How to Automatically Collect Oriented Object Metrics: A Study Based on Systematic Review

Moshe Ribeiro, Rodrigo Quites Reis, Antônio J. G. Abelém
Programa de Pós Graduação em Ciência da Computação
Universidade Federal do Pará (UFPA)
Belém, Pará - Brasil
{moshe quites, abelem}@ufpa.br

Abstract— **Aim: Getting information to automatically collect object oriented metrics (OO metrics) in order to assist the comprehension and assessment of software products. Method: It was developed a study based on a systematic review and 37 primary studies were selected from 577 papers retrieved in 3 databases. Result: 177 metrics that can be automatically collected were cataloged. Besides, 27 from such total were the most referenced. The cataloged metrics were classified according to the quality characteristics which were related; 18 collection tools have been identified. This way, it was concluded that there is a set of common procedures for collecting OO metrics and the Java and C++ are the languages with the largest number of tools on which is possible to extract metrics.**

Keywords— software; quality; metrics; oriented; object; systematic; review

I. INTRODUÇÃO

Medições podem ser utilizadas como um importante instrumento para auxiliar os profissionais da Engenharia de Software a avaliar e monitorar a qualidade dos produtos desenvolvidos, possibilitando apoio à tomada de decisão em projetos de software [10]. Nesse contexto, diversas métricas orientadas a objeto (métricas OO, doravante), extraídas a partir da análise estática do código fonte de programas orientados a objeto, têm sido propostas e utilizadas para auxiliar a entender a estrutura do software, avaliar a qualidade, estimar complexidade, predizer custo/esforço e controlar/melhorar o processo [2] [4]. Isso pode ser muito valioso em situações que o código fonte é o único (ou principal) artefato disponível do software, como é o caso de muitos sistemas legados e softwares livres.

Estudos recentes encontraram evidências da relação de métricas OO para predizer/avaliar a qualidade do produto de software - que no contexto desses artigos é o código fonte [6] [7]. A consequência disso, é que a identificação/predição de classes propensas a falhas ou de difícil manutenção, por exemplo, em estágios iniciais do desenvolvimento do software, poderia ajudar uma organização a focar em atividades de melhoria da qualidade que, por sua vez, poderia reduzir os esforços para manutenções futuras [6] [7].

Diante disso, profissionais e/ou pesquisadores, que desejam compreender/avaliar o produto de software a partir da análise estática do código fonte, podem se deparar com as seguintes questões: Quais métricas OO de código fonte são utilizadas? Quais características de qualidade são avaliadas por essas métricas? Quais ferramentas são usadas para coletar essas métricas? Para responder essas perguntas, este trabalho foi desenvolvido com objetivo de contribuir com informações pertinentes para quem quer compreender/avaliar o produto de software a partir da análise estática do código fonte através de métricas OO. Com ênfase na coleta automática dessas métricas devido ao grande esforço que seria extraí-las manualmente (em decorrência do elevado volume de informações inerentes ao código). Nesse cenário, o apoio de ferramentas que automatizem essa tarefa é fundamental.

Para alcançar um grau de rigor científico adequado, este estudo foi realizado na forma de uma revisão sistemática [8] com objetivo de reduzir o viés de uma pesquisa informal. O restante do artigo está organizado da seguinte forma: na seção II os trabalhos relacionados são comentados; em seguida, na seção III a fase de planejamento da revisão é apresentada através do detalhamento do protocolo utilizado; na seção IV é descrita a etapa de condução e os resultados obtidos; na seção V ocorre a análise dos resultados; na seção VI as ameaças à validade do trabalho são discutidas e na seção VII são apresentadas as considerações finais.

II. TRABALHOS RELACIONADOS

O trabalho de Abilio et. al [1] publicou uma revisão sistemática da literatura sobre métricas contemporâneas relacionadas com manutenibilidade de sistemas. Como resultado obtiveram um catálogo de métricas das quais: 33 estavam relacionadas com características e 78 métricas relacionadas com interesses, totalizando 111 métricas contemporâneas. As métricas relacionadas com características foram utilizadas para mensurar 9 propriedades (por exemplo, *Scattering* e *Coupling*) e as métricas de interesses foram relacionadas com 24 propriedades (por exemplo, *Coupling* e *Separation of Concerns*). Ao final, o estudo sugere a existência de um conjunto extenso de métricas relacionadas com interesses e de um conjunto mais restrito com características.

No estudo de Saraiva *et al.* [11] foi realizado um mapeamento sistemático para identificar métricas orientadas a aspectos ligadas à manutenibilidade de software. Como resultado foram obtidas 67 métricas orientadas a aspecto e 469 do paradigma orientado a objeto que, segundo o referido trabalho, podem ser adaptadas a orientação a aspectos. Somente 15% das métricas foram citadas por mais de um dos 138 estudos primários selecionados, o que levou os autores a

978-1-4673-9143-6/15/$31.00

inferir que poucos estudos utilizam métricas anteriormente propostas.

Uma das principais diferenças entre o trabalho aqui apresentado e os demais estudos citados, reside no fato deste não se limitar em identificar, catalogar e classificar métricas OO de código-fonte, mas também obter informações relacionadas com a utilização prática dessas métricas, principalmente as referentes à coleta delas. Para isso, buscaram-se as respostas para questões como: a) Quais ferramentas podem ser utilizadas para coletar métricas OO? b) Quais recursos elas oferecem? Elas são integradas com outras ferramentas utilizadas ao longo do processo de desenvolvimento de software? As respostas para essas questões podem contribuir para utilização prática de métricas OO.

III. PROTOCOLO

Baseado no trabalho de Kitchenham *et al.* [8], as seções seguintes detalham cada item do protocolo utilizado neste estudo que segue todos os procedimentos de uma revisão sistemática. Contudo, por conta de restrições de tempo, não são utilizados dados de comparação; a quantidade de bases de dados a serem utilizadas foi limitada e simplificou-se a estratégia de PICO [8], usada para definir a *string* de busca.

A. Questão de Pesquisa e Dimensão

A tabela I mostra as questões e na II as dimensões desta revisão.

TABELA I. ORGANIZAÇÃO DAS QUESTÕES

Item	Descrição	Justificativa
Q 01	Quais são as métricas OO de código fonte coletadas automaticamente?	Identificar as métricas OO para as quais existem ferramentas capazes de coletá-las automaticamente, a fim de apoiar a escolha e uso dessas métricas.
Q 01.1	Quais características de qualidade podem ser avaliadas pelas métricas OO identificadas?	Classificar as métricas de acordo com as características de qualidade que elas estão relacionadas, com intuito de apoiar a escolha das métricas a serem utilizadas em um plano de medição conforme as características de qualidade que se pretende avaliar.
Q 01.2	Quais são as métricas mais frequentes?	Verificar quais das métricas OO identificadas são mais utilizadas.
Q 02	Quais são as ferramentas para coleta automática de métricas de código fonte e como realizam essa tarefa?	Fornecer um catálogo com as ferramentas relatadas pela literatura capazes de coletar métricas OO automaticamente. Além disso, mostrar uma visão geral de como realizam essa tarefa com intuito de fornecer informações relevantes para quem desejar desenvolver a sua própria ferramenta de coleta.
Q 02.1	As ferramentas possuem integração com outras do processo de software?	Identificar ferramentas que podem ser integradas a outras de maneira a facilitar a utilização dessas informações ao longo do processo de desenvolvimento de software.
Q 02.2	Como as ferramentas apresentam as informações das métricas coletadas?	Identificar que tipos de recursos (tipos de gráficos, tabela) são utilizados pelas ferramentas para apresentarem os dados da coleta, de maneira a facilitar a análise desses resultados por parte dos interessados.
Q 02.3	Para quais linguagens de programação as ferramentas permitem a extração de métricas?	Identificar as linguagens para as quais as ferramentas conseguem coletar as métricas.

TABELA II. DIMENSÕES DA REVISÃO

Item	Descrição
População	Métricas de código fonte.
Intervenção:	Ferramentas para coleta automática de métricas de código fonte e métricas coletadas automaticamente por elas.
Comparação:	Não há.
Resultados:	Ferramentas para coleta automática de métricas de código fonte e métricas de código fonte que podem ser coletadas automaticamente.

