METACOM: Um Método para Análise de Correlação entre Métricas de Produto de Software e Propensão a Manutenção

Gabriel de Souza Pereira Moreira[1], Roberto Pepato Mellado[1], Adilson Marques da Cunha[1], Luiz Alberto Vieira Dias[1]

[1]Instituto Tecnológico de Aeronáutica (ITA) – São José dos Campos – SP – Brasil

{gspmoreira,rpepato}@gmail.com, {cunha,vdias}@ita.br

Abstract. *Considering that software quality technical characteristics influence its maintenance, this paper presents a Method for Correlation Analysis between Software Product Metrics and Maintenance Proneness named METACOM. The proposed method defines an Extract, Transform, and Load (ETL) process for metrics of object-oriented software and volume of software maintenance. The METACOM involves a correlation analysis model between obtained product measures to identify the most predictive metrics. Besides, this paper describes the METACOM application on the analysis of some software industry real projects. At the end, some remarks are presented about the main analysis results obtained from specialists.*

Resumo. *Considerando-se que as características de qualidade de um software influenciam no esforço de sua manutenção, este artigo apresenta um Método para Análise de Correlação entre Métricas de Produto de Software e Propensão à Manutenção denominado METACOM. O método proposto define um processo de extração, transformação e carga de métricas de software orientado a objetos e de volume de manutenções. O METACOM é composto por um modelo de análise de correlação entre as medidas obtidas, visando identificar métricas de produto mais preditivas. Descrevem-se também a aplicação do METACOM na análise de projetos reais da indústria de software e as considerações de especialistas sobre os principais resultados.*

1. Introdução

À medida que o ciclo de vida de um software se torna maior, a qualidade de código apresenta-se como um importante fator para a redução de custos de desenvolvimento e de manutenção. Apesar de difundidos os conhecimentos sobre os princípios de qualidade de software, ainda são raros os engenheiros que se utilizam de métodos, técnicas e ferramentas de qualidade de código na prática [Blanc 2009].

Segundo [Kemerer et al. 1999], a manutenção de sistemas representa um fenômeno mal-compreendido pela comunidade de pesquisa. Eles estimam que as atividades de manutenção de sistemas possam constituir mais de 50% do esforço de desenvolvimento de um software. Segundo [Bennett 2002], a manutenção constitui-se entre 40% e 90% do custo total do ciclo de vida de um software como produto. Neste início do século XXI, mais de 50% dos profissionais de software estão envolvidos em modificar aplicações existentes, ao invés de escrever novas aplicações [Jones 2007].

[Jones 2008] afirma que os maiores responsáveis pelo aumento do custo de manutenção relacionam-se ao tamanho, à complexidade e à idade do software.

Estudos empíricos recentes têm confirmado a hipótese de que as características técnicas da qualidade de um software influenciam diretamente no esforço e na assertividade da manutenção ([Ware 2007], [Ahn 2003]).

O trabalho que originou este artigo teve como objetivo realizar uma análise de correlação entre as métricas de produto de software orientado a objeto, visando identificar métricas com potencial para indicar classes com propensão a um maior volume de manutenção em projetos de software.

Nas seções 2, 3 e 4, apresentam-se revisões bibliográficas sobre Qualidade de Produto, Métricas e Manutenibilidade de Software, respectivamente. Na seção 5, propõe-se o método METACOM, concebido e implementado neste trabalho. A seção 6 descreve o estudo de caso desenvolvido neste trabalho. Na seção 7, os principais resultados são analisados e discutidos. Na seção 8, apresentam-se as limitações do trabalho e, finalmente, na seção 9 encontram-se as conclusões e recomendações para trabalhos futuros.

2. Qualidade de Produto de Software

A norma internacional ISO/IEC 25000:2005 - *Software Quality Requirements and Evaluation* (*SQuaRE*), que evoluiu a partir das séries de normas ISO/IEC 9126 e 14598, é uma das mais importantes a respeito da caracterização e medição de qualidade de produto de software. Ela define três modelos de qualidade: interna, externa e em uso, conforme apresentado na Figura 1. Neste cenário, cada tipo de qualidade mapeia-se para métricas específicas [Moreira et al., 2010].

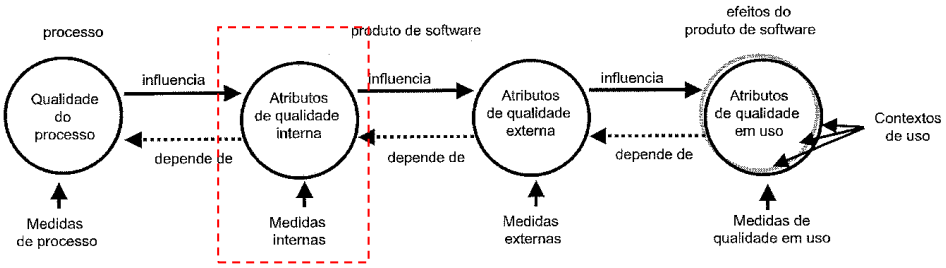

Figura 1. Qualidade no ciclo de vida de um produto de software [ISO9126-1]

Segundo a [ISO 25000], a Qualidade Interna do software é a capacidade de um conjunto de atributos estáticos de um produto de software satisfazer necessidades explícitas e implícitas, quando utilizado sob condições específicas.

Os atributos estáticos incluem os relacionados à arquitetura e estrutura do software, verificáveis por meio de inspeção ou ferramentas automáticas [ISO 25000].

A norma *SQuaRE* assume a premissa de que a Qualidade Interna influencia diretamente na qualidade externa e por conseguinte na qualidade no uso. Também preconiza a verificação dos três tipos de qualidade a partir de requisitos estabelecidos.

