ESCOLA SUPERIOR ABERTA DO BRASIL - ESAB

CURSO LATU SENSO EM ENGENHARIA DE SISTEMAS

DAVI GOMES DA COSTA

EXTRAÇÃO DE MÉTRICAS EM SOFTWARE ORIENTADO A OBJETOS

VILA VELHA – ES

2010

DAVI GOMES DA COSTA

EXTRAÇÃO DE MÉTRICAS EM SOFTWARE ORIENTADO A OBJETOS

Monografia apresentada à ESAB – Escola Superior Aberta do Brasil, sob Orientação do Prof.: Ms. Aloísio Silva.

VILA VELHA – ES

2010

EPIGRAFE

Determinação coragem e autoconfiança são fatores decisivos para o sucesso. Se estivermos possuídos por uma inabalável determinação conseguiremos superá-los. Independetemente das circunstâncias devemos ser sempre humildes, recatados e despidos de orgulho. Dalai Lama

DEDICATÓRIA

Dedico este trabalho a todos que sempre estiveram ao meu lado, fisicamente e no coração.

AGRADECIMENTOS

Gostaria de agradecer primeiro à Deus, sem Ele não estaria aqui, sou muito grato pela força de vontade e sabedoria para a elaboração deste trabalho.

Agradeço aos meus familiares por todo carinho e dedicação que sempre tiveram por mim, em especial meus pais Luiz Gonzaga da Costa e Maria José Gomes da Costa e meus irmãos Luiz Gonzaga da Costa Filho e Raquel Gomes da Costa Herzig, que não importando a situação, nunca deixaram de acreditar em mim. Amo todos vocês.

Aos meus maiores entusiastas. De forma especial ao Edgy Paiva, Alexandre Menezes, Márcio Braga, Henrique Landim, Fábio Leite e Cláudio Rocha, pela oportunidade e credibilidade depositada, mesmo antes de me conhecer.

Aos grandes amigos por todo apoio incondicional, ressalvas ao cunhado, irmão e às vezes pai, Ricardo Borges, devo parte do meu sucesso a você.

A minha companheira de todos os momentos, de todas as alegrias, de todas as superações, esse trabalho também é seu. Obrigado por ser meu porto seguro, obrigado pela paciência e por estar ao meu lado sempre que preciso. Virllane essa vitória é para você. Te amo.

LISTA DE ILUSTRAÇÕES

Figura 1 – Diagrama de classes
Figura 2 – Exemplo da métrica WMC
Figura 3 – Exemplo da métrica DIT
Figura 4 – Exemplo da métrica NOC
Figura 5 – Exemplo da métrica LCOM
Figura 6 – Exemplo da métrica CBO
Figura 7 – Exemplo da métrica NOA
Figura 8 – Exemplo da métrica CS
Figura 9 – Exemplo da métrica NOO
Figura 10 – Exemplo da métrica SI
Figura 11 – Visualização da estrutura de uma classe
Figura 12 – Tela principal do Jmetric
Figura 13 – Árvore com a estrutura da(s) classe(s)
Figura 14 – Calcular as métricas JMetrics
Figura 15 – Demonstração do gráfico do JMetric
Figura 16 – Resultados obtidos pela análise do JDepend por pacotes
Figura 17 – Resumo das métricas feitas pelo JDepend

SUMÁRIO

INTRODUÇÃO 7
CAPÍTULO 1 MÉTRICAS DE SOFTWARE 10
1.1 IMPORTÂNCIA DAS MEDIÇÕES...12
CAPÍTULO 2 LINGUAGEM ORIENTADA A OBJETO 14
CAPÍTULO 3 OBJETIVOS DAS MÉTRICAS DE SOFWARE 21
3.1 MÉTRICAS TRADICIONAIS...23
3.1.1 Constructive Const Model..23
3.1.2 LINHAS DE CÓDIGO..23
3.2 MEDIDA DE CIÊNCIA DO SOFTWARE...24
3.3 MÉTRICA DA COMPLEXIDADE CICLOMÁTICA...25
CAPÍTULO 4 MÉTRICAS DE SOFTWARE ORIENTADO A OBJETOS 27
4.1 MÉTRICAS SEGUNDO CHIDAMBER E KEMERER...31
4.2 MÉTRICAS SEGUNDO LORENZ E KIDD...34
CAPÍTULO 5 FERRAMENTAS SOBRE MÉTRICAS ORIENTADAS A OBJETOS.39
5.1 JMETRIC - JAVA METRICS ANALYSER...39
5.2 FERRAMENTA PARA CÁLCULO DE MÉTRICAS EM SOFTWARES ORIENTADOS A OBJETOS CODIFICADOS EM DELPHI.......................................42
5.3 MÉTRICAS PARA PROGRAMAÇÃO ORIENTADA A OBJETOS.......................42
5.4 FERRAMENTA JAVACC (JAVA COMPILER COMPILER).................................42
5.5 JDEPEND..43
5.6 JAVANCSS..45
CONSIDERAÇÕES FINAIS 46
REFERÊNCIAS 49

INTRODUÇÃO

A tecnologia nos maravilha com mudanças profundas em todos os níveis da atividade humana. A informática é a mais incrível manifestação deste fato. A era da informação tomou conta de nossas casas, escritórios e salas de aula e também mudou o perfil comportamental das organizações e das pessoas.

Em um ambiente de mudança e competitivo cada vez mais difícil, a gestão apropriada da informação assume uma importância determinante no processo de tomada de decisão nas organizações (NOGUEIRA, 2009). Alcançar um alto nível de qualidade de serviço ou produto é o objetivo da maioria das organizações. Nos dias atuais não é mais admissível entregar produtos com baixa qualidade e reparar as deficiências e os problemas depois que os produtos foram entregues (SOMMERVILLE, 2003).

No histórico da Engenharia de Software, Pressman (2002), relata a "Crise do Software" onde se apresentam números que divulgam a problemática da não finalização de projetos de software. Para tornar genérico o termo, acontece quando o software não satisfaz seus envolvidos, sejam usuários e/ou clientes, desenvolvedores ou empresa (REZENDE, 1999).

O gerenciamento de projetos é a utilização de habilidades, conhecimentos e técnicas para esquematizar atividades que visem exceder ou atingir as necessidades e expectativas das partes envolvidas, relacionadas ao projeto (PMBOK, 2000). O ato de atingir ou exceder as expectativas e necessidades das partes envolvidas envolve de forma invariável o equilíbrio entre demandas concorrentes: escopo, prazo, custo e qualidade (NOGUEIRA, 2009).

O insucesso de grandes projetos de software, na década de 60 e no começo da década de 70, foi a primeira indicação das dificuldades de gerenciamento dos projetos de software (NOGUEIRA, 2009). O software era entregue com muito atraso, custava muitas vezes mais do que previam as estimativas originais, não era confiável e, muitas vezes, exibia características arriscadas de desempenho

(BROOKS, 1975). As alterações descontroladas em software normalmente levam ao caos e/ou crise de software (REZENDE, 1999).

Tendo como base as melhores práticas da engenharia de software, processos alinhados às normas de qualidade de software e definidos de forma a se adaptar às características de software, através desse modelo, pretende-se sistematizar o processo de desenvolvimento de sistemas de informação (NOGUEIRA, 2004).

Druker (1975) associa o impacto do risco a produção de software, uma vez que já que é inútil tentar extinguir os riscos e questionável tentar minimizá-los, o primordial é que os riscos considerados sejam os coretos. Antes que se possa identificar os “riscos certos”, que ocorrerão durante um projeto de software, é importante identificar os demais que são óbvios tanto para gerentes quanto para os profissionais envolvidos (PRESSMAN, 2002).

Há várias medidas de garantia de qualidade essenciais para o sucesso de qualquer tipo de aplicação de software. Entre elas, uma das mais simples e menos onerosa é a medição de software. Nessa acepção, a aferição de software auxilia a tomada de decisão; pois por meio de dados quantitativos, é capaz de informar que aspectos do produto atendem ou não ao padrão de qualidade requerido, além de permitir a avaliação dos benefícios de ferramentas e métodos novos de engenharia de software, o aperfeiçoamento e entendimento do processo de produção, a análise do retorno do investimento e tornar o gerenciamento de projetos baseado em fatos e não em suposições, por exemplo.

Métricas é um tema bastante debatido em toda a comunidade de engenharia de software. Sua importância está no fato de que permite um acompanhamento mais minucioso e preciso das atividades dos projetos. Em particular, métricas podem ser uma excelente ferramenta no auxílio ao controle de projetos de desenvolvimento de software empregando metodologias ágeis.

Ao estimar as atividades do ciclo de vida do software, por agrupamentos de atividades ou individualmente, torna-se provável compará-las, medi-las e também fazer análises sobre os resultados reais em relação aos esperados. Como cada

característica (medida) é efetuada sempre com mesma abordagem e com o mesmo peso é possível comparar os esforços na realização das tarefas ou atividades.

A existência de divergências nos valores estimados em relação aos reais, obtidas a partir de análises sobre a produtividade das equipes, do tipo de software e da tecnologia aplicada, possibilitam o ajustamento nos índices ou nos pesos aplicados e com isso aumentar-se tanto a qualidade, como a produtividade, através de ações de melhoria que podem ser satisfeitas. Segundo Pressmann (2000), a utilização de métricas padronizadas é de fundamental importância para o êxito destas estimativas, desconsiderando qual métrica seja aplicada.

Desenvolvendo o software com qualidade, cria-se a legítima possibilidade de extrair de um sistema, informações importantes que venham não só para facilitar com a decisão, mas para ser um fator de excelência empresarial, permitindo novos negócios, sobrevivência e permanência num mercado influente. Para isso, é de extrema importância identificar, analisar os riscos que ameaçam o sucesso do projeto assim como gerenciá-los para que se possam atingir os objetivos empresariais (NOGUEIRA, 2009).

Este trabalho de conclusão de curso pretende apresentar a análise da complexidade e qualidade do código Orientado a Objetos, assim como auxiliar no entendimento dos benefícios das métricas. Aborda também conceito de programação Orientado a Objetos e pesquisar as métricas de software e a origem do Sistema Métrico, assim como sua importância e seus objetivos, sendo fortes aliadas no acompanhamento, estimativa e apoio em decisões durante a fabricação de produtos de software, tendo em vista uma melhor qualidade de todo este processo.

Para atender aos objetivos mencionados, foi utilizado neste trabalho uma revisão bibliográfica, contemplando os principais autores e teorias sobre o assunto, bem como, uma análise descritiva, uma vez que o pesquisador não interferiu na realidade. O presente estudo encontra-se estruturado em cinco capítulos, após esta secção introdutória, contemplando uma breve contextualização do assunto, justificativa do tema e objetivos da investigação.

CAPÍTULO 1 MÉTRICAS DE SOFTWARE

Os fatores de qualidade de um software não são, totalmente, satisfatórios para garantir o seu sucesso. Métricas técnicas comportam uma avaliação quantitativa. Levando em consideração que as medidas da Engenharia de Software não são absolutas, precisam-se criar medidas indiretas que induzem a métricas que embasam indicações de propriedade de alguma representação. Como métricas e medidas são relativas, elas estão abertas a discussões.

As métricas surgiram da execução objetiva de avaliação para quantificar indicadores sobre o processo de construção de um sistema, sendo adotados a partir de 1970. Há quatro tendências desta área (MOLLER; PAULISH, 1993) que são: **Medida da complexidade do código:** originada por volta de 1970, os conjuntos métricos foram fáceis de alcançar desde que fossem calculados pelo próprio código automatizado;

Estimativa do custo de um projeto de software: esta técnica foi criada em meados de 1970, estimando o tempo e o trabalho consumido para se desenvolver um software, fundamentando-se além de outras variáveis, no número de linhas de código empregado na implementação do sistema; **Garantia da qualidade do software:** a melhora destas técnicas teve uma repercussão maior entre os anos de 1970 e 1980, produzindo-se destaque à identificação de informações faltantes, entre as etapas do ciclo de vida do software;

Processo de desenvolvimento do software: o projeto de software recebeu complexidade e importância, sendo que a necessidade de se administrar este processo foi crucial. O processo incluiu a definição do ciclo de vida do software através do significado da seqüência das fases do desenvolvimento, oferecendo mais destaque ao controle e gerenciamento de recursos deste projeto.

