124633555200\,\frac{a^8}{b^8} + 1903025456640\,\frac{a^9}{b^9} + 27230282956800\,\frac{a^{10}}{b^{10}} +$

$$363534591974400\,\frac{a^{11}}{b^{11}} + 4505374089216000\,\frac{a^{12}}{b^{12}} + 51530216145408000\,\frac{a^{13}}{b^{13}} + 540194353296998400\,\frac{a^{14}}{b^{14}} + 5147953068690432000\,\frac{a^{15}}{b^{15}} + 44157171448406016000\,\frac{a^{16}}{b^{16}} + 336754186298449920000\,\frac{a^{17}}{b^{17}} + 2248001455555215360000\,\frac{a^{18}}{b^{18}} + 12869808333053607936000\,\frac{a^{19}}{b^{19}} + 61445373118509219840000\,\frac{a^{20}}{b^{20}} + 234916152105520005120000\,\frac{a^{21}}{b^{21}} + 674400436666564608000000\,\frac{a^{22}}{b^{22}} + 1292600836944248832000000\,\frac{a^{23}}{b^{23}} + 1240896803466478878720000\,\frac{a^{24}}{b^{24}}$$

$$4 + 180\,\frac{a}{b} + 3880\,\frac{a^2}{b^2} + 79980\,\frac{a^3}{b^3} + 1573488\,\frac{a^4}{b^4} + 29480640\,\frac{a^5}{b^5} + 524764800\,\frac{a^6}{b^6} + 8851096800\,\frac{a^7}{b^7} + 141042988800\,\frac{a^8}{b^8} + 2116348093440\,\frac{a^9}{b^9} + 29790154598400\,\frac{a^{10}}{b^{10}} + 391693180032000\,\frac{a^{11}}{b^{11}} + 4786959969792000\,\frac{a^{12}}{b^{12}} + 54064489070592000\,\frac{a^{13}}{b^{13}} + 560468536698470400\,\frac{a^{14}}{b^{14}} + 5289872352500736000\,\frac{a^{15}}{b^{15}} + 45008687151267840000\,\frac{a^{16}}{b^{16}} + 341011764812759040000\,\frac{a^{17}}{b^{17}} + 2265031769612451840000\,\frac{a^{18}}{b^{18}} + 12920899275225317376000\,\frac{a^{19}}{b^{19}} + 61547555002852638720000\,\frac{a^{20}}{b^{20}} + 235018333989863424000000\,\frac{a^{21}}{b^{21}} + 674400436666564608000000\,\frac{a^{22}}{b^{22}} + 1292600836944248832000000\,\frac{a^{23}}{b^{23}} + 1240896803466478878720000\,\frac{a^{24}}{b^{24}}$$

$$5 + 220\,\frac{a}{b} + 4640\,\frac{a^2}{b^2} + 93660\,\frac{a^3}{b^3} + 1806048\,\frac{a^4}{b^4} + 33201600\,\frac{a^5}{b^5} + 580579200\,\frac{a^6}{b^6} + 9632498400\,\frac{a^7}{b^7} + 151201209600\,\frac{a^8}{b^8} + 2238246743040\,\frac{a^9}{b^9} + 31131039744000\,\frac{a^{10}}{b^{10}} + 405102031488000\,\frac{a^{11}}{b^{11}} + 4907639632896000\,\frac{a^{12}}{b^{12}} + 55029926375424000\,\frac{a^{13}}{b^{13}} + 567226597832294400\,\frac{a^{14}}{b^{14}} + 5330420719303680000\,\frac{a^{15}}{b^{15}} + 45211428985282560000\,\frac{a^{16}}{b^{16}} + 341822732148817920000\,\frac{a^{17}}{b^{17}} + 2267464671620628480000\,\frac{a^{18}}{b^{18}} + 12925765079241670656000\,\frac{a^{19}}{b^{19}} + 61552420806868992000000\,\frac{a^{20}}{b^{20}} + 235018333989863424000000\,\frac{a^{21}}{b^{21}} + 674400436666564608000000\,\frac{a^{22}}{b^{22}} + 1292600836944248832000000\,\frac{a^{23}}{b^{23}} + 1240896803466478878720000\,\frac{a^{24}}{b^{24}}$$