No Modelo de Qualidade da norma *SQuaRE*, as qualidades externa e interna decompõem-se nos seguintes fatores de influência de qualidade: funcionalidade, manutenibilidade, usabilidade, confiabilidade, eficiência e portabilidade.

Neste trabalho, o atributo manutenibilidade recebe especial interesse, pois se relaciona com a facilidade ou complexidade de modificação de um produto de software.

O Glossário de Termos de Engenharia de Software do IEEE [IEEE 610] define manutenibilidade como a facilidade com a qual um software pode ser mantido, melhorado, adaptado ou corrigido, para satisfazer requisitos especificados. Uma modificação consiste, por exemplo, em correção, implementação de melhoria ou de adaptação a mudanças.

3. Métricas de Software

Métricas ajudam engenheiros de software a ganhar percepção na construção dos produtos que desenvolvem, propiciando melhorias da qualidade. Embora métricas técnicas de software não sejam absolutas, elas fornecem uma maneira sistemática de avaliar a qualidade, a partir de um conjunto de regras bem definidas. Isto permite a identificação e correção de defeitos, antes que eles se tornem falhas [Pressman 2009].

As próximas seções descrevem as métricas de interesse para este trabalho.

3.1. Complexidade

A complexidade estrutural avalia a construção interna de um produto de software como o número de componentes e as relações entre eles. Um software com estrutura mais complexa oferece maiores dificuldades de análise pelos desenvolvedores. Pesquisas têm mostrado que o número de defeitos encontrados em programas possui forte correlação com os seus tamanhos e complexidades internas [Koscianski 2007].

Uma métrica de complexidade bastante conhecida é a do Número de Complexidade Ciclomática (*Cyclomatic Complexity Number* - *CCN*), proposta inicialmente por [McCabe1976]. Ela propicia o cálculo do número de caminhos distintos de execução de um bloco de código-fonte e permite analisar sua estrutura, fornecendo um valor numérico que o caracteriza.

A teoria de Halstead, conhecida como “Ciência de Software”, representa uma das mais utilizadas métricas de complexidade. Calculam-se as métricas de Halstead, a partir de quatro medidas primárias: número total de operandos, número total de operadores, número distinto de operandos e número distinto de operadores. A partir destas medidas primitivas, pode-se estimar, com base no seu tamanho, qual o esforço para o desenvolvimento de um código, e até mesmo o número de defeitos esperados [Halstead 1977].

3.2. Acoplamento e Coesão

A coesão e o acoplamento influenciam a habilidade de modificar o código-fonte de um programa. Como regra geral, deve-se criar métodos e componentes com alta coesão e baixo acoplamento [Blunden 2003]. As métricas de acoplamento e coesão servem para analisar o grau de dependência entre componentes de um programa, sejam eles sub-rotinas, objetos, classes ou módulos.

Coesão e acoplamento podem favorecer ou dificultar a modificabilidade do código-fonte de um programa. A coesão mede a associação entre dois componentes de um software em relação à realização de uma dada tarefa. O acoplamento mede o grau de associação entre dois componentes. Em geral, deseja-se um alto grau de coesão e um

baixo grau de acoplamento entre módulos, pois quanto menor a troca de informações, menor a complexidade da interface e mais fácil sua compreensão e manutenção.

3.3. Sistemas Orientados a Objetos

Software Orientado a Objetos (OO) possui fundamentação diferente de software desenvolvido usando métodos estruturais. Por esta razão, as métricas para Sistemas OO devem ter as características que os distinguem ajustadas [Pressman 2009]. Segundo Berard [Berard 1995], cinco características de software OO requerem métricas especializadas: localização; encapsulamento; ocultamento de informações, herança e abstração.

Um dos mais referenciados conjuntos de métricas de software OO, chamado *MOOSE (Metrics for Object-Oriented System Environments)* ou suíte de métricas *CK* foi proposto por Chidamber e Kemerer [Chidamber 1994]. Esta suíte compõe-se de seis métricas de *design* baseadas em classes: *Weighted Methods per Class (WMC); Depth of the Inheritance Tree (DIT); Number Of Children (NOC); Coupling Between Object classes (CBO); Response For a Class (RFC);* e *Lack of Cohesion in Methods (LCOM).*

[Martin 1994] propõe um conjunto de métricas, conhecidas como *design quality metrics*, que permite medir a qualidade do *design* orientado a objetos, em termos de dependências entre os subsistemas. Entre estas métricas encontram-se a de Acoplamento Aferente (*Afferent Coupling – AC*) a de Acoplamento Eferente (*Efferent Coupling – EC*) e de Associação entre Classes (*Association Between Classes – ABC*). Um *design* com elementos altamente dependentes tende a ser rígido, sem reusabilidade e difícil de manter. Objetos com alta dependência apresentam fragilidade frente às modificações [Duvall 2007].

4. Trabalhos sobre Manutenibilidade

O gerenciamento da manutenibilidade de software tem sido marginal, na maioria dos projetos. Entre as possíveis razões encontram-se a natureza subjetiva desta análise e o risco dos resultados das inspeções não chegarem ao conhecimento da gerência, mantendo-se secretamente problemas no projeto (*design*) que poderão se revelar durante a realização de tarefas de manutenção.

[Kafura 1987] e [Lehman 1980] realizaram estudos que relacionam elementos com as medidas mais extremas (*outliers*), dentro da escala das métricas, com taxas elevadas de defeitos e tempos de codificação. O trabalho de [Kafura 1987] comprovou que estes elementos eram fontes de problemas de *design*, codificação e manutenção.

[Kafura et al.,1987] relatam um estudo sobre manutenibilidade, utilizando a técnica de avaliação subjetiva. Neste trabalho, os autores relacionam métricas de software com o julgamento de especialistas familiarizados com os sistemas analisados. O uso de avaliação subjetiva permitiu verificar que as métricas de software foram capazes de fornecer ao desenvolvedor e ao gerente de projetos uma informação consistente com a experiência dos especialistas, podendo ser usada como guia para evitar manutenção de baixa qualidade. Entre as sete métricas de complexidade utilizadas encontravam-se: o Número de Complexidade Ciclomática; a métrica de esforço de Halstead; e Linhas de Código (*Lines of Code - LoC*).