Após do surgimento destas tendências, os desenvolvedores de sistema começaram a utilizar métricas no intuito de adequar o processo de desenvolvimento de software.

A definição de processo de software se fundamenta no conceito generalizado de processo, que pode ser conceituado como uma sequência de estados de um sistema que se altera (SPINOLA, 1998).

A maior parte (NOGUEIRA, 2009) dos estudos empíricos em Engenharia de Software utiliza uma combinação entre métodos qualitativos e quantitativos. Uma das maneiras mais comuns de ajustar ambas as estratégias é a codificação, que consiste na extração de valores quantitativos dos dados qualitativos para permitir o tratamento estatístico ou outra abordagem quantitativa (HÜBNER, 2003). Segundo Arthur (1994), para gerenciar a qualidade e a produtividade, é imprescindível saber se ambas estão melhorando ou piorando.

Conforme Bergamo (2000) existem raras métricas de concordância geral para as características de software. De acordo com Pressman (1995), há métricas de processo para analise, projeto, código-fonte, teste e manutenção. Métricas de software são úteis para apresentar medidas, de preferência quantitativas, que reflitam propriedades específicas de processos e de produtos em construção, podendo ser usadas em diversas dimensões, como esforço, complexidade, tamanho, dentre outras.

Para Fernandes (1995), métricas são definidas como métodos de definir quantitativamente a extensão em que o processo, o projeto e o produto de software têm certos atributos. A coleta apropriada de métricas, com suas respectivas análises, pode ajudar o Engenheiro de Software na tomada de decisões durante o desenvolvimento de um projeto, buscando a melhoria da qualidade do processo e do produto em construção.

Para gerenciar qualidade e produtividade é indispensável saber se ambas estão melhorando ou piorando. Isto sugere a necessidade de métricas que indiquem as inclinações do desenvolvimento de um sistema. Deste modo, a gerência de um produto de software alcança um determinado estado de qualidade e precisão se existirem medidas que tornem possível a administração através dos aspectos do sistema. A métrica de software é uma medida de características do sistema que

podem ser definidas como meios para definir quantitativamente a dimensão em que o produto, a sequência e o projeto de software têm determinadas características.

Neste contexto, o processo de software estará sob controle se for tomada uma política documentação e coleta de dados durante a evolução do projeto. A finalidade da mensuração é equipar engenheiros e gerentes de produtos com um grupo de informações palpáveis para se gerenciar, projetar, controlar, estimar e melhorar os projetos com maior eficiência (TONINI, 2004).

Conforme Côrtes e Chiossi (2001), quando são calculadas métricas, tem a intenção de obter dados que irão proporcionar alternativas para uma melhoria. Este é um dos principais objetivos da métrica de software, o estudo dos fatores que influenciam o rendimento através da qualificação dos projetos de desenvolvimento de software.

As métricas podem aferir todos os estágios do desenvolvimento e vários aspectos do produto. Métricas ajudam a compreender o processo utilizado para a implementação de um sistema. De acordo com Pressman (1995), o processo é medido com a intenção de melhorá-lo e o produto é mensurado com o propósito de ampliar sua qualidade.

Medidas são necessárias para analisar o rendimento e a qualidade do processo de manutenção e desenvolvimento do produto de software gerado. Assim, “as empresas devem estabelecer métricas apropriadas e manter procedimentos para monitorar e medir as características de suas operações que possam causar impacto significativo na qualidade de seus produtos” (CÔRTES, 2001).

1.1. IMPORTÂNCIA DAS MEDIÇÕES

De acordo com Tonini (2004), quando não é conhecida a complexidade de um software não se consegue ter ciência do caminho a seguir e também o que fazer

para solucionar uma dificuldade. Um dos modos de se controlar o desenvolvimento de um sistema é o emprego da medição de software. As métricas podem medir cada estágio do desenvolvimento e muitos aspectos do produto. Métricas auxiliam a compreensão do processo utilizado para a implementação de um sistema (JACOBSON *et al*, 1992).

CAPÍTULO 2 LINGUAGEM ORIENTADA A OBJETO

A expressão Orientação a Objetos (OO) pressupõe uma organização de software em termos de conjunto de objetos discretos incorporando comportamento e estrutura próprios. Esta abordagem de organização é fundamentalmente diferente do desenvolvimento tradicional de software, onde rotinas e estrutura de dados são produzidas apenas de forma fracamente acopladas.

A Orientação a Objetos é um modelo que representa todo um conceito para fabricação de sistemas; estruturas que se conhecem do cotidiano e sobre as quais se possui maior compreensão (DALL'OGLIO, 2007). Ao invés de construir um sistema composto por um conjunto de variáveis e procedimentos nem sempre agrupadas de acordo com o contexto, como se fazia em linguagens estruturadas, tais como Cobol, Clipper e Pascal, na Orientação a Objetos usa-se uma ótica mais parecida com o mundo real.

Softwares Orientados a Objeto são baseados na interação e composição entre várias unidades chamadas objetos. Objeto é uma instância de uma classe, é capaz de comportar estados através de seus atributos e reagir a mensagens enviadas a ele, assim como se relacionar e enviar mensagens a outros objetos.

De acordo com Deitel e Deitel (2004), pode-se definir objetos como sendo componentes de software que modelam componentes do mundo real conforme com seus comportamentos e atributos. Por exemplo, se considera como objeto o “sino”, ele poderá ter como atributos sua cor, forma e tamanho, e como comportamento, o fato de tocar.

Ainda para eles, os objetos com características comuns podem ser representados por uma classe. As classes são, portanto, os arquivos .java (para linguagem JAVA) que compõem um sistema. Elas mantém, além de seus objetos, a implementação de métodos e atributos. Os métodos são os responsáveis por desempenhar tarefas sobre os objetos definidos no conjunto de classes do sistema.

Outro significado para objeto seria a de uma "entidade" ativa dotada de certas características que o tornam "inteligente", capaz de tomar certas decisões quando precisamente solicitado. Para Price (1997) a definição mais formal para objeto é uma unidade dinâmica, composta por um estado interno privativo (estrutura de dados) e um comportamento (conjunto de operações). E nesta situação, conforme (PRICE, 1997), um objeto em particular é como um processador com memória própria e independente de outros objetos.

Em termos de implementação, objeto é um bloco de dados privados envolvidos por código, de maneira que a permissão a ele só pode ser realizado sob condições especiais. Todo o comportamento dessa "entidade" encapsulado é proposto através de rotinas que gerenciam seus dados, sendo que o seu estado corrente está em seus próprios dados; cada objeto tem suas próprias características, moldadas a partir de uma matriz. Formalmente, para ser considerada uma linguagem OO, esta precisa programar quatro conceitos básicos: abstração, encapsulamento, herança e polimorfismo.

Classes representam um conjunto de objetos com características afins. Uma classe define o comportamento dos objetos, por meio de seus métodos, e quais estados ele é capaz de manter, através de atributos. Mensagem é uma chamada a um objeto para invocar um de seus métodos, acionando um procedimento descrito por sua classe, podendo alterar o valor de um ou mais atributos, alterando o estado de um objeto (BATES, 2009).

Método é uma rotina que é executada por um objeto ao receber uma mensagem. Ele pode ser público, quando é utilizado por vários aplicativos (softwares) ou privados, quando utilizado ou criado pelo aplicativo (software) do projeto (BATES, 2009).

Atributo é o componente que define a estrutura de uma classe. Os atributos são conhecidos também como variáveis de classe, e podem ser divididos em dois tipos básicos: atributos de classe e de instância. Os valores dos atributos de instância

definem o estado de cada objeto. Um atributo de classe tem um estado que é compartilhado por todos os objetos de uma classe (BATES, 2009).

A ação de programar em linguagens OO é denominada de Programação Orientada a Objetos (POO). Fazer esse tipo de programação proporciona facilidades para implementação do projeto, uma vez que permite a reutilização de código e o uso de conceitos como encapsulamento de dados, polimorfismo e herança.

As técnicas baseadas em objetos tendem a tornar mais simples o projeto de softwares complexos. Conforme Amber (1998), as organizações optam a Orientação a Objetos (OO) porque querem dar às suas aplicações maior qualidade, as quais querem programar sistemas seguros, com um menor tempo e menor custo.

Para Rocha (2001), o desenvolvimento de software com a utilização do paradigma de OO aparece como uma probabilidade para a melhoria da produtividade e qualidade, pois permite modelar o problema em termos de objetos capaz de minimizar a distância entre o problema do mundo real e sua abstração.

De acordo com Rosenberg (1998), a OO requer uma abordagem diferente tanto na implementação do projeto quanto no desenvolvimento do mesmo, além disso, as métricas de software, visto que utiliza objetos e não blocos de construção fundamentais.

Segundo Rocha (2001), dadas às diferenças entre as duas visões, é comum notar que as métricas de software desenvolvidas para serem aplicadas aos métodos tradicionais de desenvolvimento não são mapeadas com facilidade para os conceitos OO. Para tanto, faz-se necessária a exposição de algumas definições para nortear o presente trabalho e facilitar o desenvolvimento desta investigação, apresentando alguns conceitos básicos de Orientação a Objetos.

O **Encapsulamento** é o alicerce de toda a abordagem da Programação Orientada ao Objeto; devido contribuir essencialmente para minimizar os malefícios ocasionados pela interferência externa sobre os dados. Continuando desse

princípio, toda e qualquer transação feita com esses dados só pode ser feita pelos procedimentos colocados dentro desse objeto, através do envio de mensagens.

Desta forma, pode-se dizer que um dado está encapsulado quando envolvido por código de forma que só é visível na rotina onde foi feito; acontece o mesmo com uma rotina, que sendo encapsulada, suas operações internas são invisíveis às outras rotinas. E até mesmo em linguagens não OO, como no caso do Clipper 5.xx. Segundo Ferreira e Jarabeck (1995), pode-se observar um específico encapsulamento nas rotinas em que as variáveis são declaradas como LOCAL.

Nesses casos tais variáveis são visíveis apenas dentro dessas rotinas aonde foram declaradas, o que dá ao programador certa segurança quanto aos acessos indevidos por parte de outras rotinas, o que não acontece com variáveis PRIVATE ou PUBLIC, no contexto dessa linguagem. No encapsulamento pode-se imaginar a sua utilidade pensando em um vídeo cassete, onde se tem os botões de liga-desliga, para traz, para frente, entre outros.

Estes botões realizam uma série de operações existentes no aparelho, onde são executadas pelos componentes existentes dentro do aparelho (transistores, motores, cabos) Não interessa ao usuário saber como é o funcionamento interno do equipamento; esta informação só é relevante para os projetistas do aparelho. As informações relacionadas ao operador do equipamento são as existentes no meio externo (controle remoto, botões) que ativam as operações internas do equipamento.

Deste modo o aparelho de vídeo cassete pode evoluir com os avanços tecnológicos, e as pessoas que o utilizam continuam sabendo empregar o equipamento, sem a necessidade de um novo treinamento. Na área de software ocorre o mesmo: as classes podem continuar evoluindo, com aumento de tecnologia, e os programas que usam essas classes continuam compatíveis. Isto acontece porque a esses programas não interessa saber como é o funcionamento interno da classe e sim sua funcionalidade, para que ele possa executar, a medida que ela evolui, novas funções colocadas à sua disposição.

Encapsulamento consiste também na separação de aspectos externos e internos de um objeto. Este mecanismo é usado vastamente para impedir o acesso direto ao estado de um objeto, disponibilizando externamente apenas os métodos que modifiquem estes estados.

O significado de **polimorfismo**, quer dizer "várias formas"; contudo, na Informática, e em particular no mundo da POO, é definido como sendo um código que possui "muitos comportamentos" ou que produz "muitos comportamentos"; em outras palavras, é um código que pode ser utilizado em várias classes de objetos. Pode-se sugerir que a operação em questão conserva seu comportamento transparente para quaisquer tipos de argumentos; a mesma mensagem é enviada a objetos de classes distintas e eles podem reagir de maneiras diferentes.