$$6 + 258\,\frac{a}{b} + 5324\,\frac{a^2}{b^2} + 105288\,\frac{a^3}{b^3} + 1992096\,\frac{a^4}{b^4} + 35992320\,\frac{a^5}{b^5} + 619649280\,\frac{a^6}{b^6} + 10140409440\,\frac{a^7}{b^7} + 157296142080\,\frac{a^8}{b^8} + 2305291000320\,\frac{a^9}{b^9} + 31801482316800\,\frac{a^{10}}{b^{10}} + 411136014643200\,\frac{a^{11}}{b^{11}} + 4955911498137600\,\frac{a^{12}}{b^{12}} + 55367829432115200\,\frac{a^{13}}{b^{13}} + 569254016172441600\,\frac{a^{14}}{b^{14}} + 5340557811004416000\,\frac{a^{15}}{b^{15}} + 45251977352085504000\,\frac{a^{16}}{b^{16}} + 341944377249226752000\,\frac{a^{17}}{b^{17}} + 2267707961821446144000\,\frac{a^{18}}{b^{18}} + 12926008369442488320000\,\frac{a^{19}}{b^{19}} + 61552420806868992000000\,\frac{a^{20}}{b^{20}} + 235018333989863424000000\,\frac{a^{21}}{b^{21}} + 674400436666564608000000\,\frac{a^{22}}{b^{22}} + 1292600836944248832000000\,\frac{a^{23}}{b^{23}} + 1240896803466478878720000\,\frac{a^{24}}{b^{24}}$$

$$7 + 294\,\frac{a}{b} + 5936\,\frac{a^2}{b^2} + 115080\,\frac{a^3}{b^3} + 2138976\,\frac{a^4}{b^4} + 38048640\,\frac{a^5}{b^5} + 646381440\,\frac{a^6}{b^6} + 10461195360\,\frac{a^7}{b^7} + 160824787200\,\frac{a^8}{b^8} + 2340577451520\,\frac{a^9}{b^9} + 32119060377600\,\frac{a^{10}}{b^{10}} + 413676639129600\,\frac{a^{11}}{b^{11}} + 4973695869542400\,\frac{a^{12}}{b^{12}} + 55474535660544000\,\frac{a^{13}}{b^{13}} + 569787547314585600\,\frac{a^{14}}{b^{14}} + 5342691935572992000\,\frac{a^{15}}{b^{15}} + 45258379725791232000\,\frac{a^{16}}{b^{16}} + 341957181996638208000\,\frac{a^{17}}{b^{17}} + 2267720766568857600000\,\frac{a^{18}}{b^{18}} + 12926008369442488320000\,\frac{a^{19}}{b^{19}} + 61552420806868992000000\,\frac{a^{20}}{b^{20}} + 235018333989863424000000\,\frac{a^{21}}{b^{21}} + 674400436666564608000000\,\frac{a^{22}}{b^{22}} +$$

$$1292600836944248832000000\,\frac{a^{23}}{b^{23}} + 124089680346647887872000\,\frac{a^{24}}{b^{24}}$$

$$8 + 328\,\frac{a}{b} + 6480\,\frac{a^2}{b^2} + 123240\,\frac{a^3}{b^3} + 2253216\,\frac{a^4}{b^4} + 39533760\,\frac{a^5}{b^5} + 664202880\,\frac{a^6}{b^6} +$$
$$10657231200\,\frac{a^7}{b^7} + 162785145600\,\frac{a^8}{b^8} + 2358220677120\,\frac{a^9}{b^9} + 32260206182400\,\frac{a^{10}}{b^{10}} +$$
$$414664659763200\,\frac{a^{11}}{b^{11}} + 4979623993344000\,\frac{a^{12}}{b^{12}} + 55504176279552000\,\frac{a^{13}}{b^{13}} +$$
$$569906109790617600\,\frac{a^{14}}{b^{14}} + 5343047623001088000\,\frac{a^{15}}{b^{15}} + 45259091100647424000\,\frac{a^{16}}{b^{16}} +$$
$$341957893371494400000\,\frac{a^{17}}{b^{17}} + 2267720766568857600000\,\frac{a^{18}}{b^{18}} +$$
$$129260083694424888320000\,\frac{a^{19}}{b^{19}} + 615524208068689920000000\,\frac{a^{20}}{b^{20}} +$$
$$2350183339898634240000000\,\frac{a^{21}}{b^{21}} + 6744004366666564608000000\,\frac{a^{22}}{b^{22}} +$$
$$1292600836944248832000000\,\frac{a^{23}}{b^{23}} + 124089680346647887872000\,\frac{a^{24}}{b^{24}}$$