Em [Ware 2007], desenvolveu-se um estudo de como métricas de produto de software podem auxiliar os processos de tomada de decisão durante a fase de manutenção. No trabalho supracitado, realizou-se a análise de Pareto, com a seguinte hipótese: "pode-se remover 80% dos problemas de software concentrando o esforço em melhorar apenas 20% da base de código". O estudo utilizou métricas de OO, no nível de classe, propostas na suíte de Chidamber and Kemerer (CK), e identificou um potencial preditivo de volume de manutenções nas métricas de acoplamento *RFC (Response For a Class)* e *CBO (Coupling Between Objects)*.

4.1. Índice de Manutenibilidade (*Maintainability Index - MI*)

A medição da manutenibilidade de um sistema é desejável como preditor da dificuldade ou esforço a empreender para a realização de modificações no sistema. Empregam-se esforços em medir e rastrear manutenibilidade, para ajudar a reduzir ou a reverter a tendência dos sistemas sofrerem entropia, ou terem sua integridade degradada. Também se utiliza o *MI* para indicar quando se torna mais barato ou menos arriscado reescrever o código, ao invés de apenas modificá-lo [SEI 1997].

[Oman et al., 1992] propuseram o Índice de Manutenibilidade (*Maintainability Index - MI),* em uma tentativa de determinar a manutenibilidade de sistemas de software baseados no status do respectivo código-fonte. Este índice baseia-se em medições que os autores daquele trabalho executaram em sistemas de software e calibraram seus resultados com opiniões dos engenheiros que mantinham os sistemas. Os resultados examinados foram ajustados em uma função linear que melhor os representou. Desde então, acrescentou-se um pequeno número de melhorias na função [Kuipers 2007].

Segundo [Oman 1992], calcula-se o *MI* usando uma combinação de métricas, calibradas para um grande conjunto de projetos reais da indústria. O valor de *MI* é calculado por uma expressão polinomial dada pela Equação 1, onde todas as variáveis são valores médios por módulo de código [SEI 1997].

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

Equação 2. Índice de Manutenibilidade (*Maintainability Index – MI*) [SEI1997]

Onde:

- aveV = Volume V de Halstead [Halstead 1977] médio por módulo;
- aveV(g') = Complexidade Ciclomática [McCabe 1976] média por módulo;
- aveLoC = Contagem média de Linhas de Código por módulo; e, opcionalmente,
- perCM = Percentual médio de linhas de comentário por módulo.

Entretanto, aconselha-se a realização de análise crítica dos resultados desses pesquisadores, com relação à efetividade deste índice, além de se sugerir a avaliação de outras métricas além das de *LoC*, de Complexidade Ciclomática e de Halstead, para determinar manutenibilidade [SEI 1997].

Segundo [Welker 2001], sistemas OO possuem tipicamente tamanhos menores de módulo (classe), com menos operadores, operandos, caminhos de execução e linhas de código. Portanto, em OO, o valor de *MI* tende a ser maior. Entretanto, classes e heranças deveriam ser consideradas em customizações de *MI* para sistemas OO. Para Welker, o *MI* representa um excelente guia para direcionar a investigação humana. Mas

deve-se ter consciência das limitações de suas métricas, pois mudanças em tecnologias requerem também mudanças nas métricas que as representam.

5. METACOM - Método para Análise de Correlação entre Métricas de Produto de Software e Propensão a Manutenção

Neste trabalho, concebeu-se e desenvolveu-se o Método para Análise de Correlação entre Métricas de Produto de Software e Propensão à Manutenção – METACOM, apresentado na Figura 2. Este método propicia um processo automático de Extração, Transformação e Carga (*Extract, Transform and Load - ETL*) de métricas de produto e métricas de volume de manutenção. Ele envolve também o modelo de análise de correlações entre estas métricas.

O METACOM pode ser aplicado para extrair informações do histórico de projetos de desenvolvimento de software em execução ou finalizados. Os requisitos para sua utilização são a existência: de um Sistema de Controle de Versões que registre o histórico de modificações de códigos-fonte; de uma ferramenta de *issue tracker* com o registro dos casos de uso e dos defeitos detectados durante o projeto; e de um mecanismo de rastreabilidade que permita identificar mudanças de código relacionadas a casos de uso ou defeitos.

O METACOM não é restrito somente a software orientado a objetos. Ele apenas requer uma linguagem de programação para a qual existam ferramentas de análise estática de código, a fim de possibilitar a extração de métricas de produto. Os seus principais passos são:

- ***Passo 1 - Checkout* das revisões** – Neste passo, realiza-se uma cópia para a máquina local do código-fonte de todas as revisões nas quais houve alterações de classes dos projetos analisados. Neste trabalho, define-se revisão como um identificador único de um conjunto de alterações no código-fonte, enviado pelo desenvolvedor para o sistema de Controle de Versão através de uma operação de *commit*;
- ***Passo 2* - Compilação** – Neste passo, efetua-se a compilação e a geração dos binários;
- ***Passo 3* - Análise Estática de Código** – Neste passo ocorre a extração das medidas de software, a partir do código-fonte e/ou arquivos binários gerados pela compilação;
- ***Passo 4* - Extração do Histórico de Modificações** – Neste passo, se realiza a extração das medidas de volume das modificações das classes, a cada revisão, a partir dos *logs* de alteração mantidos pelo Sistema de Controle de Versões - SCV. Estas métricas incluem a soma de linhas adicionadas, a soma de linhas removidas e o total de revisões com modificações das classes;
- ***Passo 5* - Extração de informações de Casos de Uso e rastreabilidade** – Neste passo, ocorre a extração de informações dos Casos de Uso (*Use Cases - UC*) dos projetos, a partir de um sistema de monitoramento de artefatos (ferramenta de *issue tracker)*. Também são extraídas informações de rastreabilidade entre os *UC* e as classes que os compõem, através de uma integração entre as ferramentas de *issue tracker* e SCV*;*
- ***Passo 6* - Transformação e Carga** – Neste passo, realiza-se a seleção, estruturação e normalização das medidas e informações extraídas nos passos 3, 4 e 5 e sua carga em um *Data Warehouse* , modelado para análise das séries históricas das medidas de software;
- ***Passo 7* – Análise de correlações entre métricas de produto e volume de manutenções –** Este passo, descrito detalhadamente na seção 7, tem por objetivo indicar quais as métricas com maior potencial preditivo para apontar classes com propensão à alto volume de modificações. Esta análise pode basear-se apenas no histórico de modificações das classes do produto de software analisado, ou no histórico de manutenções de sistemas de mesma plataforma tecnológica e paradigma de desenvolvimento.

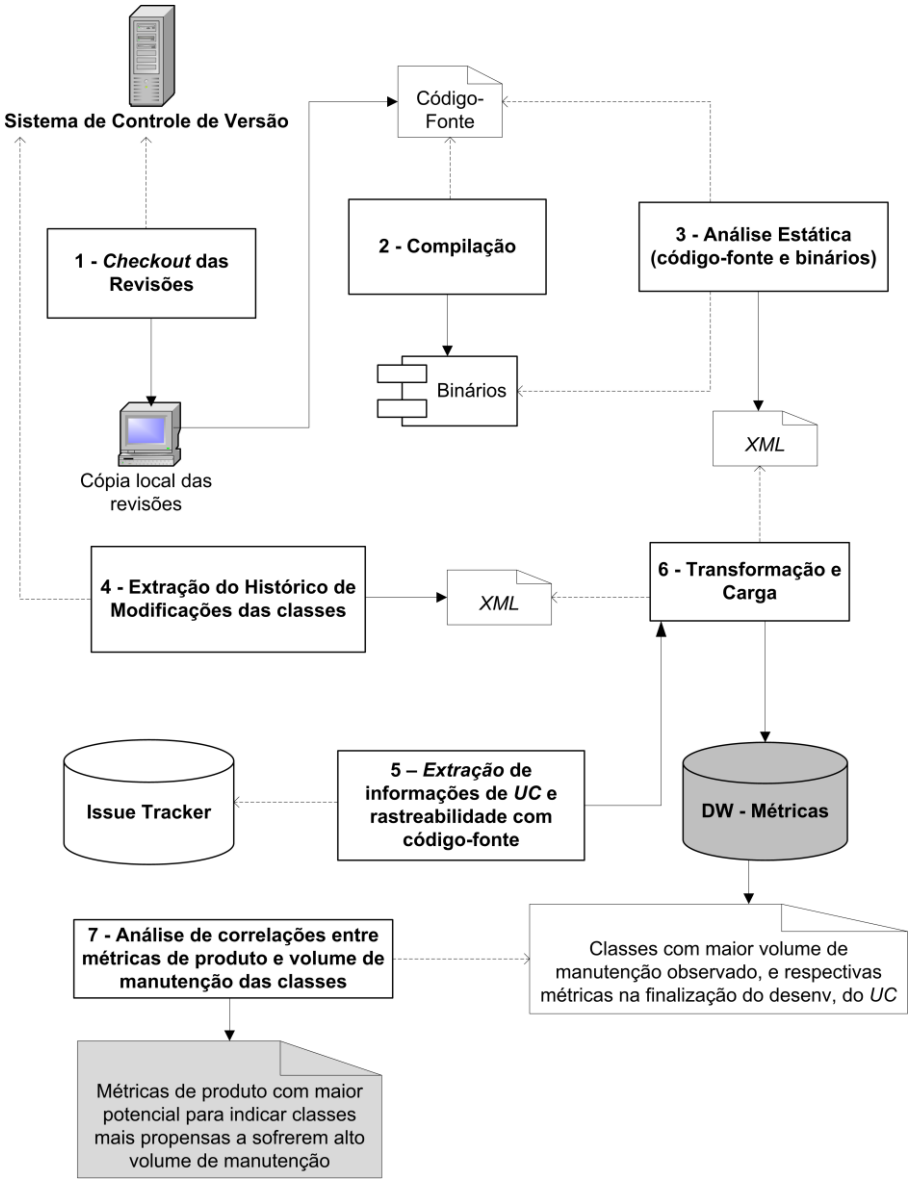

Figura 2. Método para Análise de Correlação entre Métricas de Produto de Software e Propensão a Manutenção - METACOM

6. Estudo de Caso

Neste trabalho de pesquisa, buscou-se analisar a hipótese de que métricas de produto de software podem ser utilizadas para indicar classes com maior propensão à manutenção. Este tipo de informação pode auxiliar a responder questões técnicas e gerenciais, como:

- Quais funcionalidades do código possuem maior probabilidade de causar problemas no futuro, se não forem ajustadas agora?
- Em que partes do código se deveriam investir mais esforços em inspeções e testes?
- Para quais partes do código deveriam ser atribuídos os membros dos times mais habilidosos e experientes?

Para a realização desta análise, optou-se pela utilização da técnica estatística de correlação, na tentativa de encontrar uma tendência linear direta ou inversa de métricas de produto de software OO e volume de manutenção.

6.1. Projetos de Software Considerados

Para a análise da evolução de métricas, ao longo do desenvolvimento, e sua relação com o volume de manutenção, foram analisados dois produtos desenvolvidos em projetos reais de uma fábrica de software. Ambos os projetos encontram-se voltados à plataforma *web*, segundo o paradigma de orientação a objetos e com objetivos de negócio e tecnologias similares. Em sua maior parte, a codificação dos projetos foi realizada por profissionais distintos, tendo três desenvolvedores em comum como participantes dos dois projetos.