Um método polimórfico é aquele que pode ser aplicado em várias classes de objetos sem que exista qualquer inconveniente. É o caso, por exemplo, do método *Clear* em Delphi®, que pode ser aplicado tanto à classe *TListBox* como à classe *TEdit*; nas duas situações o conteúdo desse objetos são limpos, mesmo pertencendo à classes distintas (LEITE, 1998).

Para Ferreira e Jarabeck (1995), um exemplo bastante didático para o polimorfismo é dado por um simples moedor de carne. Esse equipamento tem a papel de moer carne, produzindo carne moída para fazer bolinhos. Desta forma, não importa o tipo (classe) de carne alimentada; o resultado será sempre carne moída, não importa se de frango, de boi ou de qualquer outro tipo. Os limites impostos pelo procedimento estão no próprio objeto, definidas pelo seu fabricante e não pelo usuário.

Em polimorfismo, o tipo da referência pode ser uma superclasse com o tipo do objeto real. Quando for declarado uma variável de referência, qualquer objeto que passar no teste É-UM quanto ao tipo declarado para ela poderá ser aplicado a essa referência. Resumindo, qualquer coisa que estender o tipo declarado para a variável de referência poderá ser atribuída a ela (BATES, 2009).

Com o polimorfismo, poderá ser escrito um código que não tenha que ser alterado quando novos tipos de subclasse forem introduzidos no programa. Polimorfismo

permite que um objeto se comporte de acordo com sua classe. Deste modo, é possível se relacionar com objetos de classes diferentes, enviar mensagens iguais e deixar o objeto se comportar à definição de sua classe.

Abstração é a aptidão de ignorar detalhes de partes para enfocar a atenção em um nível mais acima de uma dificuldade. O problema é dividido em subproblemas, a seguir, outra vez em subproblemas e assim por diante, até que os problemas de forma individual fiquem suficientemente pequenos para serem fáceis. Uma vez que se solucione um dos subproblemas, não se reflete mais sobre os detalhes dessa parte, mas se trata a solução como um simples bloco de construção para o problema seguinte. Essa técnica é às vezes conhecida como dividir para conquistar.

Os princípios de modularização e abstração são usados no desenvolvimento de software. Para auxiliar a manter uma visão geral em programas complexos, busca-se identificar subcomponentes que se pode programar como identidades autônomas. Então tenta-se utilizar esses subcomponentes como se fossem partes simples sem se preocupar com suas complexidades internas.

Na programação OO, esses componentes e subcomponentes são objetos. No caso de uma construção de um carro em software, utilizando uma linguagem Orientada a Objetos, tentar-se-ia fazer o mesmo que os engenheiros da indústria automobilística fazem.

Ao invés de programar o carro em um único objeto monolítico, primeiramente se constrói objetos separados como caixa de engrenagens, motor, assento, roda e assim por diante e então monta-se o objeto carro a partir desses objetos específicos. A identificação destes tipos de objetos (e com estes, as classes) que se deve ter em um sistema de software para qualquer problema nem sempre é fácil (KÖLLING, 2008).

A **herança** proporciona definir uma classe como uma extensão de outra. A herança é um mecanismo que permite uma solução para o problema de duplicação de código. O recurso primordial dessa técnica é que declara-se os recursos comuns somente uma única vez. As classes que são relacionadas por meio de herança

formam uma hierarquia de herança. A herança pode ser usada de modo muito mais geral. Mais de duas subclasses podem herdar da mesma superclasse; e uma subclasse, por sua vez, pode ser uma superclasse para outras subclasses. Sendo assim, as classes formam uma hierarquia de heranças.

O exemplo mais comum de uma hierarquia de heranças provavelmente é a classificação de espécies utilizada pelos biólogos. Pode-se perceber que um pitbull é um cão, que é um mamífero, que é um animal. O princípio é simplório. A herança é uma técnica de abstração que nos deixa categorizar as classes de objetos sob critérios específicos e nos auxilia a especificar as características dessas classes (KÖLLING, 2008).

Herança é o recurso pelo qual uma classe pode estender outra classe, aproveitando seus comportamentos (métodos) e estados possíveis (atributos). Os desenvolvedores se beneficiarão muito da OO projetando com a herança. Poderão abolir código duplicado generalizando o comportamento comum a um grupo de classes e inserindo esse código em uma superclasse.

Então, quando precisarem alterá-lo, terão somente um local a atualizar, e a alteração repercutirá de modo instantâneo em todas as classes que herdarem esse comportamento (BATES, 2009). A herança lhe proporcionará garantir que todas as classes agrupadas sob um certo supertipo tenham todos os métodos que o supertipo tem. Nesse sentido, será definido um protocolo comum para um conjunto de classes relacionadas através da herança.

CAPÍTULO 3 OBJETIVOS DAS MÉTRICAS DE SOFWARE

Os objetivos das métricas se confundem com o da Engenharia de Software, que tem como objetivo principal a melhora da qualidade dos produtos de software e o aprimoramento da produtividade dos engenheiros de software, além do atendimento aos requisitos de eficiência e eficácia, resumindo, efetividade (MAFFEO, 1992).

Conforme Funck (1995), a utilização das métricas deve ser definida desde o início da implantação de métricas para análise de software. Existem diversas características importantes ligadas com o emprego das métricas de software. Essa escolha exige alguns pré-requisitos:

Os objetivos que se tem intenção de atingir com o uso das métricas; As métricas devem ser simples de serem utilizadas e de atender para verificar atendimentos objetivos e para auxiliar processos de tomadas de decisão; As métricas devem ser objetivas, buscando minimizar ou reduzir a influência do julgamento pessoal no cálculo, coleta e análise dos resultados. Percebe-se que a confecção do software não pode ser realizada de forma aleatória e simplista. Na verdade, deve proceder um rito uma vez que o objetivo é propiciar a melhor tomada de decisão, por exemplo.

Segundo Amber (1998), as métricas possuem objetivos definidos. O primeiro é para estimar projetos; fundamentado nas experiências anteriores pode-se utilizar métricas para estimar o esforço, o tempo e o custo de um projeto. O segundo objetivo é para selecionar as ferramentas; uma vez que elas variam conforme a necessidade de cada projeto. O terceiro objetivo é o de melhorar o processo de desenvolvimento do projeto;

Para Fernandes (1995), um dos aspectos que deve ser notado quando da implementação de iniciativas de uso de métricas é quando a sua utilidade no contexto de um projeto ou do ambiente na sua totalidade, além das categorias e

tipos de métricas, usuários das métricas, pessoas para as quais os resultados das métricas são dedicados e os seus níveis de aplicação.

A etapa de avaliação e medição exige um mecanismo para definir quais os dados que devem ser coletados e como os dados coletados devem ser interpretados (FUNCK, 1995). Neste processo é requerido um mecanismo organizado para a determinação do objetivo da medição. A definição de tal finalidade abre caminho para alguns questionamentos que definem um conjunto específico de informações (dados) a serem coletadas.

Os objetivos da avaliação e da medição são consequências das necessidades do fabricante, que podem ser a necessidade de avaliar determinada tecnologia, a necessidade de compreender melhor o uso dos recursos para melhorar a estimativa de custo, a necessidade de avaliar a qualidade do produto para poder definir sua implementação ou a necessidade de avaliar as vantagens e desvantagens de um projeto.

Sendo assim, o objetivo primordial de se utilizar medições no desenvolvimento de software é conseguir níveis cada vez maiores de qualidade, levando em consideração o projeto, o produto e o processo, visando à satisfação total dos usuários ou clientes a um custo economicamente viável.

Para Sommerville (2003), o gerenciamento de configuração (*Configuration Management* – CM) é a aplicação e desenvolvimento de padrões e procedimentos para gerenciar um produto de sistema em produção. É necessário gerenciar os sistemas em desenvolvimento porque, à medida que eles se desenvolvem, são criadas muitas versões diferentes de software.

Em uma instituição que se aplica ao desenvolvimento, seja como atividade-fim, seja como de suporte para uma empresa, existem muitos objetivos que se busca alcançar, dependendo do estágio de maturidade em que se encontram essas atividades. Vários dos objetivos definidos pela Engenharia de Software se enquadram na seguinte relação, de acordo com Feigenbaum (1986): a) Aumentar a qualidade do planejamento do projeto, b) Melhorar a satisfação dos clientes e

usuários do software, c) Aperfeiçoar a qualidade e produtividade do desenvolvimento, e d) Evoluir continuamente os métodos de gestão de projeto.

3.1. MÉTRICAS TRADICIONAIS

Serão listados alguns modelos de métricas tradicionais.

3.1.1 *Constructive Const Model*

O modelo COCOMO (*Constructive Const Model*) é avaliado a partir da quantidade (número) de linhas de código fonte produzido (FUNCK, 1995). Este modelo foi feito por Barry Boehm e resulta em estimativas de prazo, esforço, custo e tamanho da equipe para um projeto de desenvolvimento de software. O COCOMO é um conjunto de submodelos hierárquicos, que pode se dividir em submodelos básicos, intermediários ou detalhados.

3.1.2 Linhas de Código

Segundo Koscianski e Soares (2006), o modelo LOC (*Lines of Code*), é a técnica de estimativa mais velha. Ela pode ser usada para estimar o custo do software ou para definir igualdade de analogia. Existem muitas especulações e discussões sobre este modelo. Primeiro, a definição de linhas de código não é muito esclarecedora.

Um exemplo simplório seria o caso de ser colocado ou não uma linha em branco ou comentário como LOC. Alguns autores consideram estes comentários, entretanto, outros não. No caso de programas recursivos, essa técnica não funciona, porque a recursividade torna o programa mais curto. O sistema LOC é uma técnica superficial e genérica (KOSCIANSKI, 2006).

Outra dificuldade da técnica LOC é que este modelo é amplamente ligado à linguagem de programação utilizada, tornando inviável a utilização de dados históricos para projetos que não utilizam a mesma linguagem. Os benefícios do uso de LOC são (FUNCK, 1995): a) É fácil de ser obtido; b) Há farta literatura e dados sobre este sistema de métrica; e c) É utilizado por muitos modelos de estimativa de software como valor básico de entrada. Entre as desvantagens pode-se citar:

Dependência de linguagem: não é admissível comparar diretamente projetos que foram desenvolvidos em linguagens distintas. Como exemplo, pode-se medir a quantidade de tempo gasto para gerar uma instrução em uma linguagem de alto nível em comparação com uma linguagem de baixo nível;

Penalizam programas bem projetados: quando um programa é bem projetado o mesmo utiliza poucos comandos para realização de uma tarefa; ***Dificuldades de estimar no início do projeto de software:*** é provavelmente impossível estimar o LOC necessário para um sistema saindo da fase de modelagem ou da fase de levantamento de requisitos.

3.2. MEDIDA DE CIÊNCIA DO SOFTWARE

Halstead (1997) identificou a Ciência de Software - primeiramente chamada de Física do Software - como uma das primeiras amostras sobre métrica de código fundamentada num modelo de complexidade do software (SEIBT, 2001). A principal

idéia deste modelo é a compreensão de que software é um processo de manipulação mental simbólica de seus programas.

Estes símbolos podem ser especificados como operadores (em um programa executável verbos como: *DIV, IF, READ, ELSE* e os operadores propriamente ditos) ou operandos (constantes e variáveis), tendo em vista que a divisão de um programa pode ser avaliada como uma seqüência de operadores ligados a operandos.

Conforme Shepperd (1993), a ciência do software induziu consideravelmente o interesse das pessoas em meados de 1970 por ser uma novidade na metrificação do software. Além disso, as entradas básicas do software são todas extraídas com facilidade. Depois da empolgação inicial da ciência do software, foram encontrados graves problemas.

As causas podem ser relatadas em função da dificuldade que os pesquisadores acharam na comparação dos trabalhos e progresso da métrica. Outra causa seria a não associação correta entre o esforço exigido para manipulação do programa e o tempo requerido para realizar o programa e também por tratar um sistema como simplesmente um módulo.

