UNIVERSIDADE FEDERAL DO PARÁ

INSTITUTO DE CIÊNCIAS EXATAS E NATURAIS

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Moshe Dayan Sousa Ribeiro

COMO COLETAR AUTOMATICAMENTE MÉTRICAS ORIENTADAS A OBJETO: UM ESTUDO BASEADO EM REVISÃO SISTEMÁTICA

Belém
2015

Moshe Dayan Sousa Ribeiro

COMO COLETAR AUTOMATICAMENTE MÉTRICAS ORIENTADAS A OBJETO: UM ESTUDO BASEADO EM REVISÃO SISTEMÁTICA

Dissertação de Mestrado apresentada para obtenção do grau de Mestre em Ciência da Computação. Programa de Pós-Graduação em Ciência da Computação. Universidade Federal do Pará. Área de Concentração Engenharia de Software.
Orientador: Antônio Jorge Gomes Abelém

Belém
2015

Moshe Dayan Sousa Ribeiro

Como Coletar Automaticamente Métricas Orientadas a Objeto: Um Estudo Baseado em Revisão Sistemática

Dissertação de Mestrado apresentada para obtenção do grau de Mestre em Ciência da Computação. Programa de Pós-Graduação em Ciência da Computação. Universidade Federal do Pará.

Data da aprovação: __ de __ de 2015

Banca Examinadora

Programa de Pós-Graduação em Ciência da Computação - UFPA - Orientador

Prof. Dr. Antônio Jorge Gomes Abelém

Programa de Pós-Graduação em Ciência da Computação – UFPA – Membro

Prof. Dr. Rodrigo Quites Reis

Programa de Pós-Graduação em Ciência da Computação – UFPA – Membro

Profª. Drª Carla Alessandra Lima Reis

Dedico este trabalho a minha família,
que é a base de tudo e em especial a minha
amada “vó”.

AGRADECIMENTOS

A Deus por sempre me acompanhar e me guiar em todos os momentos de minha vida.

A minha família pelo simples fato de ser o motivo de tudo que faço.

Ao professor Dr. Rodrigo Quites Reis que foi meu orientador desde a graduação e me acompanhou e direcionou por toda a minha vida acadêmica, inclusive mestrado, contribuindo não somente para o meu crescimento intelectual, mas também como pessoa. Sem dúvida, mais que um professor foi e é uma referência para mim.

Ao professor Dr. Antônio Jorge Gomes Abelém que gentilmente se tornou meu orientador para que eu pudesse concluir este trabalho.

A professora Dr. Carla Alessandra Lima Reis por ter contribuído com minha formação desde a graduação e por participar da minha banca de defesa de mestrado.

A todos que contribuíram de alguma forma para que esse sonho se realizasse, eu deixo registrada a minha mais sincera gratidão.

RESUMO

Objetivo: Obter informações sobre como coletar automaticamente métricas orientadas a objeto (métricas OO) com intuito de apoiar, nessa tarefa, quem deseja compreender/avaliar o produto de software através dessas métricas. Método: um estudo baseado em revisão sistemática foi desenvolvido. 37 estudos primários foram selecionados dos 577 trabalhos recuperados em 3 bases de dados. Resultado: 177 métricas que podem ser coletadas automaticamente foram catalogadas e desse total, 28 foram as mais referenciadas; as métricas catalogadas foram classificadas conforme as características de qualidade as quais estavam relacionadas; 18 ferramentas de coleta foram identificadas; concluiu-se que existe um conjunto de procedimentos comuns para se realizar a coleta de métricas OO e que Java e C++ são as linguagens com maior número de ferramentas para as quais é possível extrair métricas.

Palavras-chave: Qualidade de Software; Produto de Software; Métricas Orientadas a Objeto; Ferramenta; Coleta de Métricas; Revisão Sistemática.

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, 28 from such total were the most referenced. The selected 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; Software Product; Object Oriented Metrics; Tool; Metrics Collection; Systematic Review.

LISTA DE FIGURAS

FIGURA 1. ABORDAGENS DE QUALIDADE SEGUNDO GARVIN 1992 18
FIGURA 2. CARACTERÍSTICAS E SUBCARACTERÍSTICAS DE QUALIDADE. ADAPTADO DE SQUARE (2011) 20
FIGURA 3. CICLO DA PESQUISA CONTENDO AS ETAPAS ANTERIORES A RLS (A) E O PROCESSO PARA REALIZAÇÃO DA MESMA DIVIDIDO EM QUATRO FASES, ADAPTADO DE BIOLCHINI ET. AL 2007 (B) 30
FIGURA 4. RESUMO DAS TRÊS FASES DEFINIDAS NO PROTOCOLO DESTA REVISÃO. 38
FIGURA 5. DISTRIBUIÇÃO DOS ESTUDOS SELECIONADOS POR BASE 40
FIGURA 6. DISTRIBUIÇÃO DOS ESTUDOS PELOS PAÍSES DAS INSTITUIÇÕES DOS PESQUISADORES 41
FIGURA 7. DISTRIBUIÇÃO DOS ESTUDOS POR ANO 42
FIGURA 8. DISTRIBUIÇÃO DAS MÉTRICAS POR PROPRIEDADE RELACIONADA 43
FIGURA 9. QUANTIDADE DE MÉTRICAS POR SUBCARACTERÍSTICA DE QUALIDADE 45
FIGURA 10. MÉTRICAS MAIS REFERENCIADAS 46
FIGURA 11. COMPARAÇÃO DAS 177 MÉTRICAS CATALOGADAS E AS 28 MAIS REFERENCIADAS COM RELAÇÃO AS PROPRIEDADES QUE ELAS MENSURAM 47
FIGURA 12. COMPARAÇÃO DAS 177 MÉTRICAS CATALOGADAS E AS 28 MAIS REFERENCIADAS COM RELAÇÃO AS SUBCARACTERÍSTICAS QUE ELAS MENSURAM 47
FIGURA 13. DISTRIBUIÇÃO DOS ESTUDOS COM BASE NAS INFORMAÇÕES SOBRE A FERRAMENTA DE COLETA 50
FIGURA 14. COMPONENTES DAS FERRAMENTAS DE COLETA DE MÉTRICAS 51
FIGURA 15. QUANTIDADE DE FERRAMENTAS QUE COLETAM MÉTRICAS POR LINGUAGEM DE PROGRAMAÇÃO 54

LISTA DE TABELAS

TABELA 1. NÍVEIS DE ABRANGÊNCIA DAS MÉTRICAS 22
TABELA 2. ORGANIZAÇÃO DAS QUESTÕES 32
TABELA 3. DIMENSÕES DA REVISÃO 33
TABELA 4. EXPRESSÃO DE BUSCA DEFINIDA 34
TABELA 5. INFORMAÇÕES EXTRAÍDAS 36
TABELA 6. CRITÉRIOS DE INCLUSÃO E EXCLUSÃO DE ESTUDOS 36
TABELA 7. RESULTADOS DA FASE 1 37
TABELA 8. RESULTADOS DA FASE 2 37
TABELA 9. FERRAMENTAS IDENTIFICADAS 49

LISTA DE ABREVIATURAS E SIGLAS

ADS	Ambiente de Desenvolvimento de Software
CMMi	*Capability Maturity Model Integrated*
ISO	*International Organization for Standardization*
LABES	Laboratório de Engenharia de Software da Universidade Federal do Pará
MPS.BR	Programa para a Melhoria do Processo de Software Brasileiro
MR-MPS	Modelo de Referência de Melhoria do Processo de Software
PSEE	*Process-Centered Software Engineering Environment*
SOFTEX	Associação para Promoção da Excelência do Software Brasileiro
SQuaRE	*Software product Quality Requirements and Evaluation*