$$9 + 360\,\frac{a}{b} + 6960\,\frac{a^2}{b^2} + 129960\,\frac{a^3}{b^3} + 2340576\,\frac{a^4}{b^4} + 40582080\,\frac{a^5}{b^5} + 675734400\,\frac{a^6}{b^6} +$$
$$10772546400\,\frac{a^7}{b^7} + 163822982400\,\frac{a^8}{b^8} + 2366523371520\,\frac{a^9}{b^9} + 32318325043200\,\frac{a^{10}}{b^{10}} +$$
$$415013372928000\,\frac{a^{11}}{b^{11}} + 4981367559168000\,\frac{a^{12}}{b^{12}} + 55511150542848000\,\frac{a^{13}}{b^{13}} +$$
$$569927032580505600\,\frac{a^{14}}{b^{14}} + 5343089468580864000\,\frac{a^{15}}{b^{15}} + 45259132946227200000\,\frac{a^{16}}{b^{16}} +$$
$$341957893371494400000\,\frac{a^{17}}{b^{17}} + 2267720766568857600000\,\frac{a^{18}}{b^{18}} +$$
$$129260083694424883200000\,\frac{a^{19}}{b^{19}} + 615524208068689920000000\,\frac{a^{20}}{b^{20}} +$$
$$2350183339898634240000000\,\frac{a^{21}}{b^{21}} + 6744004366666564608000000\,\frac{a^{22}}{b^{22}} +$$
$$1292600836944248832000000\,\frac{a^{23}}{b^{23}} + 124089680346647887872000\,\frac{a^{24}}{b^{24}}$$

$$10 + 390\,\frac{a}{b} + 7380\,\frac{a^2}{b^2} + 135420\,\frac{a^3}{b^3} + 2406096\,\frac{a^4}{b^4} + 41302800\,\frac{a^5}{b^5} + 682941600\,\frac{a^6}{b^6} +$$
$$10837411200\,\frac{a^7}{b^7} + 164341900800\,\frac{a^8}{b^8} + 2370155800320\,\frac{a^9}{b^9} + 32340119616000\,\frac{a^{10}}{b^{10}} +$$
$$415122345792000\,\frac{a^{11}}{b^{11}} + 4981803450624000\,\frac{a^{12}}{b^{12}} + 55512458217216000\,\frac{a^{13}}{b^{13}} +$$
$$569929647929241600\,\frac{a^{14}}{b^{14}} + 5343092083929600000\,\frac{a^{15}}{b^{15}} + 45259132946227200000\,\frac{a^{16}}{b^{16}} +$$
$$341957893371494400000\,\frac{a^{17}}{b^{17}} + 2267720766568857600000\,\frac{a^{18}}{b^{18}} +$$
$$129260083694424883200000\,\frac{a^{19}}{b^{19}} + 615524208068689920000000\,\frac{a^{20}}{b^{20}} +$$
$$2350183339898634240000000\,\frac{a^{21}}{b^{21}} + 6744004366666564608000000\,\frac{a^{22}}{b^{22}} +$$
$$1292600836944248832000000\,\frac{a^{23}}{b^{23}} + 124089680346647887872000\,\frac{a^{24}}{b^{24}}$$

$$11 + 418\,\frac{a}{b} + 7744\,\frac{a^2}{b^2} + 139788\,\frac{a^3}{b^3} + 2454144\,\frac{a^4}{b^4} + 41783280\,\frac{a^5}{b^5} + 687265920\,\frac{a^6}{b^6} +$$
$$10872005760\,\frac{a^7}{b^7} + 164584062720\,\frac{a^8}{b^8} + 2371608771840\,\frac{a^9}{b^9} + 32347384473600\,\frac{a^{10}}{b^{10}} +$$
$$415151405222400\,\frac{a^{11}}{b^{11}} + 4981890628915200\,\frac{a^{12}}{b^{12}} + 55512632573798400\,\frac{a^{13}}{b^{13}} +$$
$$569929822285824000\,\frac{a^{14}}{b^{14}} + 5343092083929600000\,\frac{a^{15}}{b^{15}} + 45259132946227200000\,\frac{a^{16}}{b^{16}} +$$
$$341957893371494400000\,\frac{a^{17}}{b^{17}} + 2267720766568857600000\,\frac{a^{18}}{b^{18}} +$$
$$129260083694424888320000\,\frac{a^{19}}{b^{19}} + 615524208068689920000000\,\frac{a^{20}}{b^{20}} +$$
$$2350183339898634240000000\,\frac{a^{21}}{b^{21}} + 6744004366666564608000000\,\frac{a^{22}}{b^{22}} +$$
$$1292600836944248832000000\,\frac{a^{23}}{b^{23}} + 124089680346647887872000\,\frac{a^{24}}{b^{24}}$$