B. Definição da estratégia de busca

Segundo Mafra e Travassos [9] uma revisão da literatura realizada sem uma expressão de busca (*string* de busca) pré-definida pode ser conduzida por interesses pessoais de seus pesquisadores, levando a resultados pouco confiáveis. Por isso, foi criada uma expressão de busca na qual foram definidos os principais termos a serem utilizados na composição dessa expressão. Para isso, adotou-se a estratégia de PICO, que consiste em definir os termos da *string* de busca a partir das dimensões população, intervenção, comparação e resultados (*outcomes*). Como dito anteriormente, neste trabalho essa estratégia foi utilizada de maneira simplificada e por isso não foram utilizadas todas as dimensões.

Restrições de tempo para o desenvolvimento deste trabalho limitaram a quantidade de bases de dados a serem utilizadas na revisão. Por isso, foram selecionadas Scopus, Ei Compendenx e IEEE, pois possuem uma cobertura de estudos relevantes [3] [11] e permitem exportar os resultados da aplicação da *string* de busca, pesquisa por palavras-chave e expressões lógicas.

A expressão de busca foi ajustada para a máquina de busca Scopus na qual foi adicionada a condição TITLE-ABS-KEY (tabela III), a qual restringe a aplicação da *string* de busca ao título, resumo e palavras chave com objetivo de obter resultados mais satisfatórios. Além disso, verificou-se que muitos estudos relacionados com métricas de código fonte de linguagens orientadas a objeto utilizam a denominação métricas orientadas a objeto (*object oriented metrics*). Por isso, na expressão de busca foi adicionado esse termo.

TABELA III. EXPRESSÃO DE BUSCA DEFINIDA

Expressão de Busca
TITLE-ABS-KEY((software) AND ((metric) OR (metrics) OR (measurement) OR (measure) OR (measures)) AND ((source code) OR (source codes) OR (source-code) OR (object-oriented) OR (object oriented) OR (objectoriented)) AND ((collect) OR (collected) OR (collects) OR (collection) OR (extraction) OR (extract) OR (gather) OR (gathered) OR (support) OR (calculated) OR (calculation)) AND ((automatic) OR (automated) OR (automatically)) AND ((process) OR (processes) OR (approach) OR (approaches) OR (method) OR (technique) OR (methodology) OR (strategy) OR (framework) OR (tool) OR (tools) OR (plugin) OR (toolkit)))

C. *Procedimentos e critérios para seleção dos estudos*

A tabela IV apresenta os critérios de inclusão e os de exclusão. Vale ressaltar que os critérios definidos buscaram selecionar trabalhos que abordassem métricas que podem ser aplicadas em linguagens orientadas a objeto. O foco é nas linguagens que são exclusivas desse paradigma, mas não se limitando a elas.

TABELA IV. INCLUSÃO E EXCLUSÃO DE ESTUDOS CRITÉRIOS

Código	Critério
INC 01	Apresenta ferramenta para coleta automática de métricas de código fonte;
INC 02	Apresenta métricas de código fonte que podem ser coletadas automaticamente;
EXC 01	Não responde a qualquer uma das questões estabelecidas;
EXC 02	Extração das métricas OO utilizando método diferente da análise estática do código fonte.
EXC 03	Publicação do mesmo autor, mesmo conteúdo, no entanto com título diferente;
EXC 04	Publicações referentes a resumo de simpósio e conferência (*Proceedings / Symposium / Conference*);
EXC 05	Publicações que impliquem em custos financeiros para ter acesso ao seu conteúdo;

O procedimento para seleção dos estudos foi dividido em três fases conforme descrição a seguir:

1) Aplicação da expressão de busca em todas as fontes escolhidas, os resultados obtidos foram catalogados e armazenados extraindo-se as seguintes informações dos artigos: Título da publicação, Autor(es), Ano da publicação, Fonte da publicação, Resumo/Abstract.

2) Os resultados catalogados após a execução da fase I passaram por uma nova seleção na qual seus resumos/abstracts foram avaliados conforme os critérios de inclusão e exclusão definidos. Caso o estudo não atendesse ao objetivo da revisão, os estudos eram excluídos da lista dos estudos catalogados após a execução das seguintes ações: 1) Identificação de quais critérios de exclusão foram utilizados para excluir o estudo; 2) Armazenamento das publicações excluídas.

3) Nos estudos selecionados na fase II, realizou-se uma análise do conteúdo através da leitura na íntegra de cada artigo e aplicação dos critérios de inclusão e exclusão definidos. Os estudos excluídos nessa fase passaram pelos mesmos procedimentos de exclusão da fase II. Os artigos selecionados para a lista final passaram pelos seguintes procedimentos: 1) Identificação da(s) questão/questões que o estudo responde; 2) Identificação dos critérios de inclusão utilizados para selecionar o estudo.

Para cada artigo aprovado pelo processo de seleção completo, foram extraídos os dados apresentados na tabela V, conforme a questão que o artigo responde. Utiliza-se um X para sinalizar quais informações serão extraídas quando o artigo responder a uma determinada questão.

TABELA V. INFORMAÇÕES EXTRAÍDAS

Informação a ser Extraída	Q01	Q02
Titulo, Autores, Fonte, Abstract, Ano	X	X
Identificação da ferramenta utilizada para coletar métricas de código fonte automaticamente	X	
Descrição da ferramenta	X	
Identificação das métricas de código fonte		X
Descrição das métricas de código fonte		X

IV. CONDUÇÃO

Nesta seção são apresentados os resultados, sumarizados, obtidos em cada fase do estudo (descritas na seção III, item C) bem como a definição dos procedimentos adotados. Na fase I, a *string* de busca foi executada nas bibliotecas digitais escolhidas. Foram recuperados 577 estudos, incluindo os duplicados, conforme apresentado na tabela VI.

TABELA VI. RESULTADOS DA FASE1

Estudos	Compendex	Scopus	IEEE
Recuperados	185	207	185
Total	577		

Para os estudos recuperados pela expressão de busca, foram lidos todos os resumos conforme os procedimentos definidos na fase II. A tabela VII sintetiza os resultados obtidos nesta fase.

TABELA VII. RESULTADOS DA FASE II

Estudos após a execução da fase II	COMPENDEX	SCOPUS	IEEE	Total
Excluídos na fase II	27	117	110	254
Não foi possível recuperar o estudo completo	2	22	2	26
Selecionados após a execução da fase II	1	40	42	83
Estudos em duplicidade nas bases	214			214
	Total de Estudos			577

Conforme tabela VII, na fase II foram excluídos 254 estudos e não foi possível acessar o conteúdo completo sem custos de 26, isto é, disponíveis para uma instituição brasileira com acesso ao Portal de Periódicos CAPES, o qual permite o acesso a publicações de várias editoras internacionais. Por isso, foram excluídos conforme critério de exclusão EXC 05. Um total de 214 estudos foram excluídos por estarem duplicados nas bases. Restaram assim, 83 estudos que foram submetidos à fase III. Nesta fase, os 83 estudos selecionados na fase anterior foram lidos na íntegra aplicando-se os critérios de inclusão e

exclusão. Assim, os seguintes resultados foram obtidos: 1) 46 estudos excluídos pelos critérios de exclusão; 2) 37 estudos selecionados e seus dados extraídos conforme descrito na seção III, item C.

V. ANÁLISE DOS RESULTADOS

A lista contendo os estudos selecionados nesta revisão está disponível no Apêndice 1 – Lista dos Artigos Selecionados, no final deste artigo, o qual totalizou 37 trabalhos ordenados alfabeticamente pelo nome dos autores.

A distribuição desses trabalhos ao longo dos anos é apresentada no gráfico da figura 1 e nela é possível visualizar a quantidade de artigos que focaram na criação/validação de ferramentas de coleta de métricas (legenda "Ferramenta"), bem como os estudos que focaram em propor/utilizar métricas de código fonte para um determinado fim (legenda "Métrica"), como por exemplo: identificar oportunidades de reuso e refatoração; determinar quais classes são propensas à falha ou de difícil manutenção; detectar violações de projeto de arquitetura, entre outros.