3.3. MÉTRICA DA COMPLEXIDADE CICLOMÁTICA

Este modelo foi sugerido por McCabe (1996), que estava particularmente com interesse em encontrar a quantidade de caminhos criada pelos fluxos de controle em um módulo do software, desde que fosse relacionada à dificuldade de teste e no melhor modo de dividir software em módulos (SHEPPERD, 1993).

Para Jacobson (1992), a idéia é desenhar num grafo a sequência que um programa pode seguir com todos os caminhos possíveis. A complexidade medida fornecerá

um número assinalando o quão complexo é um programa (ou sequência). Para permitir o cálculo desta métrica, os programas são inicialmente representados por grafos dirigidos representando o fluxo de controle. De um grafo G, pode ser retirada a Complexidade Ciclomática v(G). A quantidade de caminhos em um grafo pode ser dada como: o grupo mínimo de fluxos os quais podem ser usados para a fabricação de outros caminhos por meio do grafo.

A visão de forma simplificada da métrica de McCabe (1996) pode ser questionada em diversos pontos. Primeiramente, ele tinha em especial uma preocupação com os programas feitos em Fortran, onde o mapeamento do código-fonte, para um grafo de fluxo do programa era bem definido, sendo que isto não seria o caso de outras linguagens como Ada. Portanto, a medição não é sensível à complexidade, auxiliando assim na construção de declarações de sequência lineares.

A Complexidade Ciclomática é sensível à quantidade de sub-rotinas dentro de um programa, por esta razão, McCabe (1996) recomenda que este aspecto seja abordado como componentes sem relação dentro de um grafo de controle (SHEPPERD, 1995). Neste ponto teria um resultado significante, pois aumentaria a complexidade do programa inteiramente, tendo em vista que ele é dividido em muitos módulos que se imagina serem simples.

CAPÍTULO 4 MÉTRICAS DE SOFTWARE ORIENTADO A OBJETOS

Várias métricas já foram produzidas para gerações passadas de tecnologia e, em diversos casos, são utilizadas até para desenvolvimento OO, porém não são muito lógicos, devido à diferença com sistemas tradicionais é enorme (ROCHA; MOLDONADO; WEBER, 2001).

Há inúmeras sugestões para métricas OO que consideram as características básicas e interações do sistema como: quantidade de métodos; número de classes; linhas de código por método; profundidade máxima da hierarquia de classes; a relação existente entre métodos privados e públicos; entre outros. Estas métricas baseiam-se na análise detalhada do projeto.

As métricas OO podem ser divididas em duas categorias: medidas relacionadas com produtos e relacionadas com processos (JACOBSON, 1992). As métricas relacionadas com processo são usadas para medir o processo e o *status* do processo de desenvolvimento do sistema, consistem em mensurar coisas tais como: números de falhas encontradas durante o processo de testes ou cronogramas. De acordo com Jacobson (1992), para aprender a administrar e manipular um processo de desenvolvimento OO é essencial iniciar a coleta de dados destas medições tão sistematicamente quanto possível.

Algumas medidas tradicionais de produtos podem ser utilizadas para algumas soluções OO (JACOBSON, 1992). Porém, a métrica mais comum, linhas de código, não é interessante para sistemas OO, pois às vezes a menor quantidade de código escrito é o mais reutilizado e, na maioria vezes dá maior qualidade ao produto.

A seguir serão listadas algumas métricas para os softwares OO. Estas métricas estão relacionadas em três categorias: métricas de projeto, análise e construção. Estas medidas podem ser utilizadas para facilitar a melhoria dos esforços de desenvolvimento (AMBER, 1998). Elas podem identificar áreas com problemas na aplicação antes que elas apareçam como um erro percebido pelo usuário. As

métricas de construção e projeto além de mensurar aspectos significantes do sistema, são fáceis de automatizar, tornando-as mais fáceis de coletar.

Proporciona muito sentido adicionar um peso às métricas das classes para produzir uma medida de qualidade e complexidade do sistema. Quase todas as medidas examinam atributos em termos dos conceitos de OO como polimorfismo, herança e encapsulamento. Para tanto, seria necessário coletar um número significativo de contagens; portanto, seria requerido tomar valores de vários projetos e dimensioná-los selecionando as classes, os métodos e os atributos desejáveis para medir a complexidade e o tamanho de um software novo.

As métricas usadas para quantificar as características que abordam a qualidade de um software podem ser obtidas por meio da avaliação de aspectos como encapsulamento, polimorfismo, coesão, acoplamento e complexidade. Na Métrica de Encapsulamento encontra-se o fator de atributos escondidos, que sugere, em um índice que vai de 0 (não utilização) a 1(máximo de uso), a que quantidade os atributos são limitados exclusivamente a classe, não sendo possível o acesso a outras classes, pois só é acessível pelos métodos públicos e o fator de métodos ocultos que avalia o número de métodos acessíveis por outras classes.

Consoante a Métrica de Polimorfismo, tem-se a distinção entre o polimorfismo. O polimorfismo se refere a métodos com a assinatura iguais em classes herdadas. O polimorfismo puro ou sobrecarga em classes isoladas afere a quantidade de métodos com assinaturas diferentes, porém com o mesmo nome. Em relação ao polimorfismo estático nos ancestrais identifica se foi implementado algum método com mesmo nome, mas com tipos ou argumentos diferentes na classe de suas ancestrais.

Na Métrica de Coesão são verificadas as características reais relacionadas à classe e dá auxílio na abstração do seu tipo. Trata de forma básica a falta de coesão nos métodos que é descrito por Pressman (1995) como o número de métodos que tem acesso a um ou mais dos mesmos atributos.

Métrica de Acoplamento: propriedade que se refere ao relacionamento entre as classes. A busca de uma dependência baixa entre módulos auxilia a reuso dos módulos em outras aplicações e ameniza a implementação de alterações tanto no desenvolvimento quanto na manutenção. O acoplamento entre as classes acontece pela interação entre elas. Pressman (1995) afirma que à medida que o valor do fator de acoplamento cresce a complexidade do software também aumenta, consequentemente, a manutenibilidade, a inteligibilidade e o potencial da reutilização podem piorar.

Métrica de Complexidade: avalia aspectos que tem referência ao grau de compreensão do software na sua totalidade. Isso é um fator importantíssimo no momento de fazer uma manutenção no caso da ausência ou má documentação. São analisados fatores como número e tamanho de argumentos nos métodos, número de métodos nas classes, profundidade da arvore de herança, fator de herança de métodos e atributos, além disso, a referência a subclasses.

Métodos Ponderados por Classe (*Weighted Methods per Class* ou WMC): mede uma classe num sistema orientado a objetos, pela sua complexidade. Definida para cada classe como o somatório ponderado de todos os seus métodos (SHYAM, 1994). Comumente é utilizado *v(G)* como fator de peso, então WMC pode ser calculada como ?*ci*, onde *ci* é a Complexidade Ciclomática do *i*-ésimo método de uma mesma classe;

Complexidade Ciclomática de McCabe (*v(G)*): verifica a quantidade de lógica de decisão num módulo de software único (THOMAS, 1994). Em um sistema orientado a objetos, um método é um modulo. Um Grafo de controle de fluxo apresenta a estrutura lógica de um algoritmo através de arestas e vértices.

Os vértices representam instruções ou expressões computacionais (como laços, atribuições ou condicionais), enquanto as arestas representam a transferência do controle entre os vértices (ARTHUR, 1996). A Complexidade Ciclomática é determinada para cada módulo como $e - n + 2$, onde e e n são o número de arestas e vértices do grafo de controle de fluxo, respectivamente;

Falta de Coesão dos Métodos (*Lack of Cohesion of Methods* ou LCOM): mede a coesão de uma classe, além de ser calculada por meio do método de Henderson-Sellers (BRIAN, 1996). Se m(A) é o número de métodos que acessam o atributo A, LCOM é calculada como a média de m(A) para todos os atributos, subtraindo a quantidade de métodos m e dividindo o resultado por (1 - m). Um valor baixo aponta uma classe coesa, entretanto um valor próximo de 1 indica falta de coesão;

Número de Filhos (*Number of Children* ou NOC): o número total de filhos adjacentes de uma classe; Profundidade da árvore de Herança (*Depth of Inheritance Tree* ou DIT): a extensão do maior caminho a partir de uma classe até a classe-base da hierarquia; Acoplamento Aferente (*Afferent Coupling* ou AC): a quantidade total de classes de fora de um pacote que dependem de classes pertencentes ao pacote. Quando calculada no nível da classe, essa medida é conhecida como Fan-in da classe;

Acoplamento Eferente (*Efferent Coupling* ou EC): o número total de classes em um pacote que dependem de classes de fora do pacote. Quando calculada no nível da classe, essa medida também é chamada como Fan-out da classe, ou como CBO (Coupling Between Objects ou Acoplamento entre Objetos) na família de métricas CK (Chidamber-Kemerer) (BRIAN, 1996).

A Figura 1 evidencia um exemplo de diagrama de classes que desenha algumas medidas explicadas a seguir. Este diagrama de classes estabelece a criação de pessoas: uma pessoa pode ser do tipo funcionário ou cliente, por sua vez o funcionário pode ser do tipo diarista, horista ou mensalista; um funcionário tem um cargo e deve estar num departamento.

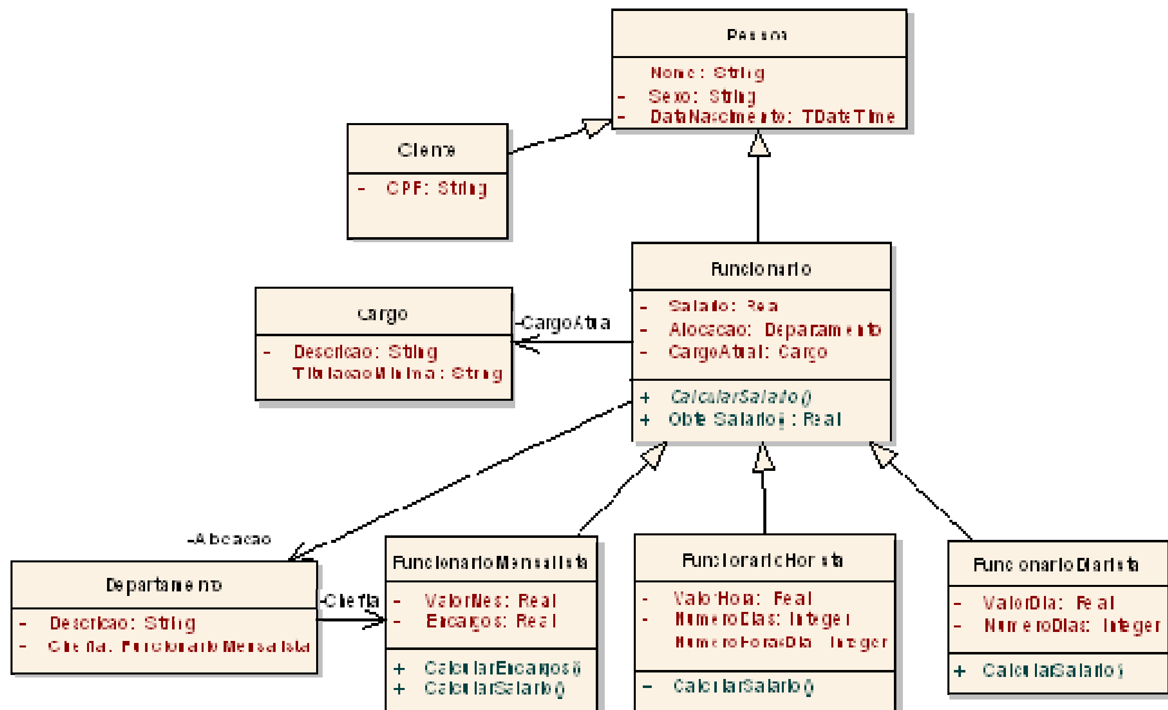

Figura 1. Diagrama de classes

4.1 MÉTRICAS SEGUNDO CHIDAMBER E KEMERER

Para Pressman (2005), a mais largamente referenciada dentre as métricas orientadas a objetos é o conjunto de métricas criado por Chidamber e Kemerer (1994). Este conjunto tem como benefício medir as características de classes individuais como tamanho e número de métodos, interações com outras classes, encapsulamento e herança, dentre outras. Como os requisitos funcionais neste nível são serviços técnicos e os parâmetros de projeto são variáveis ou objetos, se faz necessário utilizar uma métrica que contemple características de classes, métodos e atributos.