SUMÁRIO

RESUMO VI

ABSTRACT VII

LISTA DE FIGURAS VIII

LISTA DE TABELAS IX

LISTA DE ABREVIATURAS E SIGLAS X

1. INTRODUÇÃO 13

1.1 MOTIVAÇÃO E CONTEXTO 13

1.2 OBJETIVOS DA PESQUISA 15

1.3 ESTRUTURA DO TRABALHO 16

2. FUNDAMENTAÇÃO TEÓRICA E TRABALHOS RELACIONADOS 17

2.1 QUALIDADE DE SOFTWARE 17

2.2 MÉTRICAS ORIENTADAS A OBJETO 21

2.2.1 Classificação das Métricas Orientadas a Objeto 21

2.2.2 Polimorfismo 22

2.2.3 Coesão 23

2.2.4 Acoplamento 24

2.2.5 Encapsulamento 25

2.2.6 Tamanho e Complexidade 25

2.3 TRABALHOS RELACIONADOS 26

2.4 CONSIDERAÇÕES SOBRE O CAPÍTULO 27

3. METODOLOGIA 29

3.1 INTRODUÇÃO 29

3.2 QUESTÃO DE PESQUISA E DIMENSÃO 31

3.3 ESTRATÉGIA DE BUSCA E FONTE DE DADOS 33

3.4 PROCEDIMENTOS E CRITÉRIOS PARA SELEÇÃO DE ESTUDOS 35

3.5 SÍNTESE E APRESENTAÇÃO DOS RESULTADOS 37

3.6 CONSIDERAÇÕES SOBRE O CAPÍTULO 38

4. ANÁLISE DOS RESULTADOS 39

4.1 VISÃO GERAL 39

4.2 QUESTÃO 01: QUAIS SÃO AS MÉTRICAS OO COLETADAS AUTOMATICAMENTE? 42

4.3 QUESTÃO 01.1: QUAIS CARACTERÍSTICAS DE QUALIDADE PODEM SER AVALIADAS PELAS MÉTRICAS OO IDENTIFICADAS? 44

4.4 QUESTÃO 01.2: QUAIS SÃO AS MÉTRICAS MAIS FREQUENTES? 45

4.5 QUESTÃO 02: QUAIS SÃO AS FERRAMENTAS PARA COLETA AUTOMÁTICA DE MÉTRICAS DE CÓDIGO FONTE E COMO REALIZAM ESSA TAREFA? 48

4.6 QUESTÃO 02.1: AS FERRAMENTAS POSSUEM INTEGRAÇÃO COM OUTRAS DO PROCESSO DE SOFTWARE? 52

4.7 QUESTÃO 02.2: COMO AS FERRAMENTAS APRESENTAM AS INFORMAÇÕES DAS MÉTRICAS COLETADAS? 52

4.8 QUESTÃO 02.3: PARA QUAIS LINGUAGENS DE PROGRAMAÇÃO AS FERRAMENTAS PERMITEM A EXTRAÇÃO DE MÉTRICAS? 53

4.9 CONSIDERAÇÕES SOBRE O CAPÍTULO 54

5. CONCLUSÃO 56

5.1 LIMITAÇÕES DO ESTUDO E AMEAÇAS À VALIDADE 56

5.2 CONSIDERAÇÕES FINAIS E TRABALHOS FUTUROS 57

REFERÊNCIAS BIBLIOGRÁFICAS 58

REFERÊNCIAS BIBLIOGRÁFICAS REVISÃO SISTEMÁTICA DA LITERATURA 61

APÊNDICE A – LISTA DE MÉTRICAS IDENTIFICADAS 64

1. INTRODUÇÃO

Este capítulo apresenta o contexto, motivação e objetivo da pesquisa. Por fim, a estrutura deste trabalho é mostrada ao final deste capítulo.

1.1 Motivação e Contexto

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, fornecendo apoio à tomada de decisão em projetos de software (Pressman, 2011). Para acompanhar a avaliação da qualidade do software, os engenheiros de software devem utilizar técnicas de medição objetiva (Pressman, 2011). Assim, normas como ISO/IEC 9126 (2001) - primeira geração de norma de qualidade da engenharia de software para produto de software - e ISO/IEC 25000 (2005) - SQuaRE (segunda geração de normas de qualidade de software) tornam-se importantes instrumentos para realizar tal atividade.

Segundo a ISO/IEC 25020 (2007), a qualidade interna do software é o conjunto de capacidade de atributos estáticos do produto de software que atendem as necessidades explícitas e implícitas quando este produto é utlizado em condições específicas. Atributos estáticos incluem aqueles que se relacionam com a estrutura do software, arquitetura e seus componentes. Exemplos desses atributos são as métricas de código fonte, tais como o número de linhas de código e as medidas de complexidade do software.

Os requisitos de qualidade interna do software são usados para especificar as características dos produtos de software intermediários, tais como: código fonte, documentos e manuais. Além disso, eles podem ser utilizados para definir a estratégia de avaliação e critérios de verificação durante o desenvolvimento do software (ISO/IEC 25000, 2005).

Métricas internas, também, podem ser utilizadas para gerar indicadores que permitam prever o comportamento do sistema durante a fase de testes funcionais. Consequentemente, as medidas internas são mais importantes para os gerentes ou líderes de desenvolvimento, porque são ferramentas valiosas para evitar problemas durante o processo de desenvolvimento do software (ISO/IEC 9126-3, 2001).

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 (Basili *et al.*, 1996) (Chidamber e Kemerer, 1994). 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 (Isong *et al.*, 2013) (Jabangwe *et al.*, 2014). 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 (Isong *et al.*, 2013) (Jabangwe *et al.*, 2014).

Diante disso, o grupo de pesquisa LABES-UFPA, com objetivo de aumentar o apoio as atividades de medição e qualidade no ADS Centrado em Processo WebAPSEE [descrito em Lima e Reis (2007)], resolveu estender as funcionalidades de medição desenvolvidas por Nascimento (2007) e evoluídas em Ribeiro (2011).

Para isso, primeiramente realizou diversas discussões no sentido de identificar que atividade do processo de medição poderia ter um apoio ferramental mais efetivo. Ao final dessas discussões, verificou-se que em algumas situações a coleta de métricas se tornava bastante custosa, como por exemplo, quando a métrica exigia um esforço muito grande calculá-la ou quando a quantidade de métricas a ser coletada era elevada. Por isso chegou-se

ao consenso que o apoio seria no sentido de oferecer recursos para atender a uma recomendação[1] do Modelo de Referência MPS para Software MR-MPS-SW (SOFTEX, 2013) que segue a linha de tornar essa tarefa mais automatizada possível e integrada aos outros processos inerentes ao desenvolvimento de software.

A partir da oportunidade identificada de apoiar a atividade de coleta de métricas OO, surgiu este trabalho com intuito de buscar elementos necessários para automatizar a coleta de métricas OO de forma a apoiar o processo de medição e tomada de decisões gerenciais.

1.2 Objetivos da Pesquisa

O objetivo principal deste trabalho é analisar os estudos relacionados com coleta automática de métricas OO, a partir de uma Revisão Sistemática da Literatura (RSL) para responder duas questões centrais:

- **Quantas e quais são as métricas OO de código fonte coletadas automaticamente?**
- **Quais são as ferramentas para coleta automática de métricas de código fonte e como realizam essa tarefa?**

Como complemento das questões principais, foram formuladas as seguintes questões secundárias:

- Quais características de qualidade podem ser avaliadas pelas métricas OO identificadas?
- Quais são as métricas mais frequentes?
- As ferramentas possuem integração com outras do processo de software?
- Como as ferramentas apresentam as informações das métricas coletadas?
- Para quais linguagens de programação as ferramentas permitem a extração de métricas?

Como objetivos específicos:

[1] Recomendação contida no resultado esperado MED3 do processo de Medição do Guia de Implementação nível F do MR-MPS-SW.

- Contribuir com informações válidas e úteis, para a comunidade acadêmica e pesquisadores em geral, sobre como coletar métricas OO automaticamente;
- Estimular o desenvolvimento de novos estudos neste tema, por conta dos *gaps* identificados e oportunidades de pesquisas vislumbradas após a análise dos resultados obtidos neste trabalho.

1.3 Estrutura do trabalho

Além deste capítulo introdutório, este texto está organizado da seguinte forma: o segundo capítulo apresenta a fundamentação teórica deste trabalho por meio da exposição dos principais conceitos utilizados ao longo da pesquisa, que estão relacionados com Qualidade e Métricas de Software.

O terceiro capítulo se concentra em mostrar a metodologia definida e aplicada para o desenvolvimento deste trabalho que, de forma resumida, mostra a identificação da necessidade de realizar uma RSL e a apresentação de um protocolo elaborado para realização da mesma.

No quarto capítulo os resultados obtidos na RSL são apresentados como respostas as questões de pesquisa definidas no protocolo da revisão (capítulo 3). E uma análise desses resultados é realizada.

Finalmente, o quinto e último capítulo conclui este trabalho; expõe suas limitações; mostra as suas principais contribuições e apresenta os trabalhos futuros.