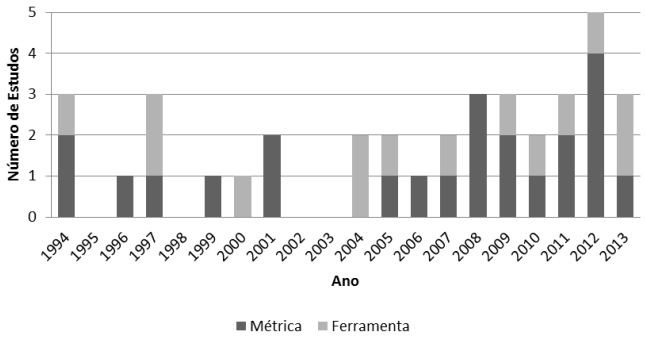

Figura 1 - Distribuição dos Estudos por Ano

Pode-se observar que a maioria dos estudos 62% (23) concentraram seus esforços em propor/utilizar métricas de código fonte enquanto que os outros 38% (14) em criar/validar ferramentas. Isso pode ser um indício que analisar, criar modelos e sugerir novas métricas são questões mais relevantes que desenvolver e/ou validar ferramentas de coleta para os estudos selecionados.

A. *Questão 01: Quais são as métricas OO coletadas automaticamente?*

Ao longo da revisão foram catalogadas 177 métricas de código fonte que estão disponíveis no Apêndice 2 – Lista das Métricas Identificadas.

Vale ressaltar que algumas métricas (do Apêndice 2 e da figura 3) são marcadas com um *. Isso significa que há variações dessas métricas e para este trabalho foram consideradas como uma única, com objetivo de agrupar os resultados relacionados a elas e assim permitir uma análise mais consistente dos resultados. Como exemplo, podemos citar a métrica *Lines of Code* (LOC): em alguns artigos era calculada levando em consideração linhas em branco e em outros não.

Ainda assim, neste trabalho foram identificadas 119 métricas citadas por um único artigo, um número relativamente alto, pois corresponde a aproximadamente 67% do total de métricas catalogadas (177). Nesses casos, verificou-se que a maioria desses artigos utilizavam métricas muito específicas para um determinado fim [A.3] [A.21] [A.22] e que dificilmente poderiam ser utilizadas para outro objetivo ou contexto. Por exemplo, Mayrand e Leblanc [A.21] apresentaram uma técnica para identificar o grau de clonagem (cópia) de funções. Para isso usaram 18 métricas que não foram referenciadas por nenhum outro artigo desta revisão, tais como: VarLenAvg *(Average variable name length)*, StrBrcNbr *(Number of breaches of structure)*, entre outras. Esse cenário poderia explicar o número relativamente alto de métricas citadas uma única vez, o que também ajudou a elevar a quantidade total de métricas catalogadas.

Dessa forma, pode ser que o conjunto de métricas de código úteis para avaliar a qualidade do produto de software, de maneira mais genérica, seja bem menor em relação ao total obtido (177). Isso é reforçado pela situação retratada na seção V, item C a qual destaca 27 métricas como as mais citadas.

B. *Questão 01.1: Quais características de qualidade podem ser avaliadas pelas métricas OO identificadas?*

Samoladas *et al*. [A.29] elaborou um modelo de qualidade baseado na ISO/IEC 25020 [5] no qual mapeou métricas de código fonte às subcaracterísticas: 1) Analisabilidade: caracteriza a capacidade de identificar a causa raiz de uma falha no software; 2) Modificabilidade: Caracteriza a quantidade de esforço para modificar o comportamento de um sistema; 3) Estabilidade: Caracteriza a capacidade do software de evitar impacto negativo decorrente de modificações efetuadas; 4) Testabilidade: Caracteriza o esforço necessário para testar uma mudança no sistema modificado. Todas essas subcaracterísticas estão relacionadas com a característica manutenibilidade [5].

No que se refere à manutenibilidade, o estudo [A.29] objetivou coletar métricas que dependam somente da análise do código fonte: ou seja, sem a necessidade de coletar métricas de código fonte e relacioná-las com outros artefatos do processo de desenvolvimento de software. Por isso, a subcaracterística conformidade não foi utilizada, pois a mesma trata de padronização, politicas e normas de um projeto.

Sendo assim, em [A.29] foram mapeadas métricas de código fonte entre as quatro subcaracterísticas apresentadas, sendo que algumas métricas constam em mais de uma subcaracterística. A partir desse mapeamento, as métricas identificadas neste trabalho foram classificadas de acordo com as subcaracterísticas de manutenibilidade da ISO/IEC [5], conforme o seguinte procedimento: observou-se que as métricas relacionadas com acoplamento eram vinculadas a Modificabilidade e Estabilidade em [A.29]. Dessa forma, todas as métricas de acoplamento identificadas neste trabalho também foram mapeadas da mesma maneira. Esse processo se repetiu para as métricas relacionadas à complexidade, coesão e tamanho. Finalmente, para 8 métricas não foi possível o mapeamento utilizando essa estratégia (como pode ser visto no

Apêndice 2). O resultado desse mapeamento pode ser visualizado na figura 2 e no Apêndice 2.

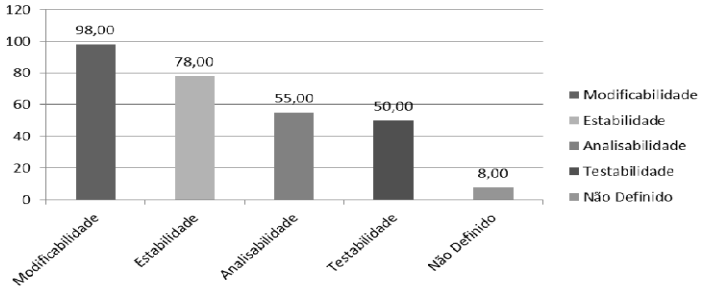

Figura 2 – Quantidade de Métrica por Subcaracterística de Qualidade

Na figura 2 pode-se notar a grande ocorrência de métricas relacionadas com modificabilidade (54%) e estabilidade (43%). Isto pode ser um indício que essas duas subcaracterísticas sejam importantes aspectos a serem considerados na avaliação da qualidade do software. Na sequência, aparecem analisabilidade (30%), testabilidade (27%) e, finalmente, o grupo das métricas não definidas, que no caso não foi possível mapeá-las (4%). No Apêndice 2 é possível visualizar as subcaracterística que cada métrica está relacionada.

C. *Questão 01.2: Quais são as métricas mais frequentes?*

O gráfico da figura 3 destaca as 27 métricas mais referenciadas pelos estudos selecionados neste trabalho. Dentre elas pode-se destacar o conjunto de seis métricas OO proposto por Chidamber e Kemerer [A.10] (DIT, NOC, CBO, LCOM, RFC, WMC). Elas aparecem entre as 10 métricas mais citadas.

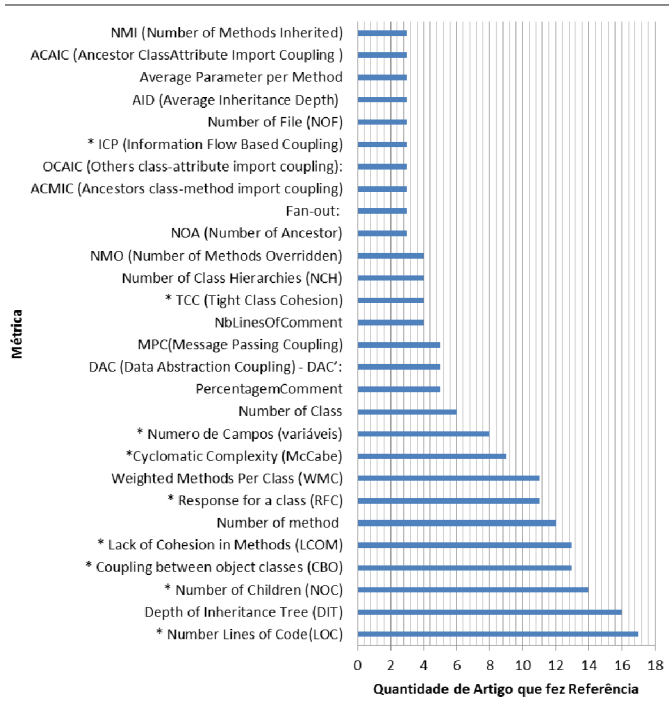

Figura 3 – Métricas mais Referenciadas

Em média 60% das métricas coletadas pelas ferramentas, identificadas neste trabalho (ver tabela VIII), estão entre essas 27 mais referenciadas. Além disso, todos os trabalhos [A.8] [A.28] [A.33] [A.31] [A.16] que buscaram avaliar e/ou validar métricas empiricamente também utilizaram métricas que estão entre essas 27. Nesse cenário, pode-se notar a importância delas para a qualidade do produto de software.