Chidamber e Kemerer (1994) propuseram seis métricas para o cálculo de complexidade de sistema OO. As métricas denominadas de CK são excelentes referências para análise quantitativa, tendo como objetivo a concentração de testes

em classes que possivelmente contêm maior número de defeitos. Em seguida uma breve descrição de cada métrica:

WMC: cálculo do número de serviços por classe. Um elevado WMC (*Weighted methods per class*) demonstra que a classe tende a se tornar específica e seus serviços têm características que atendem a necessidades individuais, diminuindo sua reutilização. A quantidade de serviços mostra ainda qual o nível de esforço deve ser gastado para o teste da complexidade da classe (Figura 2);

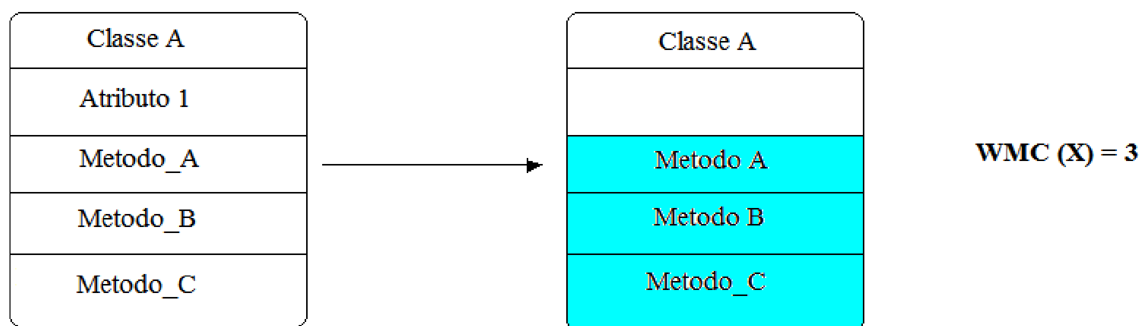

Figura 2. Exemplo da métrica WMC

DIT: é o número máximo de superclasses distribuídas hierarquicamente acima da classe analisada. Um DIT (*Depth of the inheritance*) alto mostra que várias das características da classe foram adquiridas por herança e são comuns a outras classes. Isso demonstra que as superclasses contêm um elevado nível de abstração, o que quer dizer que elas estão provavelmente preparadas para possibilitar uma boa reutilização. Em oposição, DIT pode indicar que a classe herda muitos serviços, aumentando a sua complexidade (Figura 3);

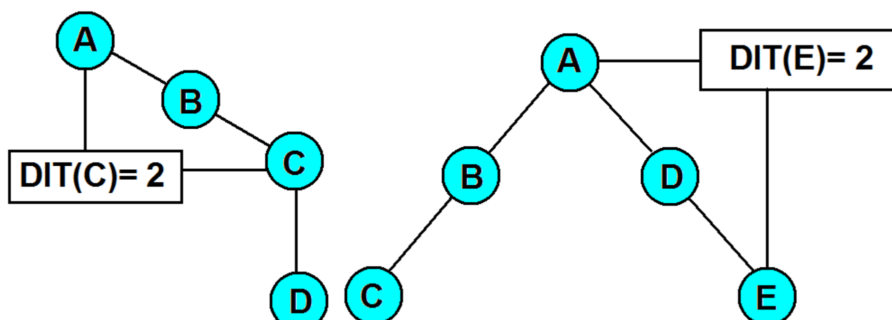

Figura 3. Exemplo da métrica DIT

NOC: quantidade de subclasses posicionadas em seguida abaixo da superclasse em questão ou filhos diretos. Um NOC (*Number of children*) elevado indica um nível baixo de abstração, pois uma superclasse com muitos filhos tem tendência a conter poucas características comuns a todas as subclasses. Um alto número de filhos diretos também pode indicar problemas estruturais, devido o fato que as subclasses podem não se encaixar a abstração implícita da classe pai (Figura 4);

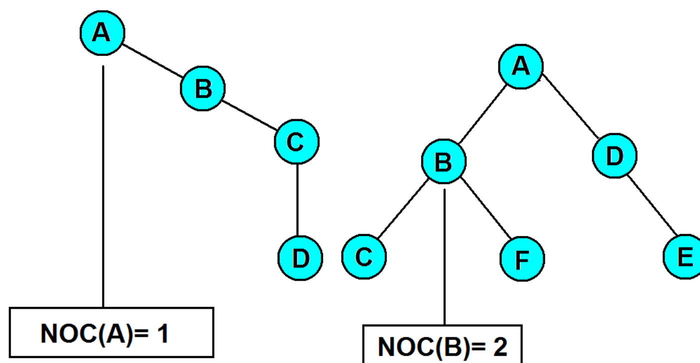

Figura 4. Exemplo da métrica NOC

LCOM: quantidade de acesso a um ou mais atributos em comum pelos serviços da própria classe. Deste modo, os serviços tornam-se ligados pelos atributos, podendo indicar que não foram bem projetados, pois mostram baixa coesão. Uma das principais características do software OO é apresentar uma coesão alta nos métodos, o que dá garantia que esses exerçam sua função corretamente. Ou seja, é importantíssimo manter o LCOM (*Lack of cohesionin methods*) da classe baixo (Figura 5);

Seja C uma classe com 4 métodos e com os seguintes conjuntos de atributos utilizados por método:

- I1 = { a, b, x}
- I2 = {a, e, f, g}
- I3 = {c, x}
-I4 = {v, u, w}

Para o cálculo de LCOM é necessário obter as interseções entre os pares.

Figura 5. Exemplo da métrica LCOM

RFC: (*Response for a class*) representa a capacidade de resposta que a classe tem ao receber mensagens de seus objetos. Uma alta capacidade de resposta exige

uma estrutura de classe projetada para acatar a essa particularidade gerando uma maior complexidade, tornando necessário um esforço de teste maior.

CBO: aponta qual é o nível de acoplamento entre as classes da aplicação. Quanto mais alta a ligação entre elas, menor a possibilidade de reuso, pois a classe torna-se dependente de outras classes para cumprir suas obrigações. Portanto o CBO (*Coupling between object classes*) está absolutamente legado ao nível de reaproveitamento. Um elevado acoplamento sugere uma baixa independência de classe, o que aumenta espantosamente a complexidade e, em consequência, o esforço de teste (Figura 6);

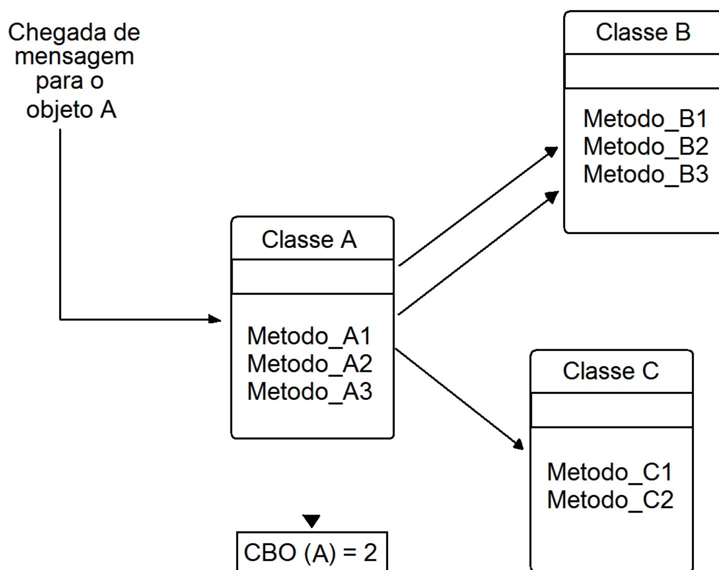

Figura 6. Exemplo da métrica CBO

4.2 MÉTRICAS SEGUNDO LORENZ E KIDD

Uma proposta distinta de métricas OO foi feita por Lorenz e Kidd (1994) tem como base o cálculo quantitativo de alguns aspectos fundamentais da OO, como os serviços e atributos, herança, acoplamento e coesão. Não divergindo das métricas de CK (Chidamber e Kemerer) no foco, mas em sua metodologia de calculo. A seguir segue a descrição de cada métrica definida:

NOA: se a classe tem um elevado número de atributos e operações privados, ela fica muito específica, reduzindo as possibilidades de reutilização. Pode-se dizer que um alto NOA (Number of operation added by subclass) pode indicar uma falha de modelo. Várias particularidades demonstram que a classe não está bem posicionada na hierarquia, já que suas características essenciais deveriam estar implícitas nos seus ancestrais (Figura 7);

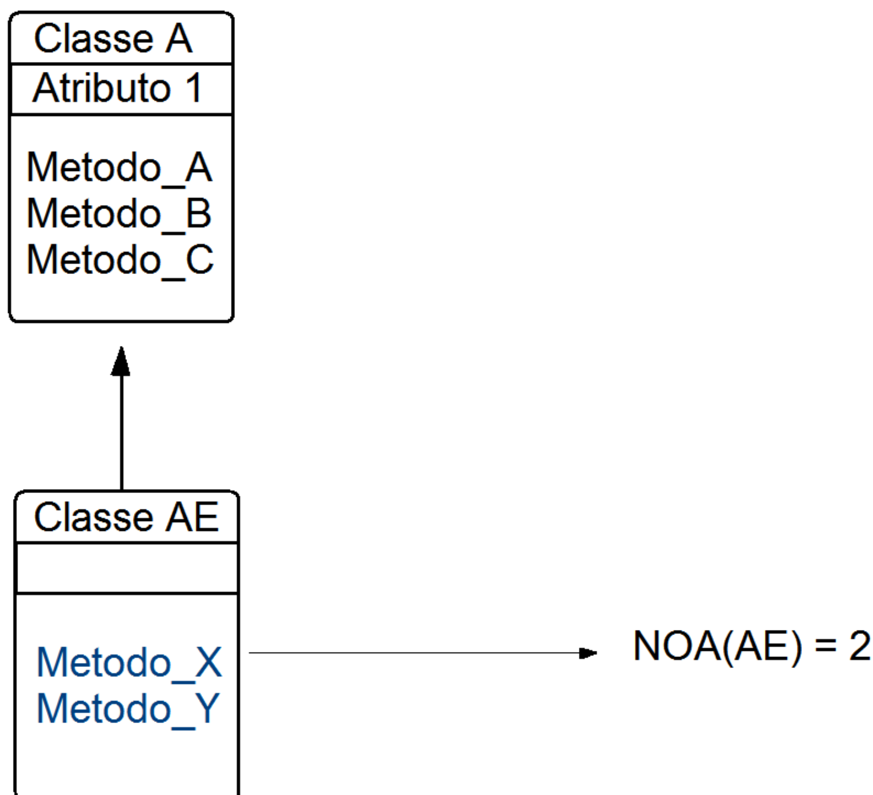

Figura 7. Exemplo da métrica NOA

CS: quantidade de serviços e atributos locais e herdados de superclasses. Os atributos públicos e serviços das classes localizadas hierarquicamente superior e os da própria classe em questão compõe o CS (*Class size*). Um elevado CS torna a classe muito específica, pois sua estrutura atende a particularidades, o que reduz a reutilização, exigindo ainda um esforço maior de testes, já que a classe fica mais complexa (Figura 8);