2. FUNDAMENTAÇÃO TEÓRICA E TRABALHOS RELACIONADOS

Nesta seção é apresentada a fundamentação teórica que mostra as definições da literatura para qualidade de software e os principais conceitos de medição de software obtidos por meio da realização de uma revisão informal da literatura. Dentre esses conceitos destaca-se o de métrica, por ser um elemento fundamental no processo de medição e na realização deste trabalho. As normas ISO/IEC 9126 e SQuaRE são descritas, bem como a classificação das métricas adotadas, em seguida mostra-se as métricas OO e suas classificações encontradas na literatura. Por fim, os trabalhos relacionados são apresentados e algumas considerações são feitas nas quais se destaca a diferença entre este trabalho e os demais relacionados.

2.1 Qualidade de Software

O conceito de qualidade é complexo e por muitas vezes tratado de forma subjetiva porque seu significado depende tanto da expectativa de diferentes pessoas como do contexto em que está sendo requerida (Kitchenham e Pfleeger, 1996). Em função da sua organização e abrangência Garvin (1992) sistematizou os conceitos de qualidade em cinco abordagens diferentes conforme figura 1.

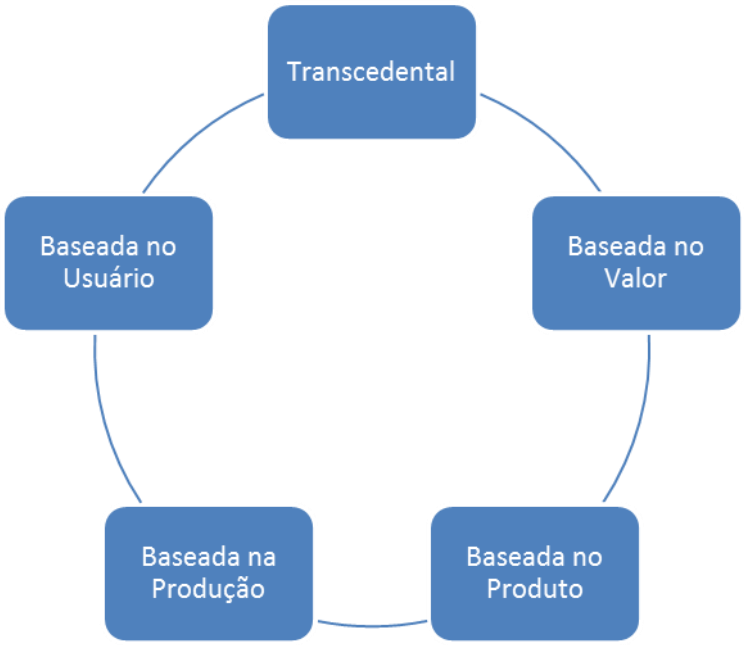


Figura 1. Abordagens de Qualidade segundo Garvin 1992

A **abordagem transcendental** considera a qualidade como sinônimo de perfeição sobre aquilo que se deseja em relação ao software. Uma visão idealizada e subjetiva, não sendo possível descrevê-la e nem criticá-la, pois os elementos capazes de qualificar alguém para tal avaliação seria a experiência e observação. Em outras palavras, nessa abordagem a qualidade seria o mesmo que beleza, atratividade e excelência, situação comumente encontrada em produtos de marcas que são reconhecidas pela qualidade em tudo que fazem.

A abordagem baseada na produção relaciona qualidade ao processo de desenvolvimento e todas as suas diferentes estratégias para controlar custos, prazos e defeitos. Nessa abordagem, um produto de qualidade é aquele que foi desenvolvido seguindo as regras, condições e procedimentos definidos no processo de desenvolvimento do software. O enfoque dessa abordagem pressupõe que um software produzido sem ter seguido um processo de desenvolvimento previamente definido será mal desenvolvido e não confiável, diminuindo a satisfação do usuário. Nesse contexto, modelos de capacidade e maturidade para software, tais como o *Capability Maturity Model for Development* CMMI-Dev (SEI, 2010) e o MR-MPS-SW (SOFTEX, 2013) definem uma série de elementos a serem alcançados durante todo o desenvolvimento do software com objetivo de garantir qualidade ao processo já que existe evidência da relação entre a qualidade do processo e do produto final (Travassos e Kalinowski, 2014).

A **abordagem baseada no produto** define a qualidade como uma variável mensurável e precisa pois está relacionada às características próprias do produto. Se o produto realiza aquilo que se espera, ele tem qualidade. Ou seja, qualidade é a adequação ao uso (Juran, 1974). No contexto de software, a visão de produto está ligada a aspectos internos, como o cumprimento da arquitetura adotada, organização do código e utilização dos padrões de codificação estabelecidos.

A **abordagem baseada no usuário** reflete as expectativas do usuário com relação ao que foi solicitado e ao efetivamente entregue. Nessa abordagem um software de qualidade seria aquele que melhor atendesse às necessidades e expectativas dos usuários no momento que eles desejam. Nesse sentido, essa a abordagem se assemelha à transcendental, porque ambas refletem uma visão subjetiva da qualidade.

A **abordagem baseada no valor** estabelece um custo/preço para o software. Ela depende do nível de qualidade auferido nas abordagens de usuário, produção e produto. Essa dependência se dá de forma a estabelecer uma relação equilibrada entre custo e qualidade oferecida pelo software. Por depender de abordagens que possuem critérios subjetivos de avaliação, essa visão é de difícil aplicação, pois seus limites não são bem definidos já que pessoas diferentes valorizam características diferentes.

Como este trabalho se concentra na **abordagem baseada no produto**, vale ressaltar que diversas instituições, como a ISO, criaram normas para permitir uma avaliação mais eficiente da qualidade do produto de software. Para isso a ISO, inicialmente, definiu a norma ISO/IEC 9126 (2001) que trata de apresentar uma estrutura para auxiliar na avaliação da qualidade de produtos de software, que consiste em um conjunto de características que devem ser verificadas em um software para que ele seja considerado um "software de qualidade”. Também elaborou a norma ISO/IEC 14598 (2001) responsável por apoiar o planejamento e a execução do processo de avaliação da qualidade do produto de software atuando como um complemento a ISO/IEC 9126, pois inclui modelos para relatórios de avaliação, técnicas para medição das características de qualidade, documentos necessários para avaliação e fases da avaliação.

Em 2009 a ISO reformulou e evoluiu as normas ISO/IEC 9126 e ISO/IEC 14598 dando origem a uma série de padrões denominada SQuaRE que estabelece um conjunto de

elementos do software mensuráveis que possibilitam quantificar a qualidade do produto de software em diferentes características e sub-características de qualidade. Dessa maneira, a qualidade do produto é obtida por meio da capacidade de atender as necessidades explícitas e implícitas relacionadas com essas características de qualidade (figura 2). Essa avaliação pode ser em relação aos elementos internos que estão relacionados com as propriedades estáticas do software e/ou externas quando esses elementos são propriedades inerentes ao comportamento do software.

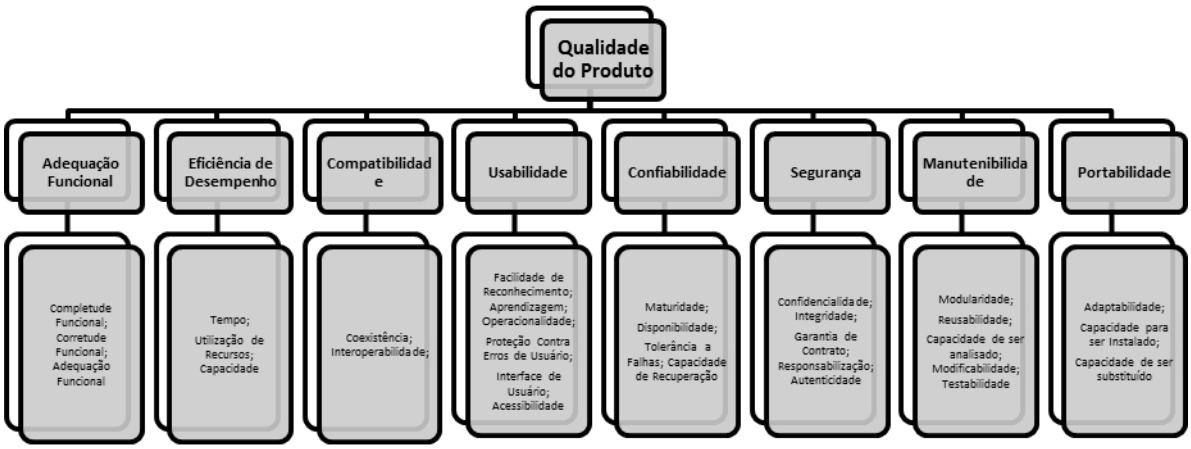


Figura 2. Características e Subcaracterísticas de Qualidade. Adaptado de SQuaRE (2011)

Nesse cenário, as características de qualidade (segundo nível da figura 2) não podem ser medidas diretamente. Para isso se faz necessário estabelecer métodos de medição (métricas) e aplicá-los para se atribuir valores a um elemento de qualidade (medição), seja ele interno ou externo, relacionando-o a uma característica de qualidade. A norma fornece definição e apresenta pelo menos uma medida para cada sub-característica (terceiro nível da figura 2) de qualidade, possível de ser obtida a partir de elementos mensuráveis de qualidade interna e/ou externa. Com base na definição apresentada, este trabalho se concentra no escopo da avaliação da qualidade de elementos de qualidade interna do software, ou seja, relacionado com as propriedades estáticas do software, mais precisamente com as propriedades estáticas do código fonte.