$$12 + 444\,\frac{a}{b} + 8056\,\frac{a^2}{b^2} + 143220\,\frac{a^3}{b^3} + 2488464\,\frac{a^4}{b^4} + 42092160\,\frac{a^5}{b^5} + 689736960\,\frac{a^6}{b^6} +$$
$$10889303040\,\frac{a^7}{b^7} + 164687846400\,\frac{a^8}{b^8} + 2372127690240\,\frac{a^9}{b^9} + 32349460147200\,\frac{a^{10}}{b^{10}} +$$
$$415157632243200\,\frac{a^{11}}{b^{11}} + 4981903082956800\,\frac{a^{12}}{b^{12}} + 55512645027840000\,\frac{a^{13}}{b^{13}} +$$
$$569929822285824000\,\frac{a^{14}}{b^{14}} + 5343092083929600000\,\frac{a^{15}}{b^{15}} + 45259132946227200000\,\frac{a^{16}}{b^{16}} +$$

$$341957893371494400000\,\frac{a^{17}}{b^{17}} + 2267720766568857600000\,\frac{a^{18}}{b^{18}} + 1292600836944248832000000\,\frac{a^{23}}{b^{23}} + 1292600836944248832000000\,\frac{a^{23}}{b^{23}} + 1240896803466478878720000\,\frac{a^{24}}{b^{24}}$$

$$13 + 468\,\frac{a}{b} + 8320\,\frac{a^2}{b^2} + 145860\,\frac{a^3}{b^3} + 2512224\,\frac{a^4}{b^4} + 42282240\,\frac{a^5}{b^5} + 691067520\,\frac{a^6}{b^6} + 10897286400\,\frac{a^7}{b^7} + 164727763200\,\frac{a^8}{b^8} + 2372287357440\,\frac{a^9}{b^9} + 32349939148800\,\frac{a^{10}}{b^{10}} + 415158590246400\,\frac{a^{11}}{b^{11}} + 4981904040960000\,\frac{a^{12}}{b^{12}} + 55512645027840000\,\frac{a^{13}}{b^{13}} + 569929822285824000\,\frac{a^{14}}{b^{14}} + 5343092083929600000\,\frac{a^{15}}{b^{15}} + 45259132946227200000\,\frac{a^{16}}{b^{16}} + 341957893371494400000\,\frac{a^{17}}{b^{17}} + 2267720766568857600000\,\frac{a^{18}}{b^{18}} + 1292600836944248832000000\,\frac{a^{23}}{b^{23}} + 1240896803466478878720000\,\frac{a^{24}}{b^{24}}$$

$$14 + 490\,\frac{a}{b} + 8540\,\frac{a^2}{b^2} + 147840\,\frac{a^3}{b^3} + 2528064\,\frac{a^4}{b^4} + 42393120\,\frac{a^5}{b^5} + 691732800\,\frac{a^6}{b^6} + 10900612800\,\frac{a^7}{b^7} + 164741068800\,\frac{a^8}{b^8} + 2372327274240\,\frac{a^9}{b^9} + 32350018982400\,\frac{a^{10}}{b^{10}} + 415158670080000\,\frac{a^{11}}{b^{11}} + 4981904040960000\,\frac{a^{12}}{b^{12}} + 55512645027840000\,\frac{a^{13}}{b^{13}} + 569929822285824000\,\frac{a^{14}}{b^{14}} + 5343092083929600000\,\frac{a^{15}}{b^{15}} + 45259132946227200000\,\frac{a^{16}}{b^{16}} + 341957893371494400000\,\frac{a^{17}}{b^{17}} + 2267720766568857600000\,\frac{a^{18}}{b^{18}} + 1292600836944248832000000\,\frac{a^{23}}{b^{23}} + 1240896803466478878720000\,\frac{a^{24}}{b^{24}}$$

$$15 + 510\,\frac{a}{b} + 8720\,\frac{a^2}{b^2} + 149280\,\frac{a^3}{b^3} + 2538144\,\frac{a^4}{b^4} + 42453600\,\frac{a^5}{b^5} + 692035200\,\frac{a^6}{b^6} + 10901822400\,\frac{a^7}{b^7} + 164744697600\,\frac{a^8}{b^8} + 2372334531840\,\frac{a^9}{b^9} + 32350026240000\,\frac{a^{10}}{b^{10}} + 415158670080000\,\frac{a^{11}}{b^{11}} + 4981904040960000\,\frac{a^{12}}{b^{12}} + 55512645027840000\,\frac{a^{13}}{b^{13}} + 569929822285824000\,\frac{a^{14}}{b^{14}} + 5343092083929600000\,\frac{a^{15}}{b^{15}} + 45259132946227200000\,\frac{a^{16}}{b^{16}} + 341957893371494400000\,\frac{a^{17}}{b^{17}} + 2267720766568857600000\,\frac{a^{18}}{b^{18}} + 1292600836944248832000000\,\frac{a^{23}}{b^{23}} + 1240896803466478878720000\,\frac{a^{24}}{b^{24}}$$