D. *Questão 02: Quais são as ferramentas para coleta automática de métricas de código fonte e como realizam essa tarefa?*

As ferramentas identificadas nos trabalhos selecionados estão listadas na tabela VIII. Nela é possível visualizar: o ano em que o artigo que a mencionou foi publicado; a referência ao artigo; o nome da ferramenta; o contexto em que o desenvolvimento se deu (se foi na academia, indústria, governo, ou em parceria entre eles); as linguagens de programação para as quais a ferramenta consegue extrair métricas; por último, as métricas coletadas por elas. Algumas informações não estavam disponíveis em todos os artigos, como o nome da ferramenta e as linguagens de programação que elas conseguem extrair métricas. Nesses casos, um traço (-) foi utilizado para representar a ausência dessas informações. Cada métrica que a ferramenta consegue coletar está representada por um número com o qual se pode identificar a métrica correspondente no Apêndice 2.

Nenhuma das ferramentas desenvolvidas no ambiente acadêmico, ou em parceria com o mesmo (como JBOOMT [A.37], Columbus Framework [A.12]), mencionou a disponibilização das mesmas para uma eventual utilização por terceiros. Por outro lado, alguns artigos que utilizaram ferramentas da indústria disponibilizaram acesso a elas. No entanto, somente o Eclipse Metrics Plug-in pode ser utilizado de forma gratuita, pois adota licença CPL1.0. Os outros trabalhos disponibilizam somente versões para experimentação que duram em média 15 dias, após esse tempo será necessário adquirir a licença do produto, mediante pagamento. Os artigos [A.19] e [A.35] relataram a utilização de mais de uma ferramenta para coleta de métricas, porém não fornecem qualquer informação sobre o porquê dessa decisão.

A ferramenta SAIL apresentada no artigo de Marinescu *et al.* [A.20] não teve nenhuma métrica vinculada a ela (tabela VIII), pois possui uma linguagem própria para definição de métricas que permite criar consultas a uma base de dados relacional que contém informações sobre o código fonte, tais como: pacotes, classes e métodos. Isto possibilita o cálculo de métricas como: Número de classes, Número de métodos, entre outras.

TABELA VIII. FERRAMENTAS IDENTIFICADAS

Ano	Ref	Nome	Contexto	Linguagem	Métrica
1994	A.9	-	Academia	C++	2, 3, 4, 7, 17
1996	A.21	DatrixTM	Indústria	C++	57, 58, 59, 77, 85, 87
1997	A.14	OOMetTool	Academia	C++, SamallTalk e Delphi	1, 2, 3, 6, 7, 8, 11, 38, 53
1997	A.35	UX-Metric e PC-Metric	Indústria	Fortran e C++	1, 9, 15, 42, 43, 35, 71, 73,

Ano	Ref	Nome	Contexto	Linguagem	Métrica
					74, 75, 76, 119
1997	A.6	QMOOD++	Academia	C++	2, 3, 6, 10, 11, 13, 17, 19, 25, 26, 33, 35, 36, 37, 38, 41, 44, 50, 52, 120, 122, 124, 125, 127, 128, 131, 163, 172, 173, 174, 175
2000	A.37	JBOOMT	Academia Indústria Governo	C++	1, 2, 3, 4, 5, 6, 7, 8, 11, 24, 26, 34
2001	A.28	AdaSTAT	-	-	1, 9, 42, 43, 56, 149, 151, 152, 153
2001	A.33	-	Academia	SmallTalk	1, 2, 6, 28, 106, 107, 108, 138
2004	A.32	WebMetrics	Academia	C, C++. Java e SmallTalk	2, 3, 4, 5, 7, 8
2004	A.12	Columbus Framework	Academia Indústria	C++	1, 2, 3, 4, 5, 6, 7, 8, 10, 11
2005	A.20	Sail	Academia	-	-
2007	A.31	Eclipse Metrics plug-in	Industria	Java	1, 2, 3, 4, 5, 8, 46, 48
2009	A.4	-	Academia	C++ e Java	2, 3, 4, 5, 6, 10, 11, 18, 19, 21, 22, 24, 25, 26, 27, 28, 34, 40, 47, 49, 50, 53, 54, 55, 63, 68, 69, 115
2009	A.23	IPLASMA	-	Java	2, 6, 39, 44, 55, 118, 129, 130, 147
2010	A.25	Ndepend	Indústria	.NET	1, 2, 3, 5, 6, 9, 10, 12, 15, 45, 46, 48, 105, 110, 111
2011	A.11	E-Quaity	Academia	Java	1, 2, 3, 4, 5, 6, 7, 8, 10, 18, 51, 136, 137, 139, 140, 141
2012	A.17	-	Academia	C++	1, 6, 8, 11, 12, 15, 20, 24, 145, 146
2013	A.19	Stan, Lattix e Understand	Industria	C e Java	1, 2, 3, 4, 5, 6, 8, 9, 10, 109

Em todos os estudos selecionados algum tipo de ferramenta foi utilizada para coletar automaticamente as métricas. No entanto, alguns estudos focaram na análise dos resultados das métricas e outros em como elas são coletadas. Na figura 4, essa situação é evidenciada já que 20 (54%) artigos não fornecem qualquer informação sobre como a ferramenta coleta a métrica, apenas mencionam que algum componente realizou essa tarefa.

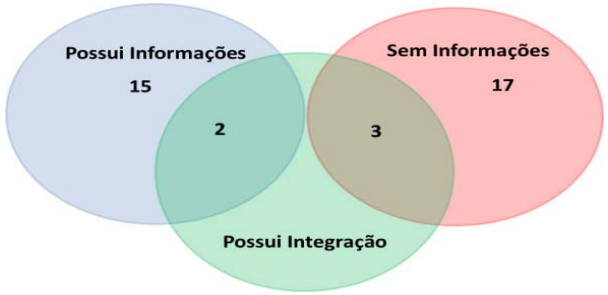

Figura 4 - Distribuição dos Estudos com case nas informações sobre a ferramenta de Coleta utilizada

Conforme figura 4, 17 estudos descrevem a ferramenta utilizada para coletar métrica. Nesse cenário notou-se a recorrência dos seguintes procedimentos para executar essa tarefa: 1) Identificar informações específicas da sintaxe da linguagem em que o código analisado foi escrito; 2) Armazenar as informações relevantes identificadas pelo procedimento anterior; 3) Calcular as métricas por meio de consulta às informações armazenadas pelo procedimento número 2; 4) Apresentação dos resultados das métricas coletadas.

A partir dos procedimentos recorrentes identificados nas ferramentas, elaborou-se um fluxo, apresentado na figura 5, que ilustra as camadas responsáveis por realizar cada procedimento mencionado. Destaca-se o ponto em que há uma variação no meio utilizado para armazenar as informações relevantes do código.

As camadas responsáveis por realizar os procedimentos de 1 a 4 são denominadas, respectivamente: Parser, Repositório de Informações, Cálculo e Apresentação.

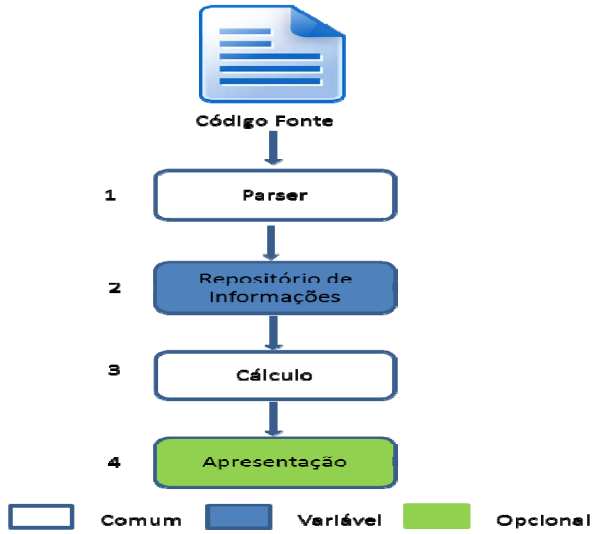

Figura 5 - Componentes das Ferramentas de Coleta de Métricas

Ao longo da revisão também foram identificados dois tipos de estruturas para armazenar as informações extraídas do código (na camada Repositório de Informações): 1) *Abstract Syntax Tree*: É uma abstração do código fonte a qual é baseada em uma árvore de representação dos *tokens* utilizados no código fonte. Dessa maneira, há uma representação do código fonte de forma exata; 2) Banco de Dados: Armazena em um modelo Relacional o conjunto de informações relevantes do código fonte tais como classes, métodos entre outros.