2.2 Métricas Orientadas a Objeto

Para garantir a qualidade do produto de software de maneira efetiva é necessário avaliar as características de qualidade do software com base em medições. Dessa forma, torna-se fundamental a utilização de métricas que possibilitem a mensuração quantitativa ou qualitativa de elementos de qualidade relacionados ao software. Além disso, métricas também são utilizadas para: prever cronograma de projetos; estimar custos; controlar produtividade entre outros. Em outras palavras, a Medição fornece subsídios fundamentais para tomada de decisão em projetos de software.

Segundo a ISO/IEC 9126 (2001), métrica pode ser conceituada como sendo um conjunto de procedimentos de medição e escalas de medidas. Nesse sentido, métricas de código-fonte são aquelas obtidas por meio da análise do código-fonte que de acordo com a chamada de trabalhos SCAM[2] é qualquer descrição completamente executável de um sistema de software.

Nesse contexto, a importância do código-fonte na qualidade de software é grande e tende a crescer (Harman, 2010). Por isso, diversas métricas de código fonte foram definidas e muitas focaram nas peculiaridades do paradigma da orientação objeto, dentre as quais se destaca o conjunto de seis métricas propostas por Chidamber e Kemerer (1994) conhecidas também como métricas CK. São métricas muito citadas na literatura (ver seção 4.4) e também um dos primeiros trabalhos a propor métricas específicas para o paradigma de Orientação a Objeto.

As diversas métricas OO encontradas na literatura são classificadas sobre diferentes aspectos e eles são discutidos com detalhes na seção seguinte (2.2.1).

2.2.1 Classificação das Métricas Orientadas a Objeto

Na literatura as métricas podem ser classificadas tanto em relação às propriedades as quais podem mensurar quanto ao seu escopo de abrangência. É importante chamar a atenção que algumas propriedades são típicas do paradigma da orientação a objeto (polimorfismo,

[2] IEEE International Working Conference on Source Code Analysis and Manipulation

acoplamento entre outras) e algumas são das linguagens em geral, como tamanho e complexidade, por exemplo.

Na tabela 1 são apresentados os três níveis de abrangência das métricas bem como a descrição de cada um deles, e uma métrica de exemplo para complementar o entendimento.

Tabela 1. Níveis de Abrangência das Métricas

Abrangência	Descrição	Exemplo
Sistema	As métricas avaliam e fornecem informações do sistema como um todo.	Número de classes – NOC.
Classe	O escopo da métrica se restringe à classe.	Número de métodos.
Método	O cálculo da métrica ocorre em nível de método.	Número de parâmetros do método.

Existem trabalhos que utilizam métricas de um nível mais baixo de abrangência para avaliar o software em um nível mais alto. Para isso, são utilizados recursos estatísticos como médias, desvio entre outros. Um exemplo desse tipo de situação ocorre quando se calcula a média de atributos por classe. Nesse caso se obtém um valor de referência para o sistema como um todo em relação a quantidade de atributos que uma classe desse sistema possui. Isso pode ser feito para várias outras métricas como média do número de métodos, média do número de parâmetros por classe e assim por diante.

As seções a seguir detalham as propriedades utilizadas para classificar as métricas OO. Além disso mostram uma métrica de exemplo para cada uma delas. Vale ressaltar que no anexo II, ao final deste trabalho, estão todas as métricas identificadas bem como a classificação delas com relação às propriedades que elas mensuram.

2.2.2 Polimorfismo

O Polimorfismo pode ser implementado e entendido de diferentes maneiras. Uma delas é que o polimorfismo permite que a implementação de uma operação possa depender do objeto que a contém Aggarwal et al. (2006). Para os objetivos deste trabalho o polimorfismo

será apresentado segundo a classificação de Benlarbi et al., (1999) na qual divide o polimorfismo em três grupos distintos:

- Polimorfismo puro - consiste na definição de métodos com mesmo nome, porém com assinaturas diferentes dentro do escopo de uma mesma classe.

- Polimorfismo estático - ocorre quando métodos com mesmo nome, mas com assinaturas diferentes são implementados em classes distintas.

- Polimorfismo dinâmico - acontece quando uma classe sobrescreve[3] o método herdado da classe pai, para isso utiliza o mesmo nome e assinatura do método sobrescrito.

A implementação do polimorfismo é interessante porque permite a redução da complexidade de projeto, pois possibilita que mensagens com a mesma assinatura sejam tratadas por diferentes classes por meio da relação de herança, por exemplo.

Uma das métricas que mensuram o polimorfismo é *Number of Methods Overridden*[4] que calcula o número de métodos sobrescritos pelas subclasses.

2.2.3 Coesão

Pode ser definida como o grau em que um conjunto de propriedades de uma entidade é parte do problema ou domínio do projeto Whitmire *apud* (Pressman, 2011). De maneira mais específica, coesão é a medida do grau em que os elementos de uma classe (módulo, pacote e assim por diante) são funcionalmente relacionados. Uma classe fortemente coesa, por exemplo, implementa a funcionalidade que está relacionada com uma característica do software e requer pouca ou nenhuma interação com outras classes. Assim, é interessante que classes sejam altamente coesas, pois facilitaram o reuso e a manutenção já que concentram todos os recursos necessários para desempenhar uma determinada função e não dependem de outras para tal.

---

[3] Termo muito conhecido em inglês como *overriding*.

[4] Essa métrica está disponível no anexo II deste trabalho, identificada pelo número 18.

Uma métrica bastante encontrada na literatura que mensura essa propriedade é *a Lack of Cohesion in Methods* - LCOM (Chidamber e Kemerer, 1994). Ela mede a dissimilaridade dos métodos de uma classe, observando a utilização dos atributos da classe por esses métodos. Em geral, valores elevados de LCOM sugerem que a classe pode ser melhor projetada quebrando-a em duas ou mais classes diferentes, embora existam casos nos quais um alto valor de LCOM é justificável. No entanto, manter a coesão alta, ou seja, manter LCOM baixo é desejável (Pressman, 2011).

Falta de coesão sinaliza que classes podem ser divididas em duas ou mais classes. Um baixo grau de coesão aumenta a complexidade, o que pode elevar os erros ao longo do desenvolvimento do software (Chidamber e Kemerer, 1994).

2.2.4 Acoplamento

Essa propriedade pode ser entendida como o grau de dependência entre as classes e por isso tem uma ligação com o conceito de **herança** da orientação a objeto já que a relação de herança entre classes constitui um acoplamento entre elas.

Para melhorar a modularidade, encapsulamento e o reuso, por exemplo, o acoplamento entre as classes deve ser o menor possível já que um alto grau de acoplamento dificulta mudanças no software, pois a alteração de uma classe pode afetar várias outras elevando o custo de manutenção. Além disso, classes muito acopladas dificilmente são reutilizadas em outros contextos pelo fato de possuírem elementos muito específicos do ambiente em que foram criadas o que difere de classes desacopladas as quais possuem um baixo nível de dependência facilitando o seu reuso em outras aplicações.

Outro fato interessante é que o grau de acoplamento é utilizado para avaliar a complexidade do teste a ser realizado em partes do projeto, porque quanto maior o acoplamento mais complexo será o teste, pois também envolvera as outras partes interdependentes.

Por tudo isso essa propriedade deve ser controlada (Poels, 1998) (Chidamber e Kemerer, 1994) e é desejável que as classes tenham o mínimo de dependência quanto possível uma das outras (Abreu, 1995).

Chidamber e Kemerer (1994) propuseram a métrica *Coupling Between Object Classes* – CBO para mensurar o acoplamento entre classes e a definiram da seguinte forma: CBO de uma classe é o número total de outras classes com a qual ela é acoplada. Para o cálculo dessa métrica entende-se acoplamento conforme exemplo a seguir: a classe A está acoplada a classe B quando métodos declarados na classe A utilizam métodos ou atributos definidos pela classe B. Nesse caso, a classe A está acoplada a classe B e CBO da classe A é igual a 1.