$$16 + 528\,\frac{a}{b} + 8864\,\frac{a^2}{b^2} + 150288\,\frac{a^3}{b^3} + 2544192\,\frac{a^4}{b^4} + 42483840\,\frac{a^5}{b^5} + 692156160\,\frac{a^6}{b^6} + 10902185280\,\frac{a^7}{b^7} + 164745423360\,\frac{a^8}{b^8} + 2372335257600\,\frac{a^9}{b^9} + 32350026240000\,\frac{a^{10}}{b^{10}} + 415158670080000\,\frac{a^{11}}{b^{11}} + 4981904040960000\,\frac{a^{12}}{b^{12}} + 55512645027840000\,\frac{a^{13}}{b^{13}} + 569929822285824000\,\frac{a^{14}}{b^{14}} + 5343092083929600000\,\frac{a^{15}}{b^{15}} + 45259132946227200000\,\frac{a^{16}}{b^{16}} + 341957893371494400000\,\frac{a^{17}}{b^{17}} + 2267720766568857600000\,\frac{a^{18}}{b^{18}} + 1292600836944248832000000\,\frac{a^{23}}{b^{23}} + 1240896803466478878720000\,\frac{a^{24}}{b^{24}}$$

$$17 + 544\,\frac{a}{b} + 8976\,\frac{a^2}{b^2} + 150960\,\frac{a^3}{b^3} + 2547552\,\frac{a^4}{b^4} + 42497280\,\frac{a^5}{b^5} + 692196480\,\frac{a^6}{b^6} +$$

$$10902265920\,\frac{a^7}{b^7} + 164745504000\,\frac{a^8}{b^8} + 2372335257600\,\frac{a^9}{b^9} + 32350026240000\,\frac{a^{10}}{b^{10}} + 415158670080000\,\frac{a^{11}}{b^{11}} + 4981904040960000\,\frac{a^{12}}{b^{12}} + 55512645027840000\,\frac{a^{13}}{b^{13}} + 569929822285824000\,\frac{a^{14}}{b^{14}} + 5343092083929600000\,\frac{a^{15}}{b^{15}} + 45259132946227200000\,\frac{a^{16}}{b^{16}} + 341957893371494400000\,\frac{a^{17}}{b^{17}} + 2267720766568857600000\,\frac{a^{18}}{b^{18}} + 12926008369442488320000\,\frac{a^{19}}{b^{19}} + 61552420806868992000000\,\frac{a^{20}}{b^{20}} + 235018333989863424000000\,\frac{a^{21}}{b^{21}} + 674400436666564608000000\,\frac{a^{22}}{b^{22}} + 1292600836944248832000000\,\frac{a^{23}}{b^{23}} + 1240896803466478878720000\,\frac{a^{24}}{b^{24}}$$

$$18 + 558\,\frac{a}{b} + 9060\,\frac{a^2}{b^2} + 151380\,\frac{a^3}{b^3} + 2549232\,\frac{a^4}{b^4} + 42502320\,\frac{a^5}{b^5} + 692206560\,\frac{a^6}{b^6} + 10902276000\,\frac{a^7}{b^7} + 164745504000\,\frac{a^8}{b^8} + 2372335257600\,\frac{a^9}{b^9} + 32350026240000\,\frac{a^{10}}{b^{10}} + 415158670080000\,\frac{a^{11}}{b^{11}} + 4981904040960000\,\frac{a^{12}}{b^{12}} + 55512645027840000\,\frac{a^{13}}{b^{13}} + 569929822285824000\,\frac{a^{14}}{b^{14}} + 5343092083929600000\,\frac{a^{15}}{b^{15}} + 45259132946227200000\,\frac{a^{16}}{b^{16}} + 341957893371494400000\,\frac{a^{17}}{b^{17}} + 2267720766568857600000\,\frac{a^{18}}{b^{18}} + 12926008369442488320000\,\frac{a^{19}}{b^{19}} + 61552420806868992000000\,\frac{a^{20}}{b^{20}} + 235018333989863424000000\,\frac{a^{21}}{b^{21}} + 674400436666564608000000\,\frac{a^{22}}{b^{22}} + 1292600836944248832000000\,\frac{a^{23}}{b^{23}} + 1240896803466478878720000\,\frac{a^{24}}{b^{24}}$$