Os projetos foram executados segundo o processo de desenvolvimento estabelecido pela Fábrica de Software, baseado no *Rational Unified Process* (*RUP*). O processo incluía como requisitos do projeto a modelagem dos casos de uso, o registro dos defeitos em ferramenta de *issue tracker* e a rastreabilidade entre código-fonte, casos de uso e defeitos. Este cenário permitiu obter informações detalhadas a respeito do histórico de desenvolvimento dos mesmos e identificar relações entre métricas de produto e volume de manutenções, em nível de classe.

Na Tabela 1, apresentam-se informações sobre os projetos, já em fase de manutenção, e na Tabela 2, dados sobre os produtos de software produzidos.

Tabela 1. Projetos considerados no estudo de caso

Projeto	Plataforma	Início Desenv.	Nº Revisões*	Desenvolvedores Distintos	Defeitos	Casos de Uso
A	Web (C#.NET 2.0)	08/2008	2.977	20	1.387	103
B	Web (C#.NET 2.0)	08/2009	1.358	8	313	93

* **Revisão** – Identificador único de um conjunto de alterações no código-fonte, enviado da cópia local do desenvolvedor para o sistema de Controle de Versão através de uma operação de *commit*.

Tabela 2. Tamanho dos Produtos de Software considerados no estudo de caso

Proj.	Plataforma (Linguagem)	LoC Total	LoC Compilável	Linhas de Comentários	Tipos	*Assemblies**
A	Web (C#.NET 2.0)	97.488	51.017	31.670	297	23
B	Web (C#.NET 2.0)	70.735	33.644	27.795	407	10

* ***Assembly*** - Componente binário do software, consistindo geralmente em arquivo executável ou Biblioteca de Vínculo Dinâmico (*Dynamic Linked Library – DLL*)

6.2. Implementação do METACOM

Para possibilitar este estudo de caso, o METACOM foi implementado, envolvendo a integração de diversas ferramentas *open-source* e comerciais, além do desenvolvimento de duas novas ferramentas e de diversos *scripts* de automatização, conforme Tabela 3.

Tabela 3. Ferramentas utilizadas na implementação do METACOM

Passo do METACOM	Ferramenta	Descrição da implementação
1 - Checkout das revisões	*SVN Extractor*	Ferramenta desenvolvida para esta implementação do METACOM, a qual realiza *checkout* de todas as revisões do projeto, a partir da utilização do Sistema de Controle de Versão (SCV) *Subversion* (*SVN*)
2 - Compilação	*NAnt* e	Foram implementados scripts para automatização da

	MSBuild	compilação .NET utilizando as ferramentas *NAnt* e *MSBuild*.
3 - Análise Estática de Código	*NDepend* e *SDMetrics C#*	Ferramentas de análise estática de código .NET foram utilizadas para extração de métricas tradicionais e especializadas em OO. *SDMetrics C#* [SDMETRICS 2011] e *NDepend* [NDEPEND 2011] foram responsáveis por analisar o código-fonte e os binários gerados, respectivamente.
4 - Extração do Histórico de Modificações	*SVNStat*	A ferramenta *SVNStat* foi utilizada para extrair, dos *logs* do SVN, informações de linhas de código adicionadas e removidas nos arquivos a cada revisão.
5 - Extração de informações de Casos de Uso e rastreabilidade	*Mantis*	Ferramenta de *issue tracker* utilizada nos projetos, a partir da qual foram obtidas informações de rastreabilidade e histórico de alterações de casos de uso e defeitos.
6 - Transformação e Carga	*DataExtractor*	Esta ferramenta foi desenvolvida para finalizar o processo de ETL das informações obtidas dos passos 3, 4 e 5 para um *Data Warehouse*.
7 – Análise de correlações entre métricas de produto e volume de manutenções	*SPSS 17.0*	Software de pacote estatístico utilizado neste trabalho para a realização das análises de estatística descritiva e correlações.

6.3. Estratégia de Análise Exploratória

No METACOM, voltado para projetos desenvolvidos conforme o paradigma de orientação a objetos, a classe é a entidade principal, da qual são extraídas medições relacionadas ao tamanho, complexidade, coesão e acoplamento com outras classes.

Buscou-se relacionar as métricas da classe, em uma determinada revisão, com o volume de modificações na classe realizadas após esta revisão. Para este experimento, o volume de modificações foi composto por quatro medidas básicas, que acumularam as informações de todas as revisões posteriores a revisão analisada, até o momento em que as métricas foram extraídas.

Como as métricas citadas acumulam os valores de revisões futuras, a tendência é de que classes desenvolvidas há mais tempo apresentem maior volume de alterações. Por este motivo, as três primeiras métricas básicas foram normalizadas em função do número de meses transcorridos, gerando-se duas métricas derivadas, com valores médios mensais.

Tabela 4. Métricas Básicas e Derivadas de Volume de Manutenção

Métricas Básicas	
LoC Adic	Soma do número de linhas de código adicionadas a classe, após a revisão analisada (revisões posteriores até o momento de extração das métricas)
LoC Rem.	Soma do número de linhas de código removidas da classe, após a revisão analisada (revisões futuras)
Nº Revisões	Total de revisões em que a classe foi alterada, após a revisão analisada (revisões futuras)
Meses Após Revisão	Total de meses transcorridos entre a revisão e a extração das métricas
Métricas Derivadas	
Média Mensal de LoC Modificado (Adic. + Rem.)	Soma das linhas de código modificadas (adicionas e removidas), normalizadas pelo número de Meses Após Revisão.
Média Mensal de Revisões	Total de revisões de alteração da classe normalizado pelo número de Meses Após Revisão.