2.2.5 Encapsulamento

O encapsulamento pode ser entendido como uma forma de fazer com que detalhes internos do funcionamento dos métodos, ou de acesso a atributos de uma classe, permaneçam ocultos para os objetos. Ou seja, é o resultado (ou ato) de ocultar do usuário os detalhes da implementação de um objeto. Por conta disso, o conhecimento a respeito da implementação interna da classe é desnecessário do ponto de vista do objeto (ou de quem vai utilizá-lo), uma vez que isso passa a ser responsabilidade dos métodos internos da classe.

Essa propriedade é importante porque separa a maneira como um objeto se comporta da maneira como ele é implementado. O que permite a proteção dos dados do objeto do uso arbitrário e não intencional. Além disso, reduz o esforço para entendimento e uso de classes já que o programador se preocupará como o que o objeto faz e não como ele faz.

Uma métrica bastante utilizada na literatura para mensurar essa propriedade é a *Method Hiding Factor* – MHF, proposta por Abreu *et al.* (1994). Essa métrica indica por meio de um índice, que varia de zero a um, o fator de métodos inacessíveis[5], levando em consideração o software como um todo. Essa métrica ajuda o desenvolvedor a identificar o grau de encapsulamento de métodos no projeto inteiro.

2.2.6 Tamanho e Complexidade

Classes e métodos longos são difíceis de serem lidos e entendidos, ocasionando o aumento do esforço para manutenções futuras e desenvolvimento de novas funcionalidades. Por outro lado, complexidade está ligada as interações entre diferentes elementos do software.

[5] Conhecidos também como métodos privados com acesso exclusivo da classe que o declarou.

Em outras palavras, quanto mais interações maior é a probabilidade de uma mudança influenciar em outras partes do software, ocasionando mais mudanças ou introduzindo falhas não previstas. Nesse contexto, (Lehman et al., 1997) afirma que a complexidade está fortemente relacionada a manutenibilidade que, por sua vez, está ligada ao esforço necessário para realizar uma mudança no software.

Dessa forma, pode-se notar que tamanho e complexidade são propriedades altamente relacionadas porque quanto maior o número de elementos que compõem o software, maior será o número possível de interações entre eles, que, por consequência, tornará os softwares grandes mais suscetíveis a ter um grau elevado de complexidade. Por isso é interessante utilizar métricas de complexidade e tamanho em conjunto.

Uma das métricas de tamanho mais referenciadas pela literatura é a *Line of Code* – LOC que pode ser utilizada em nível de método, classe ou em relação ao software como um todo. Também é utilizada para medir produtividade.

Outra métrica bastante utilizada, mas que está relacionada com complexidade, é *Average Parameter per Method* que fornece o número médio de parâmetros dos métodos de um software. Essa métrica é interessante porque métodos com um número grande de argumentos são difíceis de serem lidos, entendidos e a comunicação entre objetos se torna mais complexa por conta disso.

2.3 Trabalhos Relacionados

O trabalho de Abilio *et al.* (2012) 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.* (2012) 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 Saraiva *et al.*(2012) a inferir que poucos estudos utilizam métricas anteriormente propostas.

O trabalho de Filó (2014) propôs um catálogo com valores referência para 18 métricas orientadas a objetos, que avaliam o software em nível de métodos, classes e pacotes. Para isso, utilizou o método empírico proposto por Ferreira et al. (2012) com algumas adaptações/melhorias. Além disso, estabeleceu três faixas de valores para identificação dos valores referência propostos: Bom/Frequente, Regular/Ocasional e Ruim/Raro. Embora esses valores não expressem necessariamente as melhores práticas da Engenharia de Software, eles refletem o padrão de qualidade seguido por muitos software. Três estudos de caso foram conduzidos para avaliar esses valores de referência e os resultados indicaram que os valores referência definidos tem a capacidade de fornecer uma visão geral da qualidade de métodos, classes e pacotes por meios quantitativos.

Uma das principais diferenças entre o trabalho aqui apresentado e os estudos de Abilio *et al.* (2012) e Saraiva et al. (2012), 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? A resposta para essas questões pode contribuir para utilização prática de métricas OO. Em relação ao estudo de Filó (2014) este trabalho pode ser utilizado como instrumento de apoio na atividade coleta métricas OO para uma possível comparação com os valores de referência definidos por ele.

2.4 Considerações sobre o Capítulo

Ao longo deste capítulo mostrou-se os principais conceitos relacionados com qualidade, focando na qualidade de software até chegar na qualidade do produto de software. Nesse

contexto normas que tratam de qualidade de software foram apresentadas como a ISO/9126 e a série SQuaRE.

Uma visão mais detalhada sobre métricas orientadas a objeto foi apresentada bem como a sua classificação inerente a propriedade que ela está relacionada: polimorfismo, coesão, acoplamento, encapsulamento, tamanho e complexidade.

Por fim, os trabalhos relacionados foram expostos e uma discussão sobre eles foi realizada no sentido de deixar clara a diferença entre este trabalho e os demais.

3. METODOLOGIA

Neste capítulo é descrita a abordagem metodológica deste trabalho com objetivo de tornar sua replicação e auditoria possível por outros pesquisadores.

3.1 Introdução

A metodologia adotada neste trabalho é fortemente baseada no processo de realização de uma revisão sistemática e por esse mesmo motivo este trabalho é classificado como um estudo secundário já que segue as orientações de Kitchenham *et. al* (2007) no que se refere a elaboração do protocolo de planejamento e execução da pesquisa.

A metodologia aplicada é retratada na figura 3 a qual apresenta as etapas de pesquisa incluindo o processo de elaboração e execução desta revisão. Nota-se que após a **pesquisa informal da literatura (a)**, sobre ferramentas para coleta automática de métricas OO, percebeu-se a carência de estudos sistemáticos que abordassem de maneira mais especifica este assunto - **identificar necessidades de RLS (b)**. Por esse motivo, decidiu-se realizar uma revisão sistemática da literatura para investigar como ocorre a coleta automática de métricas OO: quais ferramentas são adotadas, que abordagem utilizam, métricas, linguagens de programação e quais características de qualidade são avaliadas.

Uma Revisão Sistemática consiste em um estudo voltado para reunião de evidências sobre um determinado tema ou assunto de maneira sistemática e formal com o objetivo de responder a questões de pesquisa. Para isso contém elementos essências como as questões de pesquisa que o estudo se propõe a responder; fontes de dados; estratégia de busca; critérios de exclusão e inclusão de estudos; procedimentos para extração e análise dos resultados. A

documentação de todo esse processo possibilita a reaplicação da busca para auditoria ou mesmo para continuação da pesquisa para outros períodos não considerados neste trabalho.

Segundo Biolchini *et.al* 2007 o processo para realização de uma revisão sistemática da literatura pode ser dividido em quatro fases, são elas: **planejamento**, **execução**, **análise dos resultados** e **empacotamento,** conforme figura 3 (b).

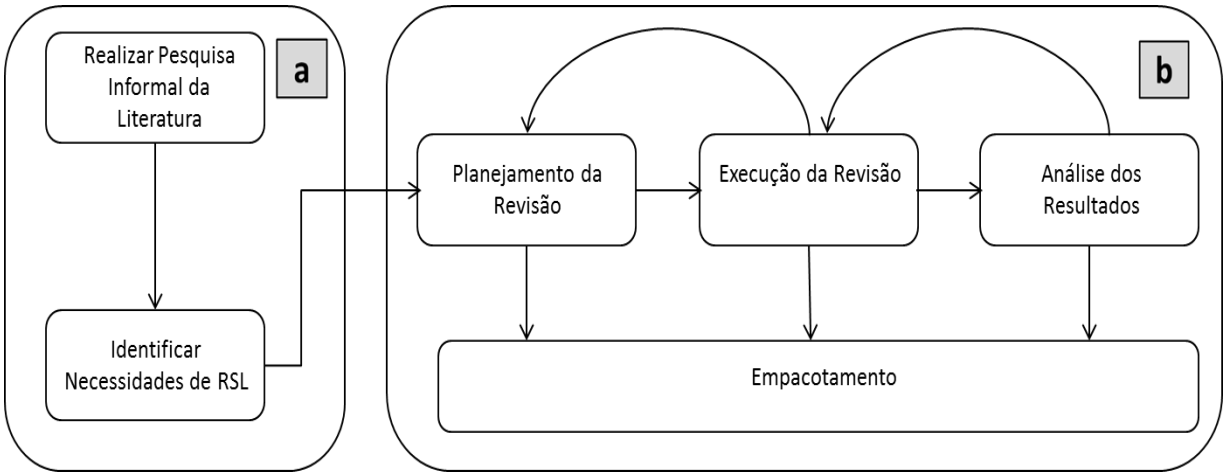


Figura 3. Ciclo da Pesquisa contendo as etapas anteriores a RLS (a) e o processo para realização da mesma dividido em quatro fases, adaptado de Biolchini et. al 2007 (b)

Na fase de **planejamento** é definido o protocolo da revisão o qual especifica os métodos que serão utilizados para realizar a revisão sistemática, guiando todas as demais atividades ao longo do processo (Mafra e Travassos 2006). Nessa fase é definida a *string* de busca com base na abordagem de PICO - *Population of interest, evaluated Intervention, Intervention Comparison e expected Outcome* - (Pai *et. al* 2004). Ainda nessa fase é importante que sejam descritos os procedimentos para seleção dos artigos, retornados pela *string* de busca, bem como os critérios de inclusão e exclusão como forma de permitir a replicação e auditoria da revisão.