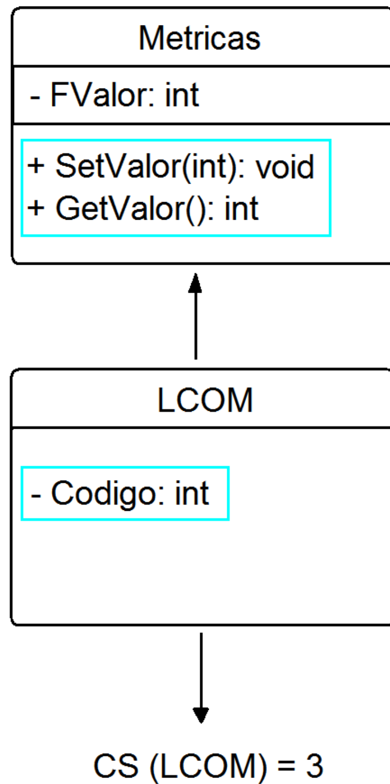

Figura 8. Exemplo da métrica CS

NOO: os métodos declarados nas superclasses são herdados pelas subclasses, mas, quando esses não atendem à necessidade individual da subclasse, podem ser redefinidos, ferindo assim a abstração implícita na superclasse. Um grande índice de NOO (*Number of operations overriden by a subclass*) sugere um problema estrutural. Se várias subclasses têm serviços redefinidos, as subclasses provavelmente estão hierarquicamente mal projetadas (Figura 9);

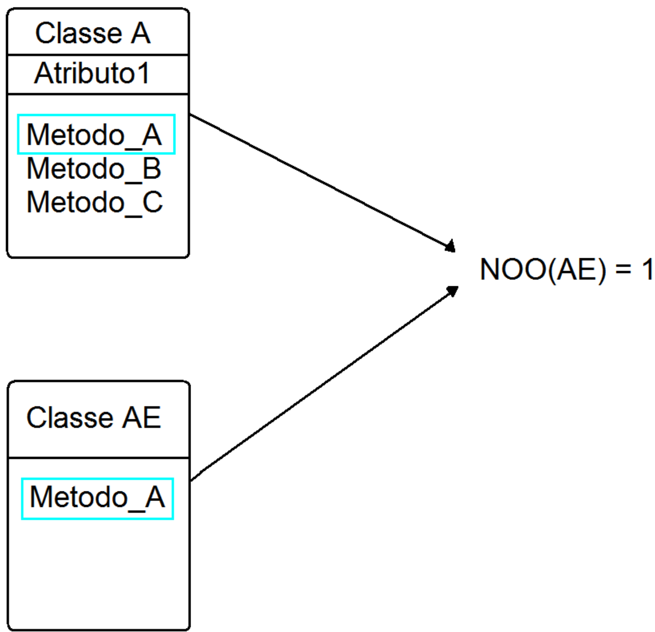

Figura 9. Exemplo da métrica NOO

SI: (*Specialization index*) número de serviços eliminados, adicionados ou redefinidos. Mostra o nível de especialização das classes ou as alterações efetuadas para atender à necessidade individual daquela classe (Figura 10).

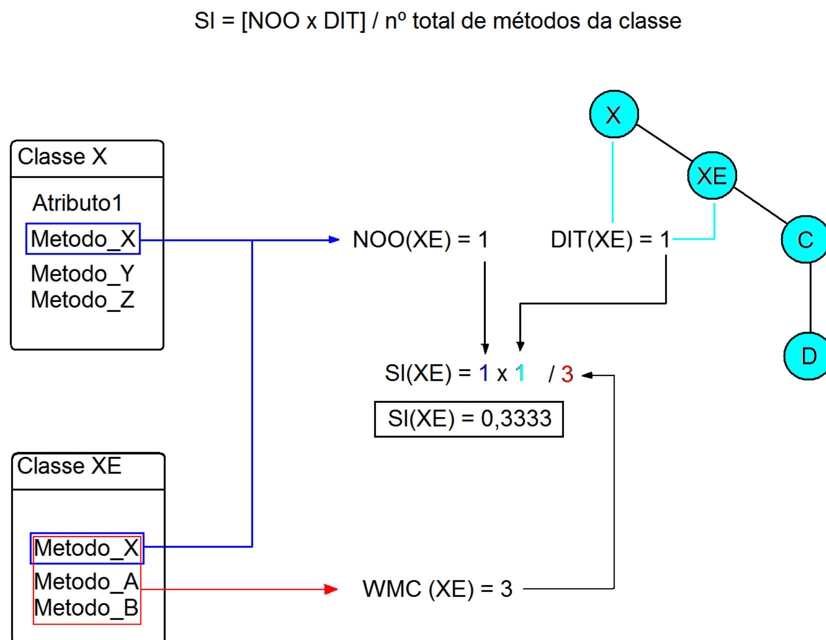

Figura 10. Exemplo da métrica SI

Uma técnica bastante interessante consiste em pesquisar intuitivamente a coesão e o acoplamento (KOSCIANSKI, 2006). Um recurso muito usado é por meio de

gráficos que demonstrem ao analisador a complexidade estrutural do produto e auxiliam a identificação de gargalos. A Figura 11 é um modelo tridimensional da estrutura de uma classe. Os cubos são atributos, acessados por métodos representados por esferas. Prontamente é percebido acoplamento ou não entre os métodos do objeto, suas dependências e oportunidades de particionamento da classe em subclasses.

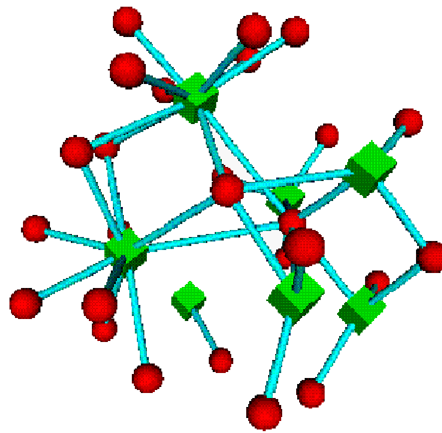

Figura 12. Visualização da estrutura de uma classe

CAPÍTULO 5 FERRAMENTAS SOBRE MÉTRICAS ORIENTADAS A OBJETOS

5.1 *JMETRIC - JAVA METRICS ANALYSER*

A ferramenta JMetric propõe a coleta de métricas OO, o projeto começou em abril de 1998 como parte de uma pesquisa relacionada a ferramentas de medição de métricas em software OO. A equipe de desenvolvimento chegou a conclusão que as ferramentas a disposição para o propósito não eram boas. Então o *JMetric* começou a ser produzida pela Universidade Tecnológica de *Swinburne* para colher métricas em projetos Java. A ferramenta é *open-source* e continua sendo aperfeiçoada. Ao executar o *JMetric* é apresentada a tela principal (Figura 12).

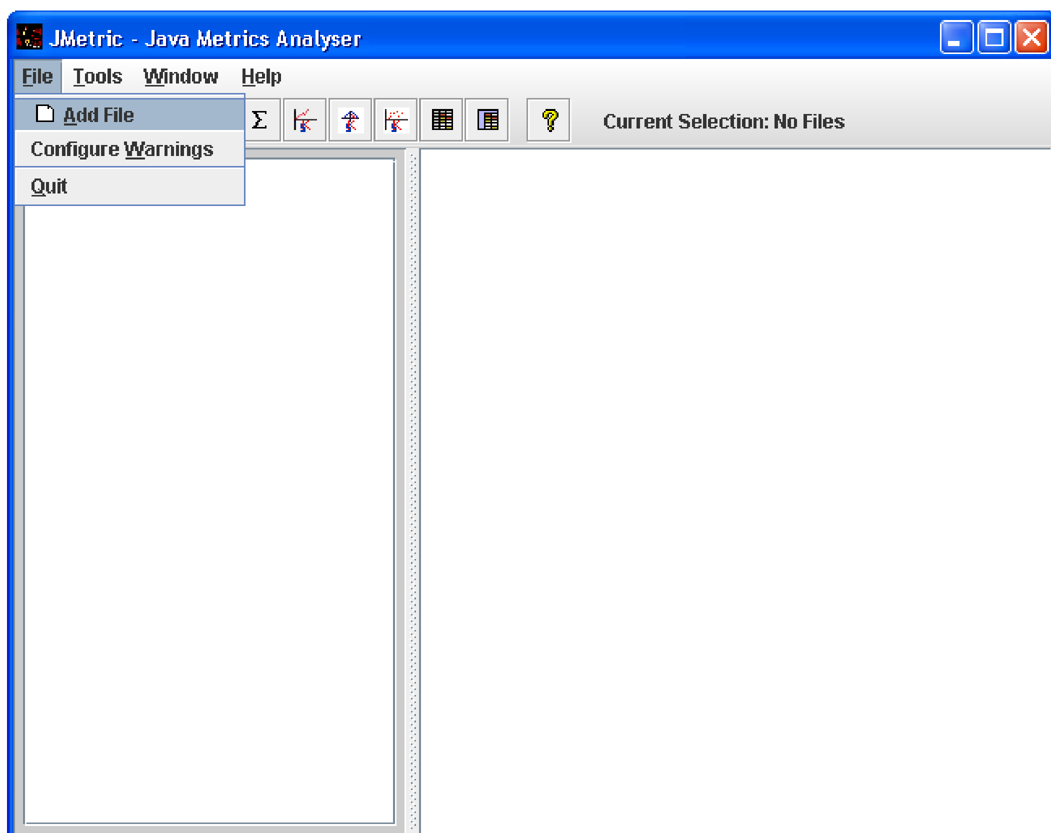

Figura 12 Tela principal do Jmetric

Para realizar o cálculo das métricas do sistema o usuário deve escolher um projeto. Deverá ser adicionado um arquivo feito em Java. Depois de adicionar o arquivo, é apresentado uma árvore com a estrutura das classes, neste caso o objeto TokenUtils (Figura 13).

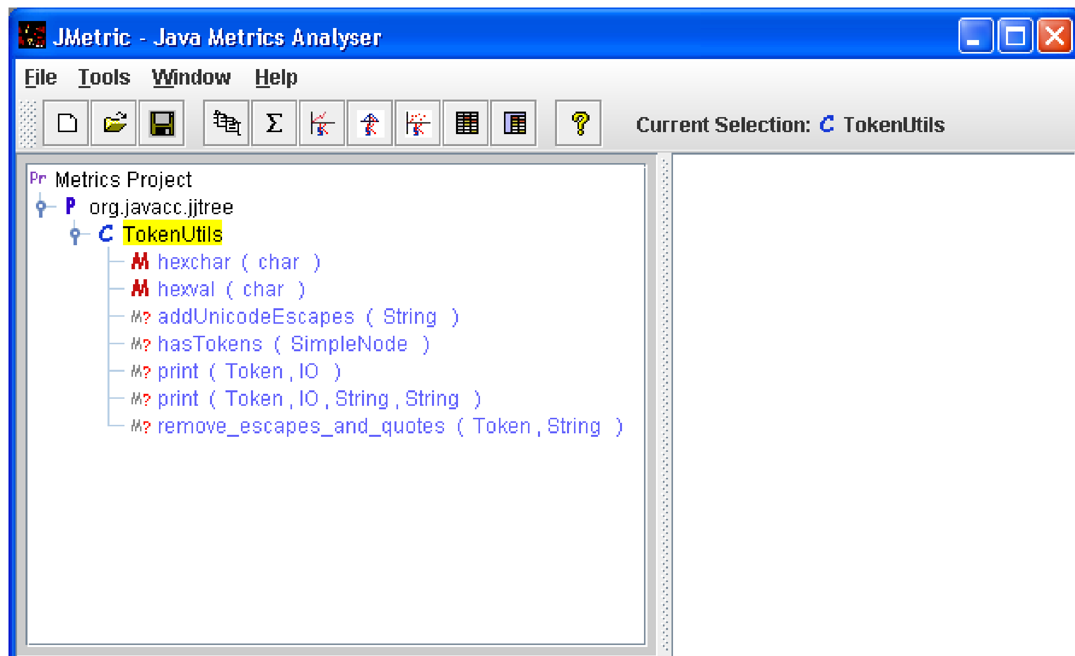

Figura 13. Árvore com a estrutura da(s) classe(s)

Em seguida, após a escolha de um arquivo para o cálculo, é necessário selecionar a alternativa de calcular (Figura 14). Depois de calcular as medidas, pode se exibir de duas formas, gráficos e tabelas, de acordo com a documentação do projeto. Então, a exibição em forma de tabela não foi possível, pois a ferramenta não mostrou resultado algum conforme o gráfico demonstrado na Figura 15.