Dos 17 artigos que apresentaram informações sobre as suas ferramentas de coleta, 14 utilizam repositórios de informações modelados em um banco de dados relacional enquanto 2 utilizam *Abstract Syntax Tree* e [A.1] não deixou claro como armazena essas informações. Todas as ferramentas que utilizam um banco de dados como repositório de informações coletam métricas somente para linguagens orientadas a objeto. Por outro lado, dos 2 artigos que utilizam *Abstract Syntax Tree*: o estudo [A.6] realiza a coleta para linguagens orientada a objeto enquanto que em [A.26] isto é feito para linguagens procedurais.

E. *Questão 02.1: As ferramentas possuem integração com outras do processo de software?*

Cinco artigos (figura 4) relataram possuir integração com outras ferramentas utilizadas no processo de desenvolvimento de software. Os artigos [A.11] [A.22] [A.26] e [A.31] relatam trabalhos que foram implementados como um *plugin* do ambiente de desenvolvimento integrado Eclipse. No entanto, não fornecem detalhes de como ocorre essa integração, apenas relatam que a ferramenta pode ser utilizada a partir desse ambiente. O estudo [A.37] apresenta a ferramenta JBOOMT responsável pela coleta automática das métricas de código fonte e é integrada à suíte JBPAS a qual possui outras ferramentas para documentar o código e dar suporte às atividades de testes. Porém, também não fornece detalhes sobre a forma que essa integração ocorre.

Diante disso, pode-se identificar a carência de integração entre as ferramentas de coleta de métricas de código fonte e as outras utilizadas ao longo do processo de desenvolvimento software.

F. *Questão 02.2: Como as ferramentas apresentam as informações das métricas coletadas?*

As ferramentas identificadas nesta revisão utilizam, na camada de apresentação, basicamente dois tipos de visualizações dos resultados das métricas coletadas: uma através de tabelas e outra por meio de gráficos. Cerca de 40% (16) dos estudos empregam tabelas para apresentarem os dados coletados. Aproximadamente 30% (11) usam gráficos de dispersão, kiviat ou criam sua própria notação; pouco mais de 10% (4) fazem uso de histogramas e 16% (6) utilizam gráficos de linhas e outros.

Alguns estudos utilizam tabelas, para apresentar os resultados da coleta, para permitir a comparação dos valores das métricas de várias versões do software ao longo do tempo [A.3] [A.4] [A.25] [A.29] ou com valores como: mínimo, máximo, média entre outros valores obtidos por meio da coleta [A.12] [A.16] [A.18] [A.33]. Nesse caso, esses estudos não se limitam a questão de como coletar as métricas, mas também em como apresentá-las de modo a facilitar a sua análise. Por outro lado, outros estudos simplesmente exibem os valores coletados das diversas métricas que a ferramenta é capaz de extrair [A.1] [A.5] [A.6] [A.9] [A.15] [A.20] [A.22] [A.32]. Esses artigos focam em como a ferramenta realiza a coleta/cálculo das métricas, ou na definição de novas métricas.

O gráfico de dispersão é aplicado por alguns trabalhos como [A.33], no qual analisa a relação entre as diferentes métricas para medir tamanho e reutilização, e em [A.28] que avalia a relação de métricas de complexidade com índice de correções do software.

Gráfico com notação própria, de kiviat, barras e outras representações visuais também são utilizadas [A.23] [A.27] [A.37]. Outros estudos [A.2] [A.13] [A.17] [A.26] [A.36] [A.34] não apresentaram de forma estruturada seus resultados, apenas os mencionaram ao longo do texto.

Histogramas são usados (nos estudos selecionados neste trabalho) para comparar valores de uma única métrica à medida que o número de classes do software aumentava [A.8] [A.10] e [A.12] para comparar valores entre duas métricas. Em [A.14] os valores da métrica DIT (*Depth of Inheritance Tree*) são analisados em relação a três linguagens OO diferentes, com objetivo de se avaliar como características particulares de cada linguagem e outros fatores, como a experiência do desenvolvedor, podem influenciar nessa métrica.

Outros artigos adotam gráficos de linhas para comparar valores de métricas entre projetos diferentes [A.16] [A.21] [A.31]. Os artigos [A.24] [A.35] utilizam o gráfico de linha para avaliar métricas ao longo do tempo, identificando as versões do software, enquanto que [A.18] realiza tarefa análoga, porém sem identificar as versões.

É possível notar o interesse dos pesquisadores por recursos visuais que apoiem a tarefa de análise dos resultados das métricas (por meio da representação gráfica desses resultados) já que diversos trabalhos foram realizados nesse sentido, como se pode observar ao longo deste item F.

G. *Questão 02.3: Para quais linguagens de programação as ferramentas permitem a extração de métricas?*

Conforme a figura 6, foram encontradas sete linguagens diferentes para as quais as ferramentas de coleta, identificadas nesta revisão, efetuaram a extração das métricas de código fonte. Outra informação importante é que somente as linguagens Fortran e C (nas versões nas quais os códigos utilizados nos estudos estavam escritos) não eram orientadas a objeto. Ou seja, quase 90% das ferramentas identificadas coletam métricas somente para linguagens orientadas a objeto. Na figura 6, também é possível notar que as linguagens que mais tiveram suporte para extração de métricas foram Java e C++, demonstrando a grande utilização dessas linguagens tanto no ambiente acadêmico quanto na indústria.

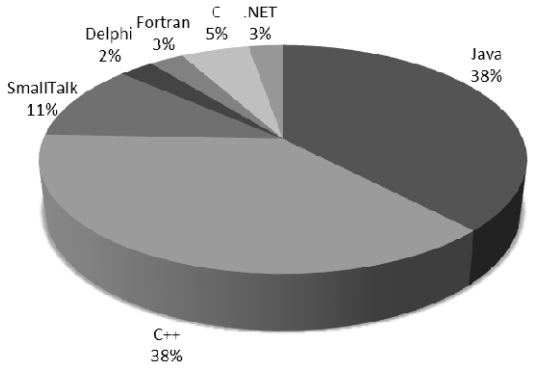

Figura 6 - Quantidade de Ferramentas que Coletam Métricas por Linguagem de Programação

VI. AMEAÇAS À VALIDADE

Uma das principais ameaças à validade para este trabalho é a seleção incompleta ou inadequada de estudos. Apesar dos autores seguirem uma abordagem sistemática de revisão, é possível que não tenham sido recuperados alguns estudos relevantes, sobretudo se eles foram publicados em fontes diferentes daquelas consideradas nesta revisão. Além disso, existe ainda a possibilidade de alguns artigos importantes terem utilizados termos diferentes dos que foram utilizados na *string* de busca, o que acarretaria na não recuperação desses estudos.

Outra ameaça, que merece destaque, está relacionada ao critério adotado, neste trabalho, para definir se uma métrica pode ser coletada automaticamente. Como dito anteriormente, somente artigos que afirmassem que uma ferramenta foi utilizada para coletar as métricas eram considerados na seleção. Esse critério pode ter causado a exclusão de diversos artigos que possuíam métricas que eram passíveis de serem coletadas automaticamente mas que, por algum motivo, no artigo, não foi mencionada a utilização de uma ferramenta para realizar essa tarefa.

Por fim, o procedimento utilizado para classificação das métricas, com relação às subcaracterísticas de qualidade da ISO [5], foi baseado no trabalho de Samoladas *et al.* [A.29] e isso pode ter limitado essa classificação já que em [A.29] foi restringida à característica de manutenibilidade.

VII. CONSIDERAÇÕES FINAIS

Este trabalho apresenta um estudo baseado em revisão sistemática o qual obteve como resultado: 177 métricas que podem ser coletadas automaticamente, das quais 27 foram as mais referenciadas; as métricas catalogadas foram classificadas conforme as características de qualidade as quais estavam relacionadas; 18 ferramentas de coleta foram identificadas e foi observada uma carência na integração dessas ferramentas com outras do processo de software; constatou-se que existem procedimentos comuns para se realizar a coleta de métricas OO e que Java e C++ são as linguagens mais visadas pelas ferramentas para coleta dessas métricas.

Uma das principais contribuições deste trabalho está no fato de não se limitar em identificar, catalogar e classificar métricas OO de código-fonte, mas também apresentar informações relacionadas com a utilização prática dessas métricas, principalmente as referentes à coleta delas. Pois apresentou as ferramentas capazes de realizar essa tarefa bem como os procedimentos aplicados para tal; os recursos oferecidos por elas e as linguagens de programação para as quais conseguem extrair métricas.