Durante o desenvolvimento de uma funcionalidade, é comum a realização de vários *commits* (processo de armazenar o código-fonte no SCV) das classes relacionadas. Porém, para avaliar o volume de modificações de manutenção das classes, foi necessário identificar o momento em que as funcionalidades ou casos de uso tiveram o seu desenvolvimento encerrado, e se iniciou a fase de manutenção da funcionalidade.

Para possibilitar a análise, neste experimento, utilizou-se o histórico do fluxo (*workflow*) dos casos de uso dos projetos no sistema de *issue tracker*, que registra a data/hora das alterações de status atribuíveis aos casos de uso: identificado, em análise, em desenvolvimento, em testes e em validação. A integração implementada entre as ferramentas de *issue tracker* e SCV permitiu a rastreabilidade entre os *UC* e as classes, identificando-se que classes compõem quais casos de uso.

A partir dessa rastreabilidade, foi possível identificar qual a revisão de alteração da classe corresponde ao momento em que se iniciou a etapa de manutenção do caso de uso. Neste trabalho, assume-se que a manutenção de uma classe se inicia quando o respectivo caso de uso for validado pelo testador.

Para a análise de correlação entre as métricas da classe e o volume de manutenção, considerou-se, para cada classe, apenas a revisão em que se iniciou a manutenção, ignorando-se as modificações na construção inicial da funcionalidade.

Como se observou que as métricas envolvidas na correlação não apresentaram distribuição estatística normal, então, optou-se pela utilização do método de coeficiente de correlação de postos de Spearman [Dickey et al. 1979]. A utilização deste método fundamenta-se na distribuição de freqüências das variáveis analisadas que, em alguns casos, possuem comportamento não linear. O sentido da correlação não pôde ser previsto para todos os casos. Portanto, a estratégia de teste seguiu o modelo *two-tailed*.

7. Análise e Discussão dos Resultados Obtidos

Neste estudo de caso, realizou-se uma análise da correlação entre as medidas de 40 métricas de produto de software e de 5 métricas de volume de manutenção, básicas e derivadas. O resultado desta análise foi consolidado na matriz de correlações da Tabela 4, onde se apresentam 9 das 40 métricas de produto, para as quais os autores , baseados

em suas experiências na indústria de software, tinham a expectativa de encontrar correlações representativas entre as métricas selecionadas e as métricas de volume de manutenção.

Tabela 4. Correlações de interesse entre Métricas de Produto de Software e Métricas de Volume de Manutenção (correlações acima de 0.6 em negrito)

Métricas das Classes (antes do início das manutenções)		**Métricas Básicas de Volume de Manutenção**			**Métricas Derivadas de Volume de Manutenção normalizadas por mês**	
Categoria de Métrica	**Métrica**	**LoC Adic. Posterior**	**LoC Rem. Posterior**	**Revisões Post.**	**Média Mensal de LoC Modificado (Adic. + Rem.) Post.**	**Média Mensal de Revisões Posteriores**
Tamanho	*LoC*	,586	,511	,491	**,609**	,518
Acoplamento	*AC*	-,126	-,168	-,139	-,156	-,165
	EC	,563	,538	,527	**,632**	**,601**
	ABC	**,628**	**,608**	,585	**,643**	**,609**
Coesão	*LCOM*	,044	,020	,015	,002	-,019
Complexidade	*MCC*	,510	,486	,453	,486	,444
	HUOpt	,574	,527	,516	,558	,510
	HUOpd	**,654**	**,608**	,593	**,653**	**,603**
Manutenibilidade	*MI*	-,404	-,419	-,393	-,397	-,409

A fim de verificar os resultados encontrados, foram convidados engenheiros de software experientes, participantes do desenvolvimento dos projetos analisados. A seguir, apresentam-se as principais considerações desses engenheiros sobre as correlações entre as métricas de Volume de Manutenções (VM) e as 9 métricas selecionadas pelos autores deste artigo:

- ***VM x LoC*** – Como esperado, a quantidade de linhas de uma classe, ao final do desenvolvimento do respectivo caso de uso, e a quantidade de linhas modificadas em revisões futuras mostraram-se moderadamente correlacionadas. Este resultado alinha-se com o conceito de que classes com muitas linhas de código costumam indicar problemas de *design* OO, pois ferem o S*ingle Responsability Principle (SIP), tornando-se mais propensas às modificações* [Martin 2002]*;*
- ***VM x EC*** **e** ***VM x ABC*** – Entre as maiores correlações com volumes de manutenção encontrados encontram-se as métricas de acoplamento. Estes resultados vão ao encontro do estudo de [Ferneley 1999] onde havia sido encontrada forte correlação entre volume de manutenção e acoplamento de exportação (*export coupling* ou *fan-out*), cujas métricas semelhantes em OO são *EC* e *ABC*. As métricas de *EC* e *ABC*, específicas para sistemas OO, indicam qual o grau de dependência de uma classe para com outras classes. Quanto maior a dependência externa, maior a possibilidade de que esta classe sofra modificações, quando os tipos dos quais ela depende forem alterados. Como observado nos resultados, pode-se inferir que classes que acumulam diversas responsabilidades, podem ser alteradas por várias razões, apresentando maior propensão às manutenções;
- ***VM x AC*** – A quantidade total de classes que dependem da classe analisada apresentou uma fraca correlação com o VM. Estes resultados também estão

alinhados com o estudo de [Ferneley 1999], onde foi encontrada fraca correlação entre acoplamento de importação (*import copling* ou *fan-in*) e volume de manutenção. Isto pode indicar que classes com *AC* elevado não se encontram necessariamente mais propensas à modificação. De fato, alterações em uma classe que possui elevado *AC* podem provocar um alto volume de modificações no sistema, já que esta é utilizada por diversas outras classes no sistema. Porém este impacto não se pode observar, a partir do volume de modificações da classe analisada, medido neste trabalho, mas sim de suas classes dependentes;