A fase seguinte é a de **execução,** na qual ocorrem todas as atividades definidas no protocolo, são elas: a) aplicação da *string* de busca nas fontes de dados selecionadas b) filtro dos estudos retornados pela *string* de busca com base nos procedimentos de seleção de artigos, definidos no protocolo, observando a aplicação dos critérios de inclusão e exclusão durante a realização dessa tarefa.

A fase de **análise** consiste na extração e sintetização de informações advindas dos estudos que foram selecionados após a aplicação de todos os procedimentos de filtragem e seleção de artigos bem como dos critérios de inclusão e exclusão com intuito de responder as questões de pesquisas e assim atender aos objetivos definidos no protocolo da revisão.

Finalmente, a fase de **empacotamento** que ocorre ao longo das outras fases, nela se registra todas as informações geradas durante o processo da revisão sistemática, desde a elaboração do protocolo até a apresentação dos resultados alcançados.

Existem variações da revisão sistemática da literatura como a *quasi*-revisão sistemática da literatura, proposta por Travassos *et al.* 2008 a qual não utiliza dados de comparação (*intervention comparison*) para realizar as buscas, embora todos os outros procedimentos da revisão estejam presentes nesse tipo de estudo. Outra variação foi a utilizada no trabalho de Ribeiro (2015) na qual todo o procedimento da revisão sistemática da literatura foi aplicado, contudo houve restrições em relação ao uso de bases de busca e simplificação no uso da abordagem PICO para a construção da *string* de busca.

Neste trabalho utilizou-se outra variação da revisão da literatura, na qual, assim como em Ribeiro (2015), restringiu-se a quantidade de fontes de dados e simplificou-se a estratégia de PICO. Vale ressaltar que algumas questões de pesquisa e objetivos deste trabalho se assemelhavam aos comumente encontrados em mapeamentos sistemáticos (Petersen *et al.* 2008) e outras eram típicas de revisão sistemática (Kitchenham *et al.* 2007). Por isso denominou-se **estudo baseado em revisão sistemática** a estratégia de revisão sistemática da literatura adotada neste trabalho pelo fato de conter características de diferentes tipos de estudos sistemáticos, revisão e mapeamento, como pode ser notado ao longo da próxima seção (3.2).

3.2 Questão de Pesquisa e Dimensão

As questões Q01, Q01.1, Q01.2, são típicas de um mapeamento sistemático, pois as respostas delas tem o objetivo de prover uma visão geral da área de pesquisa (**métricas OO**), identificando a quantidade e o tipo de pesquisas e resultados disponíveis dentro dela. Enquanto que Q02, Q02.1, Q02.2, Q02.3, são próprias de revisões sistemáticas já que são

mais definidas, focadas em um ponto específico da área de estudo, no caso **ferramentas de coleta automática de métricas OO**.

A seguir, na tabela 2 são mostradas as questões de pesquisa desta revisão bem como a justificativa para cada uma delas.

Tabela 2. Organização das Questões

Item	**Descrição**	**Justificativa**
Q 01	Quantas e quais são as métricas OO de código fonte coletadas automaticamente?	Identificar as métricas OO para as quais existem ferramentas capazes de coletá-las automaticamente, de modo a apoiar a escolha e o 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.

Item	Descrição	Justificativa
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.

Na tabela 3 podem ser visualizadas as dimensões desta revisão. Nota-se que **não há dados de comparação** conforme mencionado na seção anterior.

Tabela 3. 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.

3.3 Estratégia de Busca e Fonte de Dados

Segundo Mafra e Travassos (2006) 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 e permitem exportar os resultados da aplicação da *string* de busca, pesquisa por palavras-chave e expressões lógicas (Saraiva et al.,2012). Embora a base de dados **ACM** contenha um grande número de estudos da área de computação, ela não foi selecionada pois a **IEEE** indexa a maioria dos estudos contidos nela quando a *string* de busca definida neste trabalho foi executada nas duas bases.

Para que a *string* de busca fosse considerada adequada ela deveria recuperar os artigos de controle[6] quando executada nas bases selecionadas e aprovada por outros dois pesquisadores experientes da área. Assim, a seguinte expressão de busca foi definida.

Tabela 4. 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)))

[6] Artigos relevantes da área previamente identificados e que para este trabalho são os artigos T6, T10 e T32 .

A expressão de busca foi ajustada para a máquina de busca Scopus na qual foi adicionada a condição TITLE-ABS-KEY (tabela 4), 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 esse motivo na expressão de busca foi adicionado esse termo.

3.4 Procedimentos e Critérios para Seleção de Estudos

Os procedimentos para seleção dos estudos foram divididos 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 1 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 e 2) Armazenamento das publicações excluídas;

3. Nos estudos selecionados na fase 2, 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 2. 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 5, 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 5. Informações Extraídas

Informação a ser Extraída	Q01	Q02
Título, 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

A tabela 6 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 6. Critérios de Inclusão e Exclusão de Estudos

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 conteúdo;

3.5 Síntese e Apresentação dos Resultados

Nesta seção são apresentados os resultados, sumarizados, obtidos em cada fase do estudo (descritas no item anterior, 3.4) bem como a definição dos procedimentos adotados. Na fase 1, a *string* de busca foi executada nas bibliotecas digitais escolhidas no dia 18/11/2014 sem utilizar uma data limite de início. Foram recuperados 577 estudos, incluindo os duplicados, conforme informações apresentadas na tabela 7.

Tabela 7. Resultados da Fase 1

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 2. A tabela 8 sintetiza os resultados obtidos nesta fase.

Tabela 8. Resultados da Fase 2

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 8, na fase 2 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 e 83 selecionados para a fase seguinte. Na fase 3, 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 3.3. Na figura 4 é possível visualizar o resumo dos resultados das três fases.

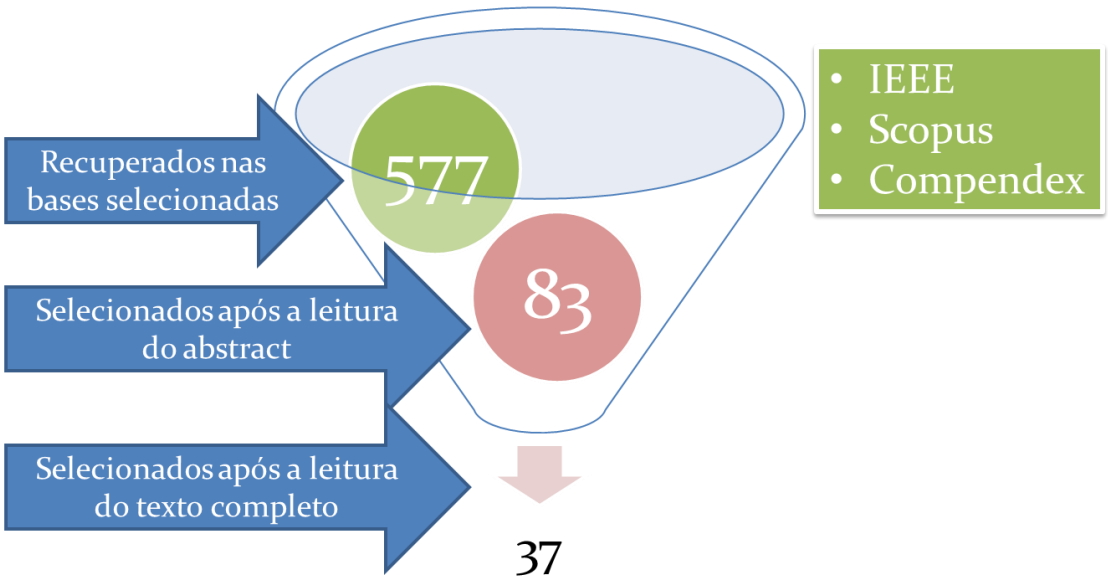


Figura 4. Resumo das três Fases Definidas no Protocolo desta Revisão.