Como trabalho futuro, pretende-se ampliar o número de bases selecionadas na revisão e incluir, como por exemplo, o Google Acadêmico e o próprio Google para recuperar estudos que apresentem ferramentas de coleta frequentemente empregadas na indústria e que, possivelmente, não foram identificadas neste trabalho por não serem comumente utilizadas no ambiente acadêmico e/ou de pesquisa. Outro trabalho futuro vislumbrado consiste no desenvolvimento e avaliação de uma abordagem que apoie desde a escolha das métricas OO, de acordo com o objetivo da organização, até a análise dos resultados para tomada de decisão. Para isso podem ser levados em consideração estudos que foram realizados com objetivo de analisar empiricamente a efetividade no uso de métricas de código fonte para melhorar a qualidade do software. Além disso, há o potencial de se alcançar resultados mais interessantes ao se integrar todas estas informações coletadas com elementos do processo de software que produz os artefatos analisados.

AGRADECIMENTO

Os autores agradecem à CAPES e Universidade Federal do Pará pelo apoio.

REFERÊNCIAS

[1] Abílio, R., Teles, P., Costa, H., Figueiredo, E. (2012) "A Systematic Review of Contemporary Metrics for Software Maintainability", Simpósio Brasileiro de Componentes, Arquitetura e Reutilização de Software, SBCARS.

[2] Basili, V., Briand, L. and Melo, W. (1996) "A validation of object-oriented design metrics as quality indicators", IEEE Transactions on Software Engineering, Vol. 22, No. 10, pp.751-761.

[3] Brereton, P., Kitchenham, B., Budgen, D., Turner, M., Khalil, M. (2007) "Lessons from applying the systematic literature review process within the software engineering domain", J Syst Softw 80(4):571–583.

[4] Chidamber, R. and Kemerer, F. (1994) "A Metrics Suite for Object-Oriented Design", IEEE Trans.software Eng., Vol. 20, No. 6.

[5] ISO/IEC 25020 (2007), Software and System Engineering - Software Product Quality Requirements and Evaluation (SQuaRE) – Measurement Ref. Model and Guide.

[6] Isong, B., Obeten, E. (2013) "A Systematic Review of the Empirical Validation of Object-Oriented Metrics Towards Fault-Proneness Prediction", International Journal of Software Engineering and Knowledge Engineering.

[7] Jabangwe, R., Borstler, J., Smite, D., Wohlin, C. (2014) "Empirical evidence on the link between object-oriented measures and external quality attributes: a systematic literature review" Journal Empirical Software Engineering

[8] Kitchenham, B. et al. (2007) "The current state of evidence-based software engineering". Keynote Presentation. International Conference on Evaluation and Assessment in Software Engineering.

[9] Mafra, S. and Travassos, G. (2006) "Primary and Secondary Studies Supporting the Search for Evidence in Software Engineering." Relatório Técnico. COPPE, Universidade Federal do Rio de Janeiro.

[10] Pressman, R.S (2011) "Engenharia de Software: uma abordagem profissional", 7ª Edição, McGraw Hill do Brasil.

[11] Saraiva, J. et al. (2012) "Aspect-oriented software maintenance metrics: A systematic mapping study," in EASE'12: Proceedings of 16th International Conference on Evaluation and Assessment in Software Engineering.

APÊNDICE 1 – LISTA DOS ARTIGOS SELECIONADOS

[A.1] Aguero, M. and Madou, F. et al. (2010) "Enhancing source code metrics scope through artificial intelligence" IMETI 2010 - 3rd International Multi-Conference on Engineering and Technological Innovation, Proceedings.

[A.2] Ajmal, O. and Missen, M.M.S. et al. (2013) "EPlag: A two layer source code plagiarism detection system" Digital Information Management (ICDIM), Eighth International Conference on.

[A.3] Al Dallal, J. (2012) "Constructing models for predicting extract subclass refactoring opportunities using object-oriented quality metrics" Information and Software Technology.

[A.4] Alikacem, H. and Sahraoui, A. (2009) "A metric extraction framework based on a high-level description language" IEEE International Working Conference on Source Code Analysis and Manipulation, SCAM.

[A.5] Banker, D., and Kauffman, J. et al. (1994) "Automating output size and reuse metrics in a repository-based computer-aided software engineering (CASE) environment" Software Engineering, IEEE Transactions.

[A.6] Bansiya, J. and Davis, C. (1997) "Automated metrics and object-oriented development" Dr. Dobb's Journal.

[A.7] Bavota, G. and De Lucia, A. et al. (2012) "Supporting extract class refactoring in Eclipse: The ARIES Project" Software Engineering (ICSE), 2012 34th International Conference on.

[A.8] Briand, L. and Wust, J. et al. (1999) "Using coupling measurement for impact analysis in object-oriented systems" Software Maintenance.

[A.9] Brooks, C. and Buell, C. (1994) "A tool for automatically gathering object-oriented metrics" Aerospace and Electronics Conference.

[A.10] Chidamber, R. and Kemerer, F. (1994) "A Metrics Suite for Object-Oriented Design". IEEE Trans.software Eng., Vol. 20, No. 6.

[A.11] Erdemir, U. and Tekin, U. et al. (2011) "E-Quality: A graph based object oriented software quality visualization tool" Visualizing Software for Understanding and Analysis (VISSOFT).

[A.12] Ferenc, R. and Siket, et al. (2004) "Extracting facts from open source software" Software Maintenance.

[A.13] Hamou-Lhadj, A., Lethbridge, T. (2006) "Summarizing the content of large traces to facilitate the understanding of the behaviour of a software system" IEEE International Conference on Program Comprehension.

[A.14] Hericko, M. and Rozman, I. et al. (1997) "OO metrics data gathering environment" Technology of Object-Oriented Languages.

[A.15] Kaur, K., Singh, H. (2011) "Towards a valid metric for class cohesion at design level" International Conference on Emerging Applications of Information Technology, EAIT.

[A.16] Kayarvizhy, N. and Kanmani, S. (2011) "Analysis of quality of object oriented systems using object oriented metrics" Electronics Computer Technology (ICECT).

[A.17] Kozak, C. and Squire, M. (2011) "A secondary data archive for code-level debian metrics" International Workshop on Replication in Empirical Software Engineering Research, RESER.

[A.18] Kwiatkowsk, M. and Verhoef, C. (2013) "Recovering management information from source code" Science of Computer Programming.

[A.19] Lee, J. and Jung, W. (2013) "Automated metric visualizations for analyzing source code repositories" International Conference on Information Science and Applications, ICISA.

[A.20] Marinescu, C. and Marinescu, Ret al. (2005) "Towards a simplified implementation of object-oriented design metrics" Software Metrics.

[A.21] Mayrand, J. and Leblanc, C. (1996) "Experiment on the automatic detection of function clones in a software system using metrics" Software Maintenance.

[A.22] McQuillan, A. and Power, F. (2008) "A metamodel for the measurement of object-oriented systems: An analysis using alloy" Conference on Software Testing, Verification and Validation, ICST.

[A.23] Mihancea, F., Marinescu, R. (2009) "Discovering comprehension pitfalls in class hierarchies" Conference on Software Maintenance and Reengineering, CSMR.

[A.24] Mihancea, P. and Marinescu, R. (2005) "Towards the Optimization of Automatic Detection of Design Flaws in Object-Oriented Software Systems" Conference Software Maintenance and Reengineering.

[A.25] Pereira, M. and Mellado, R (2010) "Software product measurement and analysis in a continuous integration environment" International Conference on Information Technology: New Generations, ITNG.

[A.26] Plosch, R. and Gruber, H. et al. (2008) "Tool support for expert-centred code assessments" International Conference on Software Testing, Verification and Validation, ICST.

[A.27] Roubtsov, S. and Telea, A. (2007) "SQuAVisiT: A software quality assessment and visualisation toolset" IEEE International Working Conference on Source Code Analysis and Manipulation, SCAM.

[A.28] Saboe, M. (2001) "The use of software quality metrics in the materiel release process experience report" Quality Software.

[A.29] Samoladas, I. and Gousios, G. et al. (2008) "The SQO-OSS quality model: Measurement based open source software evaluation" International Federation for Information Processing, IFIP.

[A.30] Sandhu, P. and, Kaur, H. (2009) "Modeling of reusability of object oriented software system" World Academy of Science, Engineering and Technology.