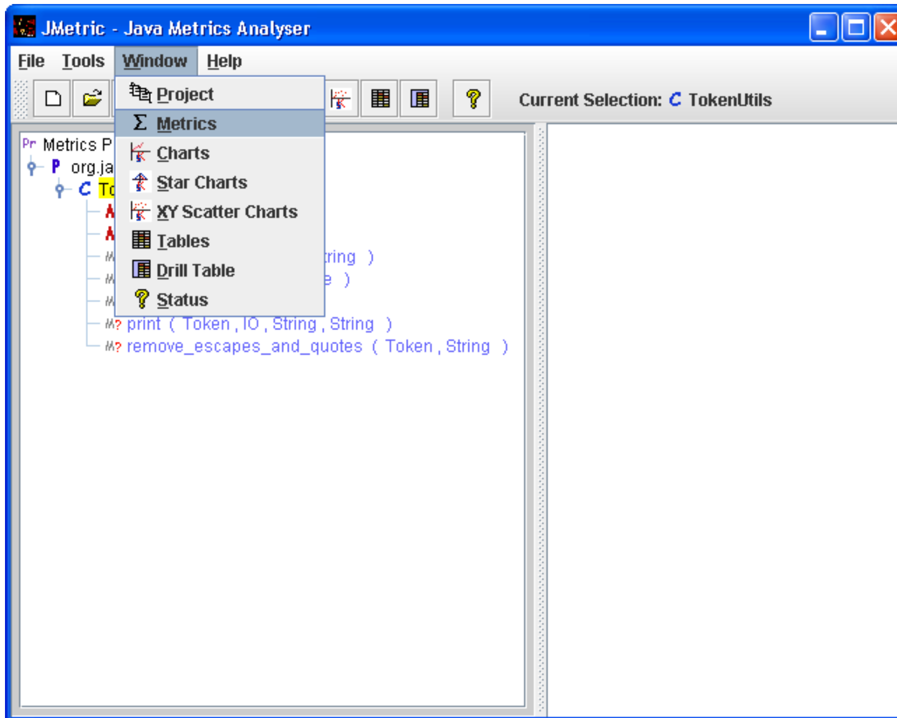

Figura 14. Calcular as métricas JMetrics

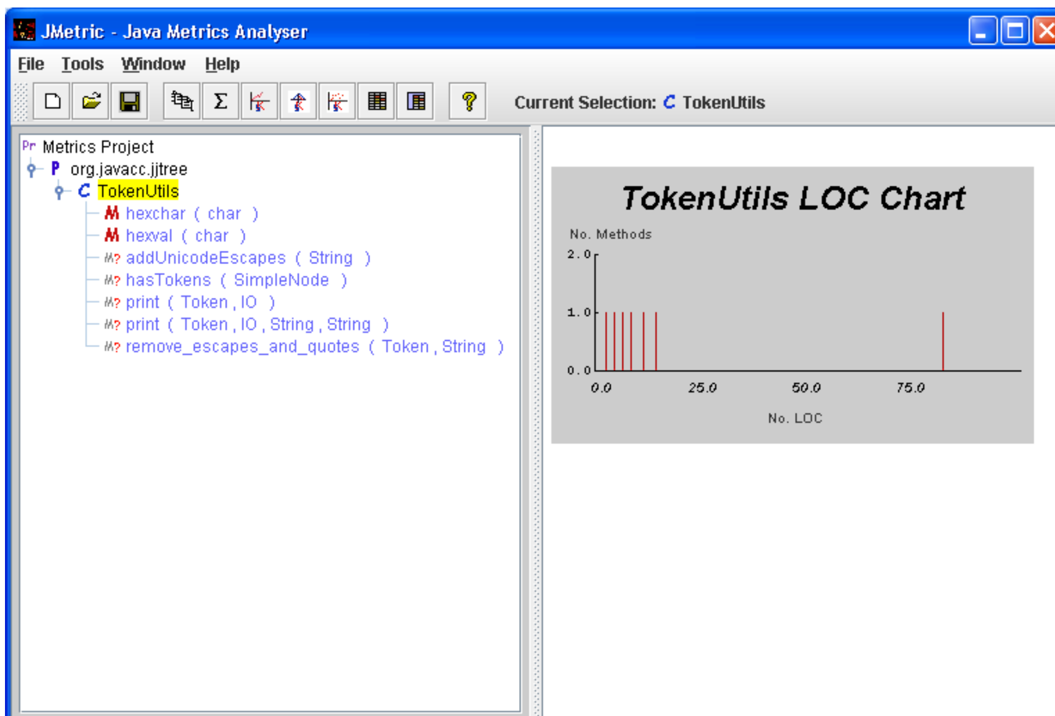

Figura 15 Demonstração do gráfico do JMetric

5.2. FERRAMENTA PARA CÁLCULO DE MÉTRICAS EM SOFTWARES ORIENTADOS A OBJETOS CODIFICADOS EM DELPHI

Seibt (2001) descreve um protótipo de uma ferramenta para cálculo de métricas em software orientado a objetos. O citado protótipo é capaz de avaliar o código fonte de um projeto OO em Delphi, extraindo as classes, seus atributos e métodos para posterior cálculo de métricas para software OO. A ferramenta proporciona calcular dezenove métricas de construção e de projeto, entre estas, métodos ponderados por classe e profundidade da árvore de herança.

5.3 MÉTRICAS PARA PROGRAMAÇÃO ORIENTADA A OBJETOS

Em Cardoso (1999) é apresentado um software implementado para dar auxílio no cálculo de métricas em sistemas OO. O sistema avalia o código fonte de projetos em Delphi e fornece cálculos de métricas como classes sucessoras, contagem de métodos, descendentes e ascendentes específicas para OO.

5.4 FERRAMENTA JAVACC (*JAVA COMPILER COMPILER*)

JavaCC é uma ferramenta geradora de *parser,* em outras palavras*,* é uma ferramenta utilizada para ler uma gramática e converter em um programa Java, e também é um analisador sintático e léxico que reconhece se um determinado texto pertence a uma determinada gramática. Através de uma gramática para o uso do *JavaCC*, será gerado código fonte de um *parser* para a linguagem escolhida (BOUDOUX, 2004).

Para que seja implementado métricas de código fonte usando o *JavaCC* é preciso seguir uma seqüência de passos básicos (BOUDOUX, 2004): Passo 1 – Adaptação/Definição da Gramática da linguagem para o JavaCC (por exemplo C++, C, Java); Passo 2– Geração do código fonte auxiliar em Java para a análise sintática e léxica; Passo 3 – Definição da Métrica a ser utilizada.

Passo 4 – Análise do código obtido no passo 2 (*parser*) para a implementação das métricas requeridas; Passo 5 – Instrumentação/adaptação do *parser* para a análise (este é o passo que exige maior complexidade); Passo 6 – Testar os resultados da métrica implementada, caso seja necessário deve-se retornar ao passo 4 para ver mais uma vez as definições da métrica desejada; Tendo como base o conhecimento da linguagem, no *parser* gerado e na definição das métricas torna-se possível a sua implementação na linguagem pretendida.

5.5 *JDEPEND*

O *JDepend* faz uma leitura de todos os diretórios que contêm classes Java e produz métricas de qualidade do *design* para cada pacote Java. Esta ferramenta proporciona que a qualidade de um *design* seja medida automaticamente em termos de reusabilidade, extensibilidade e manutenibilidade de modo a gerenciar as dependências de pacotes de forma efetiva.

Packages

[summary] [packages] [cycles] [explanations]

org.codehaus.mojo.jdepend

Afferent Couplings	Efferent Couplings	Abstractness	Instability	Distance
0	12	0.0%	100.0%	0.0%

Abstract Classes	Concrete Classes	Used by Packages	Uses Packages
None	org.codehaus.mojo.jdepend.JDependMojo org.codehaus.mojo.jdepend.JDependXMLReportParser org.codehaus.mojo.jdepend.ReportGenerator	*None*	java.io java.lang java.util javax.xml.parsers jdepend.xmlui org.apache.maven.project org.apache.maven.reporting org.codehaus.doxia.sink org.codehaus.doxia.site.renderer org.codehaus.mojo.jdepend.objects org.xml.sax org.xml.sax.helpers

org.codehaus.mojo.jdepend.objects

Afferent Couplings	Efferent Couplings	Abstractness	Instability	Distance
1	2	0.0%	67.0%	33.0%

Abstract Classes	Concrete Classes	Used by Packages	Uses Packages
None	org.codehaus.mojo.jdepend.objects.CyclePackage org.codehaus.mojo.jdepend.objects.JDPackage o.jdepend.objects.Packages org.codehaus.mojo.jdepend.objects.Stats	org.codehaus.mojo.jdepend	java.lang java.util

Figura 16: Resultados obtidos pela análise do *JDepend* por pacotes.

A Figura 16 apresenta um relatório feito pelo *JDepend* com as métricas subdivididas em pacotes. A Figura 17 corresponde a um relatório sucinto com as métricas conseguidas pelo *JDepend*. Os campos da tabela devem ser interpretados da seguinte maneira:

TC: número total de classes.
CC: número de classes concretas.
AC: número de classes abstratas.
Ca: acoplamento aferente.
Ce: acoplamento eferente.
A: nível de abstração.
I: instabilidade.
D: distância da seqüência principal.
V: volatilidade.

Summary

[summary] [packages] [cycles] [explanations]

Package	TC	CC	AC	Ca	Ce	A	I	D	V
org.codehaus.mojo.jdepend	3	3	0	0	12	0.0%	100.0%	0.0%	1
org.codehaus.mojo.jdepend.objects	4	4	0	1	2	0.0%	67.0%	33.0%	1

Figura 17: Resumo das métricas feitas pelo *JDepend*.

5.6 *JAVANCSS*

JavaNCSS é uma simples ferramenta que avalia dois padrões de métricas de código-fonte para a linguagem de programação Java: a complexidade ciclomática, também conhecida como métrica de McCabe (1996) e o número de instruções de código sem comentário (*Non Commenting Source Statements*). As métricas são coletadas integralmente – para cada função ou para cada classe. Algumas métricas disponíveis por esta ferramenta são: classes, contagem de pacotes, funções e classes internas; quantidade de comentários Javadoc por classe e método; e as médias desses valores são computadas.

CONSIDERAÇÕES FINAIS

Temos capacidade de medir a qualidade de software através de suas características internas e externas. Os fatores internos de qualidade não são percebidos facilmente pelo usuário, devido ser intrínsecos à concepção do sistema. Definimos fatores externos aqueles que podem ser de forma eficiente percebidos pelo usuário, como por exemplo: facilidade de uso, eficiência, robustez e portabilidade. Com o surgimento da globalização, as organizações almejam cada vez mais apoiar-se em sistemas de informações que possam dar relevantes contribuições aos negócios (NOGUEIRA, 2009).

A gerência de um produto de software alcança um determinado estado de precisão e qualidade se houver medidas que tornem possível a administração através dos aspectos do sistema. A métrica de software é uma medida de propriedades do sistema que podem ser definidas como caminhos para determinar quantitativamente a dimensão em que o projeto, a sequência e o produto de software têm certas características.

As métricas e as medições proporcionam uma melhor compreensão do processo utilizado para desenvolver um produto, assim como uma melhor avaliação do próprio produto e do projeto. As medições podem permitir melhorias no processo, aumentando a produtividade e comparações entre projetos de desenvolvimento. Constata-se que a partir do uso de um planejamento de garantia de qualidade que tenha o estabelecimento de métricas, consegue se obter um histórico e através dele a produtividade e qualidade do projeto irá elevar, diminuindo assim a margem de erros.

O controle de qualidade requer o uso de métricas. Porém, a utilização dessas métricas é uma tarefa difícil, pois exige esforço na aquisição da informação. Para avaliar a qualidade de software orientado a objetos não há métricas precisas. O que existe são métricas que quantificam fatores da Orientação a Objetos que nos deixam realizar um processo de avaliação.