$$19 + 570\,\frac{a}{b} + 9120\,\frac{a^2}{b^2} + 151620\,\frac{a^3}{b^3} + 2549952\,\frac{a^4}{b^4} + 42503760\,\frac{a^5}{b^5} + 692208000\,\frac{a^6}{b^6} + 10902276000\,\frac{a^7}{b^7} + 164745504000\,\frac{a^8}{b^8} + 2372335257600\,\frac{a^9}{b^9} + 32350026240000\,\frac{a^{10}}{b^{10}} + 415158670080000\,\frac{a^{11}}{b^{11}} + 4981904040960000\,\frac{a^{12}}{b^{12}} + 55512645027840000\,\frac{a^{13}}{b^{13}} + 569929822285824000\,\frac{a^{14}}{b^{14}} + 5343092083929600000\,\frac{a^{15}}{b^{15}} + 45259132946227200000\,\frac{a^{16}}{b^{16}} + 341957893371494400000\,\frac{a^{17}}{b^{17}} + 2267720766568857600000\,\frac{a^{18}}{b^{18}} + 12926008369442488320000\,\frac{a^{19}}{b^{19}} + 61552420806868992000000\,\frac{a^{20}}{b^{20}} + 235018333989863424000000\,\frac{a^{21}}{b^{21}} + 674400436666564608000000\,\frac{a^{22}}{b^{22}} + 1292600836944248832000000\,\frac{a^{23}}{b^{23}} + 1240896803466478878720000\,\frac{a^{24}}{b^{24}}$$

$$20 + 580\,\frac{a}{b} + 9160\,\frac{a^2}{b^2} + 151740\,\frac{a^3}{b^3} + 2550192\,\frac{a^4}{b^4} + 42504000\,\frac{a^5}{b^5} + 692208000\,\frac{a^6}{b^6} + 10902276000\,\frac{a^7}{b^7} + 164745504000\,\frac{a^8}{b^8} + 2372335257600\,\frac{a^9}{b^9} + 32350026240000\,\frac{a^{10}}{b^{10}} + 415158670080000\,\frac{a^{11}}{b^{11}} + 4981904040960000\,\frac{a^{12}}{b^{12}} + 55512645027840000\,\frac{a^{13}}{b^{13}} + 569929822285824000\,\frac{a^{14}}{b^{14}} + 5343092083929600000\,\frac{a^{15}}{b^{15}} + 45259132946227200000\,\frac{a^{16}}{b^{16}} + 341957893371494400000\,\frac{a^{17}}{b^{17}} + 2267720766568857600000\,\frac{a^{18}}{b^{18}} + 12926008369442488320000\,\frac{a^{19}}{b^{19}} + 61552420806868992000000\,\frac{a^{20}}{b^{20}} + 235018333989863424000000\,\frac{a^{21}}{b^{21}} + 674400436666564608000000\,\frac{a^{22}}{b^{22}} + 1292600836944248832000000\,\frac{a^{23}}{b^{23}} + 1240896803466478878720000\,\frac{a^{24}}{b^{24}}$$

$$21 + 588\,\frac{a}{b} + 9184\,\frac{a^2}{b^2} + 151788\,\frac{a^3}{b^3} + 2550240\,\frac{a^4}{b^4} + 42504000\,\frac{a^5}{b^5} + 692208000\,\frac{a^6}{b^6} + 10902276000\,\frac{a^7}{b^7} + 164745504000\,\frac{a^8}{b^8} + 2372335257600\,\frac{a^9}{b^9} + 32350026240000\,\frac{a^{10}}{b^{10}} + 415158670080000\,\frac{a^{11}}{b^{11}} + 4981904040960000\,\frac{a^{12}}{b^{12}} + 55512645027840000\,\frac{a^{13}}{b^{13}} + 569929822285824000\,\frac{a^{14}}{b^{14}} + 5343092083929600000\,\frac{a^{15}}{b^{15}} + 45259132946227200000\,\frac{a^{16}}{b^{16}} + 341957893371494400000\,\frac{a^{17}}{b^{17}} + 2267720766568857600000\,\frac{a^{18}}{b^{18}} + 12926008369442488320000\,\frac{a^{19}}{b^{19}} + 61552420806868992000000\,\frac{a^{20}}{b^{20}} +$$

$$235018333989863424000000\,\frac{a^{21}}{b^{21}} + 6744004366666564608000000\,\frac{a^{22}}{b^{22}} + 1292600836944248832000000\,\frac{a^{23}}{b^{23}} + 1240896803466478878720000\,\frac{a^{24}}{b^{24}}$$