- ***VM x LCOM*** – Ao contrário do esperado pelos autores, não se observou neste experimento correlação entre a coesão de uma classe e volume de manutenção;
- ***VM x MCC*** – A complexidade média dos métodos de classe apresentou moderada correlação com o volume de manutenção. A complexidade tende dificultar a análise do código-fonte, porém, observou-se que este fator impactou, de forma moderada, a quantidade de linhas adicionadas e removidas;
- ***VM x HUOpd e VM x HUOpt*** – A maior correlação com volume de manutenção encontrada ocorreu com a métrica *Halstead Operands* (*HUOpd*): o número de operandos de uma classe. Esta correlação representativa era esperada pelos autores deste trabalho e pôde ser verificada na análise realizada. Pode-se inferir que a utilização de muitos operandos (e.g. objetos, identificadores, constantes) em uma classe pode aumentar o acoplamento e a complexidade da mesma, impactando em elevado volume de manutenções. Já a métrica *Halstead Operators* (*HUOpt*) apresentou correlação moderada com volume futuro de manutenção. Esta métrica computa o número de operadores da classe, que inclui instruções condicionais, qualificadores e palavras reservadas; e
- ***VM x MI*** – O índice de manutenibilidade, proposto em [Oman et al., 1992] e avaliado pelo *SEI* como promissor devido ao seu poder e simplicidade, apresentou fraca correlação com volume de manutenções. Esta situação pode indicar a necessidade de reajustar as constantes da função de cálculo do MI ou para inclusão de fatores relacionados ao *design* OO, como métricas de acoplamento, que apresentaram correlação representativa com volume de manutenção neste trabalho.

8. Limitações

Este experimento baseou-se em dois projetos desenvolvidos em uma mesma empresa, com o mesmo processo de desenvolvimento e tendo os produtos com objetivo de negócio e semelhantes plataformas tecnológicas. Recomenda-se a implementação do METACOM em outros contextos para comparação dos resultados de correlação obtidos.

As métricas de volume de manutenção foram extraídas, a partir da rastreabilidade entre modificações de código, casos de uso e defeitos. Esta rastreabilidade depende de um processo manual, no qual o desenvolvedor deve informar o identificador do caso de uso ou do defeito no momento de *commit* de alterações para o SCV. Isto provavelmente permitiu que alterações de manutenção não fossem devidamente associadas aos UC ou defeitos.

Nesta investigação, a manutenção não foi avaliada quanto à perspectiva de sistema, mas em nível de classes. Em projetos desenvolvidos iterativamente, como neste

estudo de caso, funcionalidades são mantidas enquanto outras novas são desenvolvidas, a cada iteração. Portanto, considerou-se que o início da manutenção de uma classe ocorre quando o primeiro UC associado à mesma foi validado. A partir desta revisão, todas as alterações nesta classe serão consideradas como manutenção. Esta situação pode gerar desvios em métricas de volume de manutenção, se houverem outros casos de uso que ainda estejam em desenvolvimento e que façam uso da mesma classe.

9. Conclusões

O trabalho que deu origem a este artigo teve como objetivo realizar uma análise de correlação entre as métricas de produto de software orientado a objeto, para identificar métricas com potencial de indicar classes com propensão a um maior volume de manutenção, como observado nos projetos da indústria de software considerados neste artigo. Com este propósito, concebeu-se o Método para Análise de Correlação entre Métricas de Produto de Software e Propensão a Manutenção - METACOM, que descreve um processo de Extração, Transformação e Carga (*Extract, Transform, and Load – ETL*) de métricas de produto de software e de volume de manutenção, a partir de códigos-fonte, de ferramentas de controle de versão e de i*ssue tracker*. O METACOM foi aplicado com sucesso em um estudo de caso que analisou dois softwares produzidos em projetos da indústria.

Para a análise exploratória das métricas, utilizou-se o método de correlações de Spearman. A análise do resultado das correlações permitiu invalidar e confirmar expectativas prévias dos autores sobre o potencial preditivo de algumas métricas, baseadas em suas experiências na indústria. Correlações moderadas entre: volume de manutenções e Linhas de Código (*LoC*); *Halstead Operands* (*HUOpd*) e Complexidade Ciclomática Média dos métodos de classe (*MCC*) se confirmaram. Entretanto, as métricas de falta de coesão (*LCOM*) e de Acoplamento Aferente (*AC*) da classe apresentaram fraca correlação com volume de manutenções, ao contrário do esperado.

As métricas de Acoplamento Eferente (*EC*) e de Associação Entre Classes (*ABC*) que indicam o grau de dependência de uma classe para com outras, apresentaram correlação representativa com volume de manutenções. Estas métricas retrataram fortemente a qualidade do *design* e acrescentaram outra perspectiva de predição de manutenção no paradigma OO.

Por fim, o Índice de Manutenibilidade (*Maintainability Index - MI)* proposto por [Oman, 1992], recomendado pelo *Software Engineering Institute - SEI* [SEI 1997] e utilizado por diversas organizações, não apresentou correlação com o volume de manutenção.

9.1. Recomendações e Sugestões para Trabalhos Futuros

Recomenda-se que, baseado nas correlações obtidas e apresentadas nesta pesquisa, avaliem-se as métricas de acoplamento *EC* e *ABC* em um novo índice de manutenibilidade, de forma a melhorar a assertividade da predição de volume de manutenções em sistemas OO.