A combinação das métricas apresentadas pode efetuar métricas de mais alto nível, gerando novos indicadores de produtividade ou qualidade para o desenvolvimento de software. Esta combinação pode ser feita pelo projetista ou pelo gerente do projeto buscando criar meios mais eficientes de controle das atividades e seus resultados.

Para conseguir resultados mais significativos, as métricas precisam ser aplicadas constantemente, envolvendo as etapas de medição, planejamento, análise de resultados, tomada de decisão e implementação das decisões. Deste modo, pode-se construir uma base histórica do artefato medido que dará suporte ao engenheiro de software analisar que processos, ferramentas e métodos se aplicam melhor àquele tipo de produto.

Além disso, a preocupação com medidas errôneas pode produzir incentivos errados, levando a consequências não esperadas. Goldratt (1991) ressalta que as pessoas agem de acordo com a forma com que estão sendo analisadas. Por meio de ferramentas automatizadas, é possível colher um grande número de métricas com menor esforço, o que possibilita a implantação de processos de medição em qualquer tipo de sistema, desde os mais simples até os mais complexos, o que contribui para a qualidade do produto final.

A gerência de riscos do projeto abrange os processos relacionados à identificação, resposta e análise aos riscos do projeto. Isto inclui a maximização dos resultados de eventos positivos e minimização das consequências de eventos negativos (PMBOK, 2000).

Conforme Nogueira (2004), a implementação das práticas de gestão de projetos é considerada fatores fundamentais de sucesso no desenvolvimento de software, devida a sua importância e da sistematização do trabalho a ser realizado. Consequentemente tais práticas podem ser alcançadas com a utilização de métricas de software. A Engenharia de Riscos envolve duas áreas chaves no seu processo. São elas: Gerenciamento de Riscos e Análise de Riscos (PETERS; PEDRYCZ, 2001).

A gestão de riscos em todo o ciclo de vida do desenvolvimento é crucial para o sucesso do projeto. O sucesso ou o fracasso do projeto está diretamente relacionado a essas variáveis. Conforme Pivetta (2002), os gerentes de projetos de sistemas de informação deveriam avaliar constantemente as métricas de software durante o processo de desenvolvimento para minimizar as possibilidades de fracassos. Como decorrência da monitoração, os membros das equipes de desenvolvimento obtêm uma melhor visão do andamento do projeto (PIVETTA, 2002). A estimativa é tão significante que alguns autores diferenciam incerteza e risco pela ausência ou presença da estimativa (MACHADO, 2002).

Fernandes (1995) propõe um modelo de medições, buscando definir as estimativas de qualidade, tempo, recursos, prazo, confiabilidade, de modo que o desenvolvimento do software, assim como a gestão do produto, possa ser gerenciada. Apenas utilizando métodos baseados em métricas já praticados e conhecidos com resultados experimentados é que se permitirá às empresas acrescentarem experiências no ajuste de seus índices de produtividade e a partir disso obterem os valores exatos para a sua realidade (LONGSTREET, 2003).

Neste trabalho apresentamos um conjunto de conceitos ligados a métricas de software. Conclui-se que há uma preocupação com a qualidade dos projetos de desenvolvimento de software orientado a objeto e que as métricas é uma das ferramentas que o gerente de projeto deve utilizar para corrigir e diagnosticar os rumos do projeto. Por isso, deve-se por em prática a adoção de métricas orientadas a objeto, combinadas com os demais tipos de métricas, sem receio. Por se tratar de uma mudança de paradigma, estas métricas ainda não estão consistentes, porém, a utilização constante das medições constituirá uma base para os ajustes necessários nos modelos adotados.

REFERÊNCIAS

AMBER, Scott W. **Análise e projeto orientado a objeto:** seu guia para desenvolver sistemas robustos com tecnologia de objetos. Tradução Oswaldo Zanelli. Rio de Janeiro: Infobook, 1998.

ARTHUR H. Watson and MCCABE, Thomas J. **Structured testing: A testing methodology using the cyclomatic complexity metric.** Technical report, NIST Special Publication 500-235, 1996. 54

ARTHUR, Lowell Jay. **Melhorando a qualidade do software: um guia para o TQM**. Rio de Janeiro: Infobook, 1994.

BATES, Bert, SIERRA Katy. **Use a Cabeça Java.** 2º Ed. Tradução: Aldair José Coelho. Rio de Janeiro: Alta Books, 2009..

BERGAMO, Marilia L. e Melo, Walcélio. **Tutorial sobre Usabilidade de Software**. Universidade Católica de Brasília, 2000.

BOUDOUX, Reis. **Implementando Métricas de Código Fonte usando JavaCC.** 2004. Trabalho de Conclusão de Curso. (Graduação em Sistemas de Informação) - Universidade Salvador. *Orientador:* Glauco de Figueiredo Carneiro.

BRIAN, Henderson-Sellers. **Object-Oriented Metrics: Measures of Complexity.** Prentice Hall PTR, 1996.

BROOKS, F.P. ***The mythical man month.*** Addison Wesley, 1975.

CARDOSO, Eduardo J. **Métricas para programação orientada a objetos.** 1999. 45 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) – Universidade Regional de Blumenau, Blumenau.

CAROLYN B. Seaman. **Qualitative methods in empirical studies of software engineering.** IEEE Transactions on Software Engineering, 25(4):557–572, 1999.

CHIDAMBER, Shyam R. and KEMERER , Chris F. **A metrics suite for object oriented design.** IEEE Transactions on Software Engineering, 20(6):476–493, 1994.

CÔRTES, Mario L.; CHIOSSI, Thelma C. S. **Modelos de qualidade de software.** Campinas: Editora da UNICAMP, 2001.

DEITEL, Harvey M. e DEITEL, Paul J.: **Java: Como Programar.** Bookman, 2004.

ELIYAHU M. Goldratt. **The Haystack Syndrome: Sifting Information Out of the Data Ocean.** North River Press, 1991.

FEIGENBAUM, A. V. (1986) - ***Total quality control.*** 3. ed. New York, cGraw Hill.

FERNANDES, Aguinaldo Aragon. **Gerência de software através de métricas: garantindo a qualidade do projeto, processo e produto**. São Paulo: Atlas, 1995.

FERREIRA, Marcelo; JARABECK, Flávio. **Programação Orientada ao Objeto com Clipper 5.0**, São Paulo, Makron Books, 1991.

FERREIRA, Marcelo; JARABECK, Flávio. CA **Visual Objects-O Livro**, SP, Express, 1995.

FONSECA, Wannessa Rocha da. **Ferramenta de Extração de Métricas para Apoio à Avaliação de Especificações Orientadas a Objetos**. Universidade Federal de Santa Catarina, 2002.

FUNCK, Mônica Andréa. **Estudo e aplicação das métricas da qualidade do processo de desenvolvimento de aplicações em banco de dados.** 1995. 104 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

HALSTEAD, Maurice H. : **"Elements of Software Science"**, Elsevier North-Holland, New York, 1977.

JACOBSON, Ivar et al. **Object oriented software engineering:** a use case driven approach Wokingham: Addison Wesley, 1992.

KÖLLING, Michael, BARNES, David. J. **Programação orientada a objetos com Java.** 4. Ed. São Paulo: Pearson Prentice Hall, 2009.

KOSCIANSKI, Andre; SOARES, Michel dos S. **Qualidade de software:** Aprenda as metodologias e técnicas mais modernas para o desenvolvimento de software. São Paulo: Novatec, 2006.

LEITE, Mário. **Curso Básico de Delphi**. Maringá, SYC, 1998.

LONGSTREET, D. (2003) – ***Use Cases e Function Points.*** *http://softwaremetrics.com/Articles/usecase.htm.*

LORENZ, Mark; KIDD, Jeff. **Object-oriented software metrics:** a practical guide. New Jersey: PTR Prenticel Hall, 1994.

MACHADO, Cristina Ângela Filipak. **A-Risk: um método para identificar e quantificar risco de prazo em projetos de desenvolvimento de software.** 2002. 239f. Dissertação (Mestrado) – Programa de Pós-Graduação em Informática Aplicada, Pontifícia Universidade Católica do Paraná, Curitiba.

MAFFEO, Bruno. **Engenharia de Software e Especificação de Sistemas.** Rio de Janeiro: Campus, 1992. 323p.

MCCABE T.J. **“A software Complexity Measure”** ,*IEEE Trans Software Engineering* Vol2 December 1976.

MOLLER, Kurt. H., PAULISH, Daniel J. **Software metrics**: a practitioneris guide to improved product development. Los Alamitos: IEEE, 1993.

NOGUEIRA, Marcelo. **Gestão de Riscos na implantação de ERP.** In: WCETE Congresso Mundial de Educação da Engenharia, 2004, Santos. Anais. USA: IEEE, 2003.CD-ROM.

NOGUEIRA, Marcelo. **Engenharia de Software**: Um Framework para Gestão de Riscos em Projetos de Software. Rio de Janeiro: Editora Ciência Moderna Ltda., 2009

PETERS, James F.;PEDRYCZ, Witold. **Engenharia de Software.** Rio de Janeiro: Campus, 2001. 602 p.

PIVETTA, Valdimir Uliana. **Modelo de apoio à gestão de riscos no desenvolvimento de software.** 2002. 103f. Monografia (MBA) – Programa de MBA em Engenharia de Software, Escola Politécnica da Universidade de São Paulo, São Paulo.

PMBOK, Project Management Institute. ***The guide to the Project management body of knowledge*.** Pennsylvania, USA, 2000.

PRESSMAN, Roger S. **Engenharia de software.** São Paulo: Makron Books, 1995.

PRESSMAN, R. S. (2000) - ***Software Engineering, A practitioner`s Approach*.** London, McGraw-Hill.

PRESSMAN, ROGER S. **Engenharia de Software.** 5º ed. Rio de Janeiro: McGraw Hill, 2002. 843 p.

PRESSMAN, R. S.. **Software Engineering: a practitioner's approach.** 6a. ed .New York, EUA: McGraw-Hill, 2005.

PRICE, Tom. **Programa de Especialização: opção para o Mestrado.** Porto Alegre,UFRGS, 1997.

REZENDE, Denis Alcides. **Engenharia de Software e Sistemas de Informações.** Rio de Janeiro: Brasport, 1999. 292 p.

ROCHA, Ana R.; MOLDONADO, José C.; WEBER, Kival C. **Qualidade de software:** teoria e prática. São Paulo: Prentice Hall, 2001.

ROSENBERG, Linda. **Applying and interpreting object oriented metrics.** Utah, abr. 1998. Disponível em: <http://satc.gsfc.nasa.gov/support/STC_APR98/apply_oo apply_oo.html>. Acesso em: 07 jun. 2006

SEIBT, Patrícia R. R. S. **Ferramenta para cálculo de métricas em softwares orientados a objetos codificados em Delphi.** 2001. 86 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) - Universidade Regional de Blumenau, Blumenau.

SHEPPERD, Martin. **Foundation of software measurement.** New York: Prentice Hall, 1995.

SOMMERVILLE, Ian. **Engenharia de Software.** 6º ed. São Paulo: Addison Wesley, 2003. 592 p.

SPINOLA, Mauro De Mesquita. **Diretrizes para o desenvolvimento de software de sistema embutidos.** 1998.251 f. Tese (Doutorado) – Programa de Pós-Graduação em Engenharia de Produção, Escola Politécnica da Universidade de São Paulo, São Paulo.

Sun Microsystems: **The Java Tutorials.** Disponível em: http://java.sun.com/docs/books/tutorial. Acesso em: 02 out. 2007.

THOMAS J. McCabe and ARTHUR H. Watson. **Software complexity.** Crosstalk: Journal of Defense Software Engineering, 7:5–9, 1994.

TONINI, Antonio C. **Métricas de software.** [S.l.], 2004. Disponível em: *http://www.spin.org.br/Pdf/metricas%202.ppt*>. Acesso em: 12 nov. 2005.