$$22 + 594\,\frac{a}{b} + 9196\,\frac{a^2}{b^2} + 151800\,\frac{a^3}{b^3} + 2550240\,\frac{a^4}{b^4} + 42504000\,\frac{a^5}{b^5} + 692208000\,\frac{a^6}{b^6} + 10902276000\,\frac{a^7}{b^7} + 164745504000\,\frac{a^8}{b^8} + 2372335257600\,\frac{a^9}{b^9} + 32350026240000\,\frac{a^{10}}{b^{10}} + 415158670080000\,\frac{a^{11}}{b^{11}} + 4981904040960000\,\frac{a^{12}}{b^{12}} + 55512645027840000\,\frac{a^{13}}{b^{13}} + 569929822285824000\,\frac{a^{14}}{b^{14}} + 5343092083929600000\,\frac{a^{15}}{b^{15}} + 45259132946227200000\,\frac{a^{16}}{b^{16}} + 341957893371494400000\,\frac{a^{17}}{b^{17}} + 2267720766568857600000\,\frac{a^{18}}{b^{18}} + 1292600836944248832000000\,\frac{a^{19}}{b^{19}} + 6155242080686899200000\,\frac{a^{20}}{b^{20}} + 235018333989863424000000\,\frac{a^{21}}{b^{21}} + 6744004366666564608000000\,\frac{a^{22}}{b^{22}} + 1292600836944248832000000\,\frac{a^{23}}{b^{23}} + 1240896803466478878720000\,\frac{a^{24}}{b^{24}}$$

$$23 + 598\,\frac{a}{b} + 9200\,\frac{a^2}{b^2} + 151800\,\frac{a^3}{b^3} + 2550240\,\frac{a^4}{b^4} + 42504000\,\frac{a^5}{b^5} + 692208000\,\frac{a^6}{b^6} + 10902276000\,\frac{a^7}{b^7} + 164745504000\,\frac{a^8}{b^8} + 2372335257600\,\frac{a^9}{b^9} + 32350026240000\,\frac{a^{10}}{b^{10}} + 415158670080000\,\frac{a^{11}}{b^{11}} + 4981904040960000\,\frac{a^{12}}{b^{12}} + 55512645027840000\,\frac{a^{13}}{b^{13}} + 569929822285824000\,\frac{a^{14}}{b^{14}} + 5343092083929600000\,\frac{a^{15}}{b^{15}} + 45259132946227200000\,\frac{a^{16}}{b^{16}} + 341957893371494400000\,\frac{a^{17}}{b^{17}} + 2267720766568857600000\,\frac{a^{18}}{b^{18}} + 1292600836944248832000000\,\frac{a^{19}}{b^{19}} + 6155242080686899200000\,\frac{a^{20}}{b^{20}} + 235018333989863424000000\,\frac{a^{21}}{b^{21}} + 6744004366666564608000000\,\frac{a^{22}}{b^{22}} + 1292600836944248832000000\,\frac{a^{23}}{b^{23}} + 1240896803466478878720000\,\frac{a^{24}}{b^{24}}$$

$$24 + 600\,\frac{a}{b} + 9200\,\frac{a^2}{b^2} + 151800\,\frac{a^3}{b^3} + 2550240\,\frac{a^4}{b^4} + 42504000\,\frac{a^5}{b^5} + 692208000\,\frac{a^6}{b^6} + 10902276000\,\frac{a^7}{b^7} + 164745504000\,\frac{a^8}{b^8} + 2372335257600\,\frac{a^9}{b^9} + 32350026240000\,\frac{a^{10}}{b^{10}} + 415158670080000\,\frac{a^{11}}{b^{11}} + 4981904040960000\,\frac{a^{12}}{b^{12}} + 55512645027840000\,\frac{a^{13}}{b^{13}} + 569929822285824000\,\frac{a^{14}}{b^{14}} + 5343092083929600000\,\frac{a^{15}}{b^{15}} + 45259132946227200000\,\frac{a^{16}}{b^{16}} + 341957893371494400000\,\frac{a^{17}}{b^{17}} + 2267720766568857600000\,\frac{a^{18}}{b^{18}} + 1292600836944248832000000\,\frac{a^{19}}{b^{19}} + 6155242080686899200000\,\frac{a^{20}}{b^{20}} + 235018333989863424000000\,\frac{a^{21}}{b^{21}} + 6744004366666564608000000\,\frac{a^{22}}{b^{22}} + 1292600836944248832000000\,\frac{a^{23}}{b^{23}} + 1240896803466478878720000\,\frac{a^{24}}{b^{24}}$$