[A.31] Sato, D. and Goldman, A. et al. (2005) "Tracking the evolution of object-oriented quality metrics on agile projects" Lecture Notes in Computer Scienc.

[A.32] Scotto, M. and Sillitti, A. (2004) "A Relational Approach to Software Metrics" Proceedings of the ACM Symposium on Applied Computing, 2, 1536 – 1540.

[A.33] Subramanian, G. and Corbin, W. (2001) "An empirical study of certain object-oriented software metrics" Journal of Systems and Software.

[A.34] Taibi, F. (2012) "Filtered mining in program code repositories" Information Retrieval Knowledge Management (CAMP), 2012 International Conference on.

[A.35] Welker, D. and Oman, W. et al. (1997) "Development and application of an automated source code maintainability index" Journal of Software Maintenance and Evolution.

[A.36] Xiaojing, W. and Contreras, A. et al. (2012) "Interval-based algorithms to extract fuzzy measures for Software Quality Assessment" Fuzzy Information Processing Society (NAFIPS), 2012 Annual Meeting of the North American.

[A.37] Xie, T. and Yuan, W. et al. (2000) "JBOOMT: Jade Bird Object-Oriented Metrics Tool" Chinese Journal of Electronics, 9, 202-20

APÊNDICE 2 – Lista das Métricas Identificadas

#	Métricas	Referência	Subcaracterísticas
1	* (LOC) Number Lines of Code	A.1,A.11,A.12,A.17,A.18,A.19, A.2,A.25, A.26, A.27, A.28, A.3, A.31, A.33, A.34, A.35, A.37	Analisabilidade, Testabilidade
2	(DIT) Depth of Inheritance Tree	A.10, A.11, A.12, A.14, A.22, A.23, A.25, A.29, A.30, A.31, A.32, A.33, A.37, A.4, A.6, A.9	Modificabilidade, Estabilidade
3	*(NOC) Number of Children	A.10,A.11,A.12,A.16,A.22,A.25, A.29,A.30,A.31,A.32, A.37, A.4, A.6, A.9	Estabilidade, Testabilidade
4	* (CBO) Coupling between object classes	A.10,A.11,A.12,A.16,A.22,A.29, .30,A.31,A.32,A.37,A.4,A.8, A.9	Modificabilidade, Estabilidade
5	* (LCOM) Lack of Cohesion in Methods	A.10,A.11,A.12,A.22,A.25,A.29, A.3,A.30, A.31, A.32, A.37, A.4, A.7	Modificabilidade
6	Number of method	A.11,A.12,A.17,A.2, A.23, A.25, A.3, A.33, A.36, A.37, A.4,A.6	Analisabilidade, Testabilidade
7	*(RFC) Response for a class	A.10, A.11, A.12, A.16, A.22, A.29, A.3, A.30, A.32, A.37, A.9	Testabilidade
8	(WMC) Weighted Methods Per Class	A.10,A.11,A.12,A.17,A.22,A.24, A.29, A.30, A.31, A.32, A.37	Analisabilidade
9	* (McCabe) Cyclomatic Complexity	A.18,A.19,A.2,A.25, A.27, A.28, A.29, A.34, A.35	Analisabilidade, Testabilidade
10	* Number of Field	A.11, A.12, A.16, A.2, A.25, A.3, A.4, A.6	Analisabilidade, Testabilidade
11	Number of Class	A.12, A.17, A.36, A.37, A.4, A.6	Analisabilidade, Testabilidade
12	Percentage of Comment	A.1, A.17, A.25, A.29, A.34	Analisabilidade
13	(DAC) Data Abstraction Coupling	A.16, A.22, A.3, A.6, A.8	Modificabilidade, Estabilidade
14	(MPC) Message Passing Coupling	A.16, A.22, A.3, A.7, A.8	Modificabilidade, Estabilidade
15	NbLinesOfComment	A.17, A.2, A.25, A.35	Analisabilidade
16	*(TCC) Tight Class Cohesion	A.16, A.22, A.24, A.3	Modificabilidade
17	(NCH) Number of Class Hierarchies	A.29, A.36, A.6, A.9	Analisabilidade
18	(NMO) Number of Methods Overridden	A.11, A.16, A.24, A.4	Modificabilidade, Estabilidade
19	(NOA) Number of Ancestor	A.16, A.4, A.6	Estabilidade, Testabilidade
20	(Fan-out) number of local flows out of that procedure plus the number of data structures that the procedure updates	A.13, A.17, A.20	Modificabilidade, Estabilidade
21	(ACMIC) Ancestors class-method import coupling	A.22, A.4, A.8	Modificabilidade, Estabilidade
22	(OCAIC) Others class-attribute import coupling	A.22, A.4,A.8	Modificabilidade, Estabilidade
23	*(ICP) Information Flow Based Coupling	A.16, A.22, A.8	Modificabilidade, Estabilidade
24	(NOF) Number of File	A.17, A.37, A.4	Analisabilidade, Testabilidade
25	(AID) Average Inheritance Depth	A.16, A.4, A.6	Modificabilidade, Estabilidade
26	Average Parameter per Method	A.37, A.4, A.6	Analisabilidade, Testabilidade
27	(ACAIC) Ancestor ClassAttribute Import Coupling	A.22, A.4, A.8	Modificabilidade, Estabilidade
28	(NMI) Number of Methods Inherited	A.16, A.33, A.4	Modificabilidade, Estabilidade
29	(OCMIC) Coupling to other classes, not related via inheritance. There is a Class-Attribute interaction between classes C and D, if C has an attribute of type D.	A.22,A.8	Modificabilidade, Estabilidade
30	(AMMIC) Coupling to ancestor classes. There is a Method-Method interaction between classes C and D, if C invokes a method of D, or if a method of class D is passed as parameter (function pointer) to a method of class C.	A.22,A.8	Modificabilidade, Estabilidade
31	(OMMIC) Coupling to other classes, i.e., not related via inheritance. There is a Method-Method interaction between classes C and D, if C invokes a method of D, or if a method of class D is passed as parameter (function pointer) to a method of class C.	A.22,A.8	Modificabilidade, Estabilidade
32	(OCMEC) Number of distinct classes used as types of the parameters of the methods in the class	A.22, A.3	Modificabilidade, Estabilidade
33	DAM (Data Acess Metric)	A.36, A.6	Modificabilidade, Estabilidade
34	Number of Module (NOM)	A.37, A.4	Analisabilidade, Testabilidade
35	DCC (Direct Class Coupling)	A.36, A.6	Modificabilidade, Estabilidade
36	CIS (Class Interface Size)	A.36, A.6	Modificabilidade
37	Number of Polymorphic Methods	A.36, A.6	Modificabilidade, Estabilidade
38	NPA (Number of PublicAttributes)	A.24, A.6	Modificabilidade, Estabilidade
39	Number of Calls (NOCALLS)	A.23, A.27	Modificabilidade, Estabilidade
40	NOD (Number Of Descendants)	A.16, A.4	Estabilidade, Testabilidade
41	ANA (A measure of generalization-specialization aspect of design)	A.36, A.6	Estabilidade, Testabilidade
42	Maintainability index	A.28, A.35	
43	program effort (E)	A.28, A.35	Analisabilidade, Testabilidade