3.6 Considerações sobre o Capítulo

Neste capítulo foi exposta a abordagem metodológica para o desenvolvimento deste trabalho, a qual iniciou-se com uma pesquisa informal da literatura que teve como resultado a identificação da necessidade de realizar uma RSL seguida pela elaboração do protocolo para realização da mesma. O objetivo dessa RSL é reunir informações úteis, relatadas pela literatura, para realizar a Coleta Automática de Métricas Orientadas a Objetos identificando os principais elementos envolvidos nessa tarefa.

4. ANÁLISE DOS RESULTADOS

Este capítulo apresenta os resultados da RSL obtidos através da aplicação e execução do protocolo apresentado no capítulo três. Esses resultados podem ser divididos em duas partes conforme descrição a seguir:

- **Visão geral**: apresenta os resultados e análise de uma maneira mais ampla, sem focar especificamente em uma das questões de pesquisa (ver tabela 2). O objetivo disso é proporcionar uma visão geral da área para contextualizar as respostas às questões de pesquisa que são baseadas em pontos específicos dos resultados obtidos.

- **Respostas às questões**: apresenta os principais resultados utilizados para responder as questões de pesquisa bem como a análise realizada para tal.

4.1 Visão Geral

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 artigos em relação às bases utilizadas pode ser visualizada conforme figura 5.

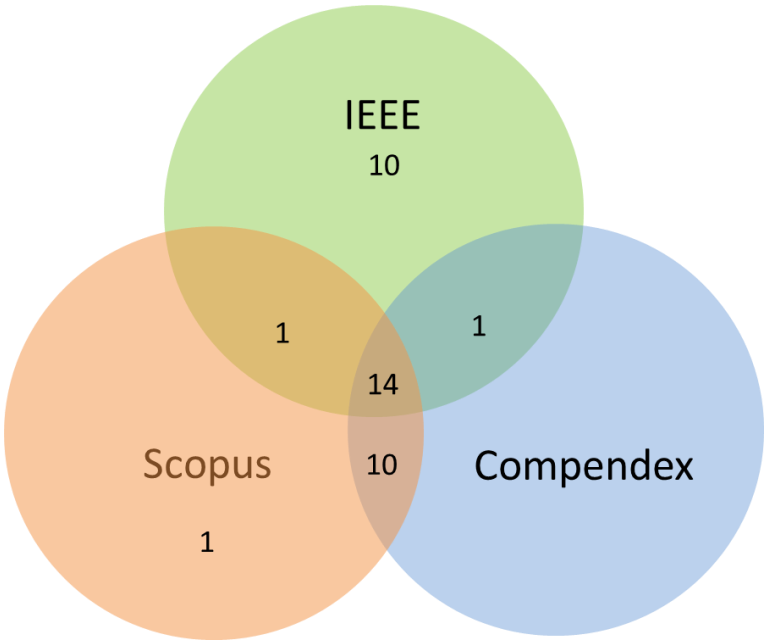


Figura 5. Distribuição dos Estudos Selecionados por Base

Na figura 5 pode-se notar que a IEEE concentrou o maior número de estudos (10) encontrados exclusivamente em uma base, representando quase 30% dos estudos selecionados nesta revisão. Além disso, a IEEE contém 26 (70%) artigos selecionados o que é praticamente o que Scopus e Compendex somaram juntas (27). Isso demonstra a importância da utilização dessa base nesse tipo de estudo. Outro fato interessante diz respeito a Scopus e Compendex, dos 27 estudos recuperados pelas duas bases somente 3 não estão em ambas, sugerindo uma cobertura equivalente de estudos entre essas bases, pelo menos no caso deste trabalho e com a *string* de busca executada.

De acordo com a distribuição geográfica dos estudos (ver figura 6), os 37 artigos selecionados (ver anexo I) foram desenvolvidos em 20 países diferentes. A maioria desses artigos se concentrou em instituições dos Estados Unidos (9 artigos que correspondem a cerca de 25% do total). Uma grande diferença em relação aos seguintes: Romênia, Índia e Canadá todos com 3 artigos cada um que corresponde a 8% do total. Logo após vem Brasil, Holanda e Itália cada um contribuindo com 2 artigos. Por fim, os outros 13 países cada um com um artigo.

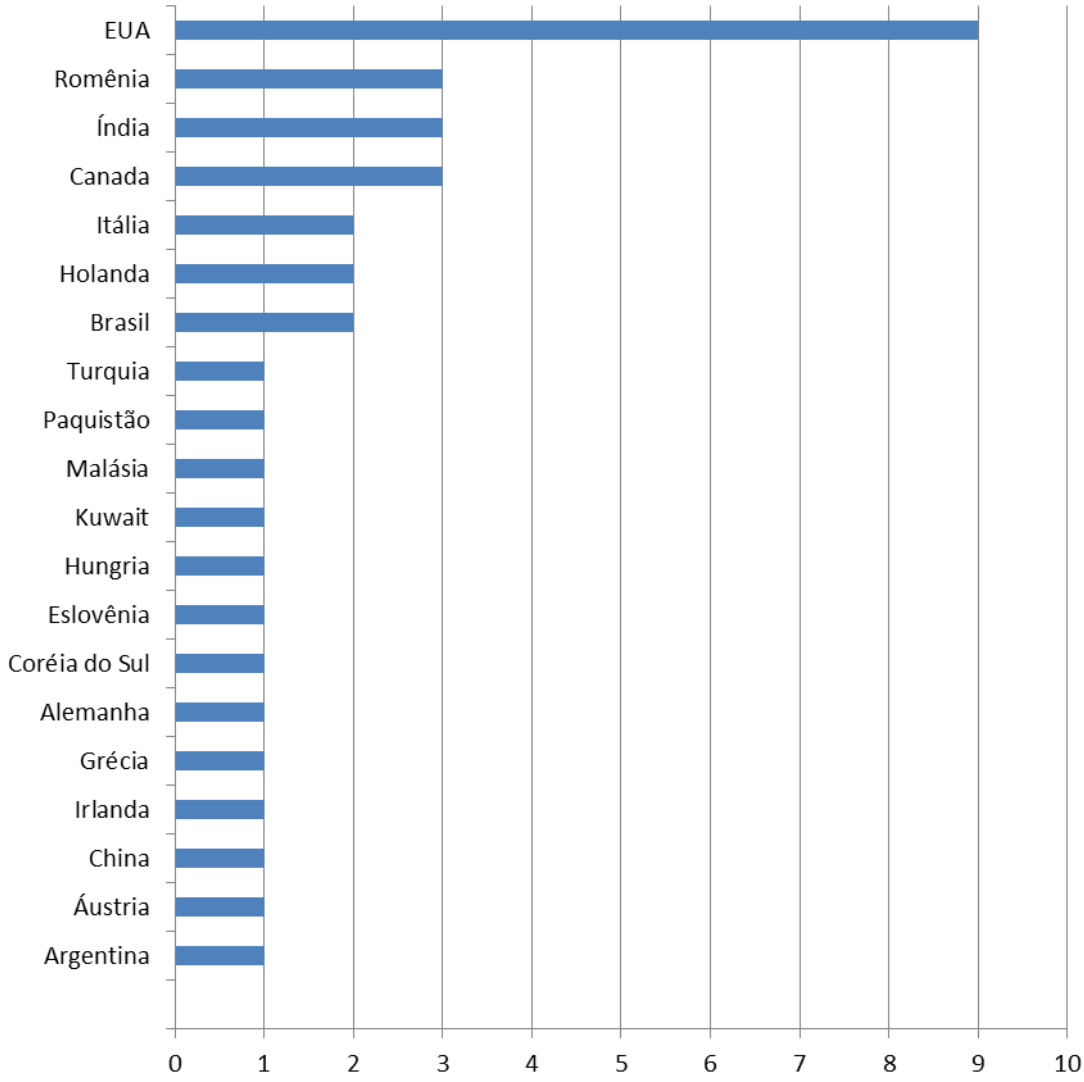


Figura 6. Distribuição dos Estudos pelos Países das Instituições dos Pesquisadores

A distribuição dos artigos selecionados ao longo dos anos é apresentada no gráfico da figura 7 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.