Este trabalho endereçou um estudo de caso envolvendo apenas produtos de software orientados a objetos e estaticamente tipados, desenvolvidos na linguagem C#. Para a sua complementação, sugere-se a experimentação do METACOM em outras linguagens estaticamente tipadas como Java, Object Pascal e C++, para as quais se

esperam comportamentos semelhantes. Para outras linguagens dinamicamente tipadas como Python, PHP, Javascript e Ruby, acredita-se que as abordagens de análise tenham de ser tratadas de forma diferente, principalmente devido às particularidades relacionadas às meta-programações e ao *design* dinâmico destas linguagens. Nestes casos, para essas linguagens, sugere-se uma investigação mais específica e aprofundada.

9.2. Agradecimentos

Os autores agradecem o suporte da empresa IMAGEM – Soluções de Inteligência Geográfica, por permitir acesso ao seu histórico de manutenção de software, e também ao Instituto Tecnológico de Aeronáutica – ITA, por prover o ambiente acadêmico de pesquisa.

10. Referências

AHN, Y.; SUH, J.; KIM, S.; KIM, H. (2003) “The software maintenance project effort estimation model based on function points”. Journal of Software Maintenance, John Wiley & Sons, Inc., New York, NY, USA, v. 15, n. 2, p. 71-85. ISSN 1040-550X.

BERARD, E. (1995) “Metrics for Object-Oriented Software Engineering". Computer Software Engineering.

BLANC-DIT-GRENADIER, N.; RAMBERT, M.; SOURROUILLE, J.-L.; AUBRY, R. (1995) “Toward a real integration of quality in software development”. In: Paris France: p. CDROM. INSA PACTE QUALITE.

BLUNDEN, B. (2003) “Software Exorcism - A Handbook for Debugging and Optimizing Legacy Code”.: Apress.

DICKEY, D.; FULLER, W. (1979) “Journal of the American Statistical Association”, Vol. 74, No. 366, pp. 427 – 431.

DUVALL, P. M.; MATYAS, S.; GLOVER, A. (2007) “Continuous integration - improving software quality and reducing risk”: Addison-Wesley.

FERNELEY, E. (1999) “Design Metrics as an Aid to Software Maintenance: An Empirical Study”, J Softw Maint Evol - R, 11, pp. 55 – 72

HALSTEAD, M. (1977) “Elements of Software Science” : Elsevier North-Holland.

IEEE 610-1990 “Standard Glossary of Software Engineering Terminology”: IEEE.

ISO 25000-2005. “Software Engineering - Software product Quality Requirements and Evaluation (SQuaRE) - Guide to SQuaRE”.: ISO.

JONES, C. (2007) “Geriatric Issues of Aging Software”. Disponível em: <http://www.stsc.hill.af.mil/crosstalk/2007/12/0712Jones.html>. Dec 2007.

JONES, C. (2008) “Applied Software Measurement” (third edition). New York, NY, USA: McGraw Hill, 2008.

KAFURA, D.; REDDY, G. (1987) “The use of software complexity metrics in software maintenance”. IEEE Transactions on Software Engineering, v. 13, n. 3, p. 335-343.

KEMERER, C. F.; SLAUGHTER, S. (1999) An empirical approach to studying software evolution. IEEE Trans. Softw. Eng., IEEE Press, NJ, USA, v. 25, n. 4.

KOSCIANSKI, A.; SOARES, M. S. (2007) “Qualidade de Software - 2a edição”. São Paulo: Novatec Editora. ISBN 978-85-7522-112-9.

KUIPERS, T.; VISSER, J. (2007) “Maintainability index revisited”. In: Proc. of 11th European Conf. on SW Maintenance and Reeng. - CSMR. IEEE Computer Society.

LEHMAN, M. M. (1980) “On understanding laws, evolution and conservation in the large-program life cycle”. J. Syst. Software, v. 1, n. 3, p. 213-232.

MARTIN, R. C. “Object Oriented Design Quality Metrics - an Analysis of Dependencies”. October 1994. Disponível em: <http://www.objectmentor.com/resources/articles/oodmetrc.pdf>.

MCCABE, T. (1976) “A complexity measure”. IEEE Trans. Software Eng., v.2, p.308.

MCCABE, T.; WATSON, A. (1994) Software complexity. Crosstalk, v. 7, n. 12, p. 5-9.

MCCALL, J.; RICHARDS, P.; WALTERS, G. (1977) “Factors in software quality”. Hanscom AFB, MA. v. 1-3.

MOREIRA, G. S. P.; MELLADO, R. P.; MONTINI, D.; DIAS, L. A. V.; CUNHA, A. M. (2010) “Software product measurement and analysis in a continuous integration environment”. In: Proc. 7th Information Technology: New Generations (ITNG).

NDEPEND - Metrics Definition. Acessado em Março de 2011. Disponível em <http://www.ndepend.com/metrics.aspx>.

OMAN, P.; HAGEMEISTER, J. (1992) “Metrics for assessing a software system maintainability”. In: Proc. of Conference on Software Maintenance. p. 337-344

PRESSMAN, R. S. “Software Engineering: A Practitioner's Approach”, 7th International edition. New York, NY, USA: McGraw-Hill, 2009. ISBN 0071267824.

SEI (1997) “Handbook - C4 Software Technology - Reference Guide - A Prototype”.

SDMETRICS. “Semantic Designs Metrics extractor for C# Source Code”. Disponível em <http://www.semdesigns.com/Products/Metrics/CSharpMetrics.html>

WARE, M. P.; WILKIE, F. G.; SHAPCOTT, M. (2007) “The application of product measures in directing software maintenance activity”. J. Softw. Maint. Evol., John Wiley & Sons, Inc., New York, NY, USA, v. 19, n. 2, p. 133-154. ISSN 1532-060X.

WELKER, K. (2001) “The software maintainability index revisited”. CROSSTALK - The Journal of Defense Software Engineering, Agosto, 2001.

WELKER, K.; OMAN, P. (1997) “Development and Application of an Automated Source Code Maintainability Index”, J Softw Maint Evol - R, 9, pp. 127 – 159.