44	* Average Derived Classes per Class	A.23, A.6	Estabilidade, Testabilidade
45	Numero de instruções	A.2, A.25	Analisabilidade
46	Afferent coupling at type level (TypeCa)	A.25, A.31	Modificabilidade, Estabilidade
47	DCMEC (Descendants class-method export coupling)	A.22, A.4	Modificabilidade, Estabilidade
48	Efferent coupling at type level (TypeCe)	A.25, A.31	Modificabilidade, Estabilidade
49	DCAEC (Descendants class-attribute export coupling)	A.22, A.4	Modificabilidade, Estabilidade
50	NIC(número de classes independentes)	A.4, A.6	Analisabilidade
51	CAM (Coupling among method)	A.11, A.36	Modificabilidade, Estabilidade
52	MFA (Measure of funcional abstraction)	A.36, A.6	Modificabilidade
53	NOP (Number of Parent)	A.16, A.4	Estabilidade, Testabilidade
54	CLD (Class to Leaf Depth)	A.16, A.4	Estabilidade, Testabilidade
55	TBI (Total base interfaces of system)	A.23, A.4	Analisabilidade
56	program volume (v)	A.28, A.35	Analisabilidade, Testabilidade
57	CndNbr - Number of Decisions	A.21, A.29	Estabilidade, Testabilidade
58	StmDecNbr - Number of Declaration statements	A.21, A.29	Analisabilidade
59	StrBrcNbr – Number of breaches of structure	A.21	Estabilidade, Testabilidade
60	Number of nested levels	A.29	Modificabilidade, Testabilidade
61	Number of unconditional jumps	A.29	Modificabilidade, Testabilidade
62	Vocabulary frequency	A.29	Modificabilidade
63	NMN (Number of Methods New)	A.4	Modificabilidade, Estabilidade
64	Class comments frequency	A.29	Analisabilidade
65	Average Size of Statements	A.29	Analisabilidade,Modificabilidade
66	ATFD	A.24	Modificabilidade, Estabilidade
67	INAG	A.8	Modificabilidade, Estabilidade
68	NEM (Number of external called method)	A.4	Modificabilidade, Estabilidade
69	NEA (Number of external used Attribute)	A.4	Modificabilidade,Estabilidade
70	PIMAS	A.8	Modificabilidade,Estabilidade
71	aveVG	A.35	Analisabilidade, Testabilidade
72	INTERNAL REUSE	A.5	Modificabilidade
73	aveV(media de volume por módulo)	A.35	Analisabilidade, Testabilidade
74	aveLOC (média de linhas de código por módulo)	A.35	Analisabilidade, Testabilidade
75	Subroutine averages	A.35	
76	extended cyclomatic complexity	A.35	Analisabilidade, Testabilidade
77	NstLvlAvg	A.21	Modificabilidade, Estabilidade
78	PIM	A.8	Modificabilidade, Estabilidade
79	REUSE VALUE	A.5	Modificabilidade
80	SIMAS -	A.8	Modificabilidade, Estabilidade
81	EXTERNAL REUSE	A.5	Modificabilidade
82	REUSE LEVERAGE	A.5	Modificabilidade
83	NEW OBJECT PERCENT	A.5	Modificabilidade
84	Changing Method (CM)	A.20	Modificabilidade
85	StmCtlNbr (Number of Control Statements)	A.21	Estabilidade, Testabilidade
86	Directly called components	A.29	Estabilidade
87	PthIndNbr (Number of Independents Paths)	A.21	Estabilidade, Testabilidade
88	C3 (Conceptual Cohesion of Classes)	A.7	Modificabilidade
89	v22	A.1	Analisabilidade
90	v11	A.1	Analisabilidade
91	v10	A.1	Analisabilidade
92	v6	A.1	Analisabilidade
93	PCCC (Path Connectivity Class Cohesion)	A.3	Modificabilidade
94	Oln	A.3	Modificabilidade
95	ICBMC (Improved Cohesion Based on Member Connectivity)	A.3	Modificabilidade
96	CBMC (Cohesion Based on Member Connectivity)	A.3	Modificabilidade
97	DC1	A.3	Modificabilidade
98	DCd	A.3	Modificabilidade
99	Coh	A.3	Modificabilidade
100	SCOM (Class Cohesion Metric)	A.3	Modificabilidade
101	Number of entry nodes	A.29	Estabilidade
102	LSCC (Low-level design Similarity-based Class Cohesion)	A.3	Modificabilidade
103	Number of exit nodes	A.29	Estabilidade
104	MOA	A.36	Modificabilidade
105	Type Rank	A.25	
106	method total reuse	A.33	Modificabilidade
107	method same subclass reuse	A.33	Modificabilidade
108	method same class reuse	A.33	Modificabilidade
109	Count of Blank lines	A.2	
110	(ABC) Association Between Class	A.25	Analisabilidade,Modificabilidade Estabilidade,Testabilidade

111	PercentageCoverage	A.25	Testabilidade
112	Count of Keywords	A.2	
113	Obsolete language constructs.	A.18	
114	Average cyclomatic complexity per method	A.29	Analisabilidade, Testabilidade
115	NIS (Number of interface)	A.4	Analisabilidade, Testabilidade
116	NdsNbr (Number of Nodes)	A.21	Estabilidade
117	CC (Class Cohesion)	A.3	Modificabilidade
118	ANU (Average Non Uniformity)	A.23	Modificabilidade, Estabilidade
119	perCM	A.35	Analisabilidade
120	OAM (Operation Acess Metric)	A.6	Modificabilidade, Estabilidade
121	CndCplAvg - Average Complexity of Decisions	A.21	Analisabilidade, Testabilidade
122	NOI (Number of Inline (Trivial) Methods)	A.6	Modificabilidade, Estabilidade
123	CalUnq - Unique Calls to Other Functions	A.21	Modificabilidade, Estabilidade
124	NAD (Number of Abstract Data Types)	A.6	Modificabilidade
125	NRA (Number of Reference Attributes)	A.6	Analisabilidade, Testabilidade
126	CalNbr - Total Calls to other functions	A.21	Modificabilidade, Estabilidade
127	CSB (Class Size in Bytes)	A.6	Analisabilidade, Testabilidade
128	CEC (Class Entropy Complexity)	A.6	Analisabilidade, Testabilidade
129	Average of- Weak Uniformity (AWU)	A.23	Modificabilidade, Estabilidade
130	Average of -Strong Uniformity (ASU)	A.23	Modificabilidade, Estabilidade
131	MAA (Measure of Attribute Abstraction)	A.6	Modificabilidade
132	VarLenAvg - Average variable name length	A.21	Analisabilidade
133	v1	A.1	Analisabilidade, Testabilidade
134	ComLogNbr - Number of non-blank lines	A.21	Analisabilidade, Testabilidade
135	ComStrVol - Volume of Control Comments	A.21	Analisabilidade
136	In-Degree (Number of Incoming Edges)	A.11	Modificabilidade, Estabilidade
137	NOSF (Number of Static Fields)	A.11	Analisabilidade, Testabilidade
138	NOSM (Number of Static Methods)	A.33	Analisabilidade, Testabilidade
139	Out-Degree (Number of outgoing edges)	A.11	Modificabilidade, Estabilidade
140	Specialization Index	A.11	
141	ntrn	A.11	
142	NHD	A.15	Modificabilidade
143	SNHD	A.15	Modificabilidade
144	NHDM	A.15	Modificabilidade
145	TODO Count	A.17	Analisabilidade
146	WMC (Average Weighted Methods Per Class)	A.17	Analisabilidade
147	TA (Type Affinity)	A.23	Modificabilidade, Estabilidade
148	FMMEC	A.22	Modificabilidade, Estabilidade
149	Essential complexity	A.28	Analisabilidade, Testabilidade
150	ComDecVol (Volume of declarations comments)	A.21	Analisabilidade
151	Program Length	A.28	Analisabilidade, Testabilidade
152	Program Vocabulary	A.28	Analisabilidade, Testabilidade
153	Difficulty	A.28	Analisabilidade, Testabilidade
154	IMNU (ImportNotUsed)	A.26	
155	ICH (Information-flow based cohesion)	A.22	Modificabilidade
156	COF (Coupling factor)	A.22	Modificabilidade, Estabilidade
157	IFCAIC	A.22	Modificabilidade, Estabilidade
158	NdsExtNbr (Number of Exits in a function)	A.21	Estabilidade, Testabilidade
159	IFCMIC	A.22	Modificabilidade, Estabilidade
160	LopNbr (Number of Loops)	A.21	Analisabilidade, Testabilidade
161	StmExeNbr - Number of executable statements	A.21	Analisabilidade
162	IFMMIC	A.22	Modificabilidade, Estabilidade
163	NLC	A.6	Estabilidade, Testabilidade
164	DMMEC	A.22	Modificabilidade, Estabilidade
165	OMMEC	A.22	Modificabilidade, Estabilidade
166	FCMEC	A.22	Modificabilidade, Estabilidade
167	CndSpnAvg (Average Decision Span)	A.21	Estabilidade, Testabilidade
168	OCAEC	A.22	Modificabilidade, Estabilidade
169	FCAEC	A.22	Modificabilidade, Estabilidade
170	Co		Modificabilidade
171	NewCo	A.22	Modificabilidade
172	NSI (Numbero of Single Inheterance)	A.6	Modificabilidade, Estabilidade
173	NMI (Number of Multiple Inherited)	A.6	Modificabilidade, Estabilidade
174	NNC	A.6	Modificabilidade, Estabilidade
175	NAC	A.6	Modificabilidade
176	ArcNbr (Number of Arcs)	A.21	Testabilidade
177	KntNbr (Number of Knots)	A.21	Testabilidade