$$25 + 600\,\frac{a}{b} + 9200\,\frac{a^2}{b^2} + 151800\,\frac{a^3}{b^3} + 2550240\,\frac{a^4}{b^4} + 42504000\,\frac{a^5}{b^5} + 692208000\,\frac{a^6}{b^6} + 10902276000\,\frac{a^7}{b^7} + 164745504000\,\frac{a^8}{b^8} + 2372335257600\,\frac{a^9}{b^9} + 32350026240000\,\frac{a^{10}}{b^{10}} + 415158670080000\,\frac{a^{11}}{b^{11}} + 4981904040960000\,\frac{a^{12}}{b^{12}} + 55512645027840000\,\frac{a^{13}}{b^{13}} + 569929822285824000\,\frac{a^{14}}{b^{14}} + 5343092083929600000\,\frac{a^{15}}{b^{15}} + 45259132946227200000\,\frac{a^{16}}{b^{16}} + 341957893371494400000\,\frac{a^{17}}{b^{17}} + 2267720766568857600000\,\frac{a^{18}}{b^{18}} + 1292600836944248832000000\,\frac{a^{19}}{b^{19}} + 6155242080686899200000\,\frac{a^{20}}{b^{20}} + 235018333989863424000000\,\frac{a^{21}}{b^{21}} + 6744004366666564608000000\,\frac{a^{22}}{b^{22}} + 1292600836944248832000000\,\frac{a^{23}}{b^{23}} + 1240896803466478878720000\,\frac{a^{24}}{b^{24}}$$

A.2 Resultados Obtidos Considerando-se a Conectividade

4 módulos

$$\begin{bmatrix} 1+12\,\frac{a^3f^3}{b^3}+12\,\frac{a^2f^2}{b^2}+6\,\frac{af}{b} \\ 2+12\,\frac{a^3f^3}{b^3}+16\,\frac{a^2f^2}{b^2}+10\,\frac{af}{b} \\ 3+12\,\frac{a^3f^3}{b^3}+16\,\frac{a^2f^2}{b^2}+12\,\frac{af}{b} \\ 4+12\,\frac{a^3f^3}{b^3}+16\,\frac{a^2f^2}{b^2}+12\,\frac{af}{b} \end{bmatrix}$$

5 módulos

$$\begin{bmatrix} 1+48\,\frac{a^4f^4}{b^4}+48\,\frac{a^3f^3}{b^3}+24\,\frac{a^2f^2}{b^2}+8\,\frac{af}{b} \\ 2+48\,\frac{a^4f^4}{b^4}+60\,\frac{a^3f^3}{b^3}+36\,\frac{a^2f^2}{b^2}+14\,\frac{af}{b} \\ 3+48\,\frac{a^4f^4}{b^4}+60\,\frac{a^3f^3}{b^3}+40\,\frac{a^2f^2}{b^2}+18\,\frac{af}{b} \\ 4+48\,\frac{a^4f^4}{b^4}+60\,\frac{a^3f^3}{b^3}+40\,\frac{a^2f^2}{b^2}+20\,\frac{af}{b} \\ 5+48\,\frac{a^4f^4}{b^4}+60\,\frac{a^3f^3}{b^3}+40\,\frac{a^2f^2}{b^2}+20\,\frac{af}{b} \end{bmatrix}$$

6 módulos

$$\begin{bmatrix} 1+240\,\frac{a^5f^5}{b^5}+240\,\frac{a^4f^4}{b^4}+120\,\frac{a^3f^3}{b^3}+40\,\frac{a^2f^2}{b^2}+10\,\frac{af}{b} \\ 2+240\,\frac{a^5f^5}{b^5}+288\,\frac{a^4f^4}{b^4}+168\,\frac{a^3f^3}{b^3}+64\,\frac{a^2f^2}{b^2}+18\,\frac{af}{b} \\ 3+240\,\frac{a^5f^5}{b^5}+288\,\frac{a^4f^4}{b^4}+180\,\frac{a^3f^3}{b^3}+76\,\frac{a^2f^2}{b^2}+24\,\frac{af}{b} \\ 4+240\,\frac{a^5f^5}{b^5}+288\,\frac{a^4f^4}{b^4}+180\,\frac{a^3f^3}{b^3}+80\,\frac{a^2f^2}{b^2}+28\,\frac{af}{b} \\ 5+240\,\frac{a^5f^5}{b^5}+288\,\frac{a^4f^4}{b^4}+180\,\frac{a^3f^3}{b^3}+80\,\frac{a^2f^2}{b^2}+30\,\frac{af}{b} \\ 6+240\,\frac{a^5f^5}{b^5}+288\,\frac{a^4f^4}{b^4}+180\,\frac{a^3f^3}{b^3}+80\,\frac{a^2f^2}{b^2}+30\,\frac{af}{b} \end{bmatrix}$$