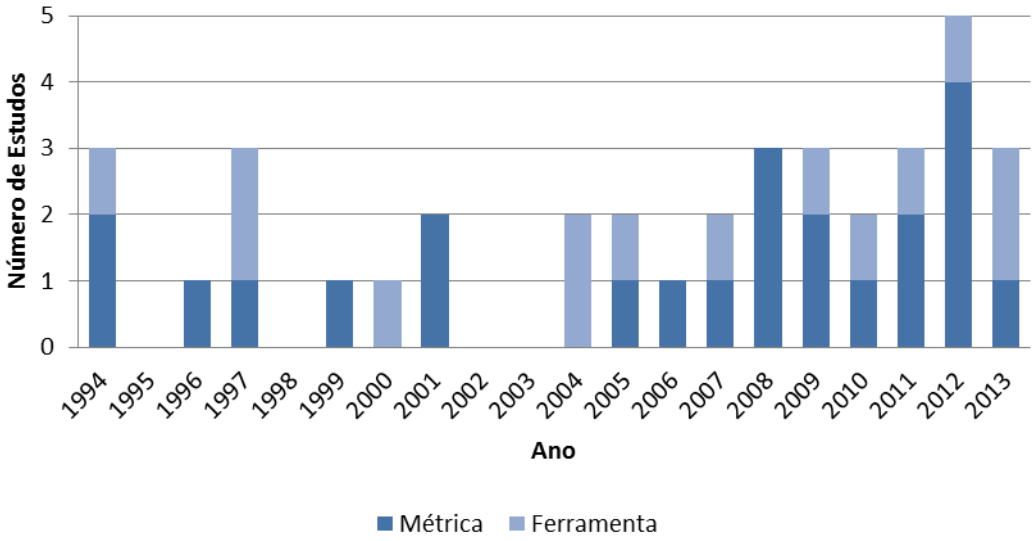


Figura 7. Distribuição dos Estudos por Ano

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.

4.2 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. Nesse apêndice além das métricas identificadas estão também as referências onde foram encontradas, as subcaracterísticas de qualidade que elas estão relacionadas (ver seção 4.3) bem como a classificação delas.(ver seção 2.2.1).

No gráfico da figura 8 nota-se a elevada quantidade de métricas de acoplamento 71 (41%), seguida das relacionadas com complexidade 46 (25%), coesão 26 (15%) e tamanho 24 (13%). Por fim, encapsulamento e polimorfismo somam 10 (6%). Esses resultados sugerem que acoplamento é uma propriedade importante já que muitas métricas estão relacionadas com ela, assim como complexidade e tamanho que juntas somam 70 (40%) e geralmente são utilizadas em conjunto. Por outro lado, as propriedades encapsulamento e polimorfismo

parecem não terem sido suficientemente exploradas ou não influenciam tão fortemente na qualidade do software já que existem poucas métricas relacionadas com essas propriedades.

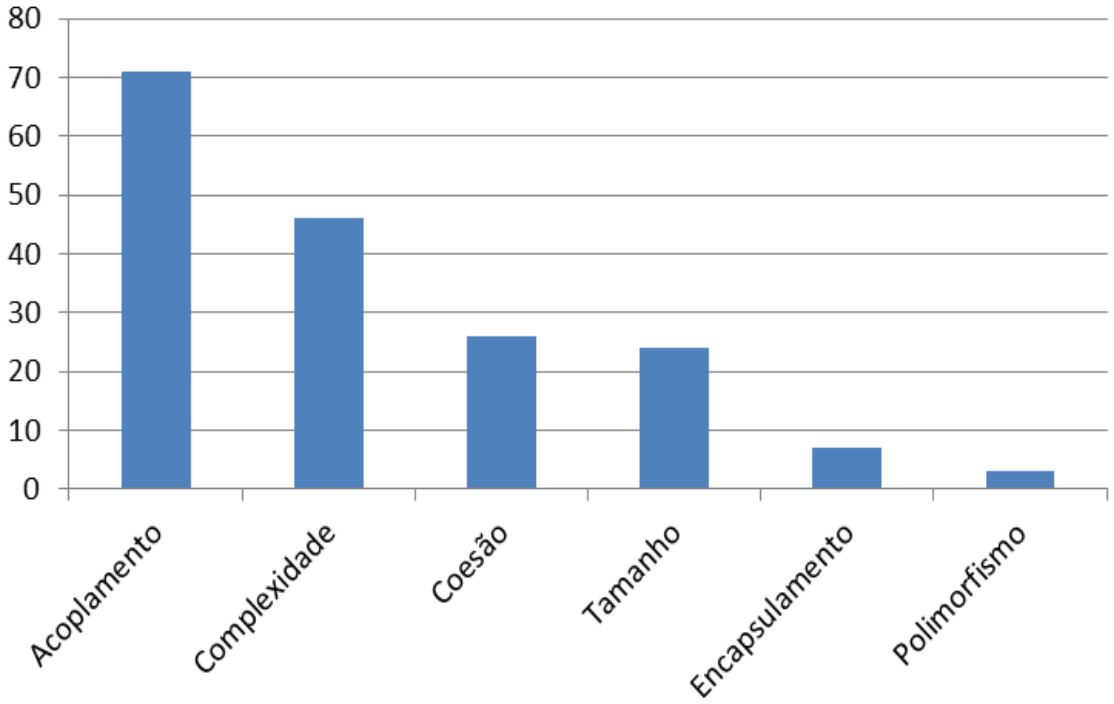


Figura 8. Distribuição das Métricas por Propriedade Relacionada

Vale ressaltar que algumas métricas (do Apêndice 2 e da figura 10) 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 [T.3] [T.21] [T.22] e que dificilmente poderiam ser utilizadas para outro objetivo ou contexto. Por exemplo, Mayrand e Leblanc [T.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 4.4 a qual destaca 28 métricas como as mais citadas.

4.3 Questão 01.1: Quais características de qualidade podem ser avaliadas pelas métricas OO identificadas?

Samoladas *et al.* [T.29] elaborou um modelo de qualidade baseado na ISO/IEC 9126 (2001) 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 (ISO/IEC 9126, 2001).

No que se refere à manutenibilidade, o estudo [T.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 [T.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 9126 (2001), conforme o seguinte procedimento: observou-se que as métricas relacionadas com acoplamento eram vinculadas a Modificabilidade e Estabilidade em [T.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 9.

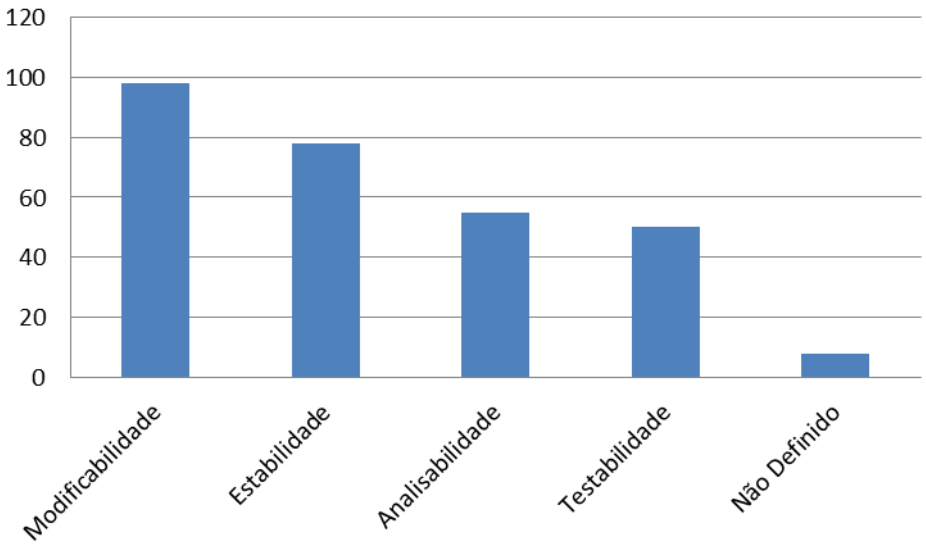


Figura 9. Quantidade de Métricas por Subcaracterística de Qualidade

Na figura 9 pode-se notar a grande ocorrência de métricas relacionadas à 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.

4.4 Questão 01.2: Quais são as métricas mais frequentes?

O gráfico da figura 10 destaca as 28 métricas mais referenciadas pelos estudos selecionados neste trabalho. Dentre elas pode-se destacar o conjunto de seis métricas OO proposto por Chidamber e Kemerer [T.10] (DIT, NOC, CBO, LCOM, RFC, WMC). Elas aparecem entre as 10 métricas mais citadas.

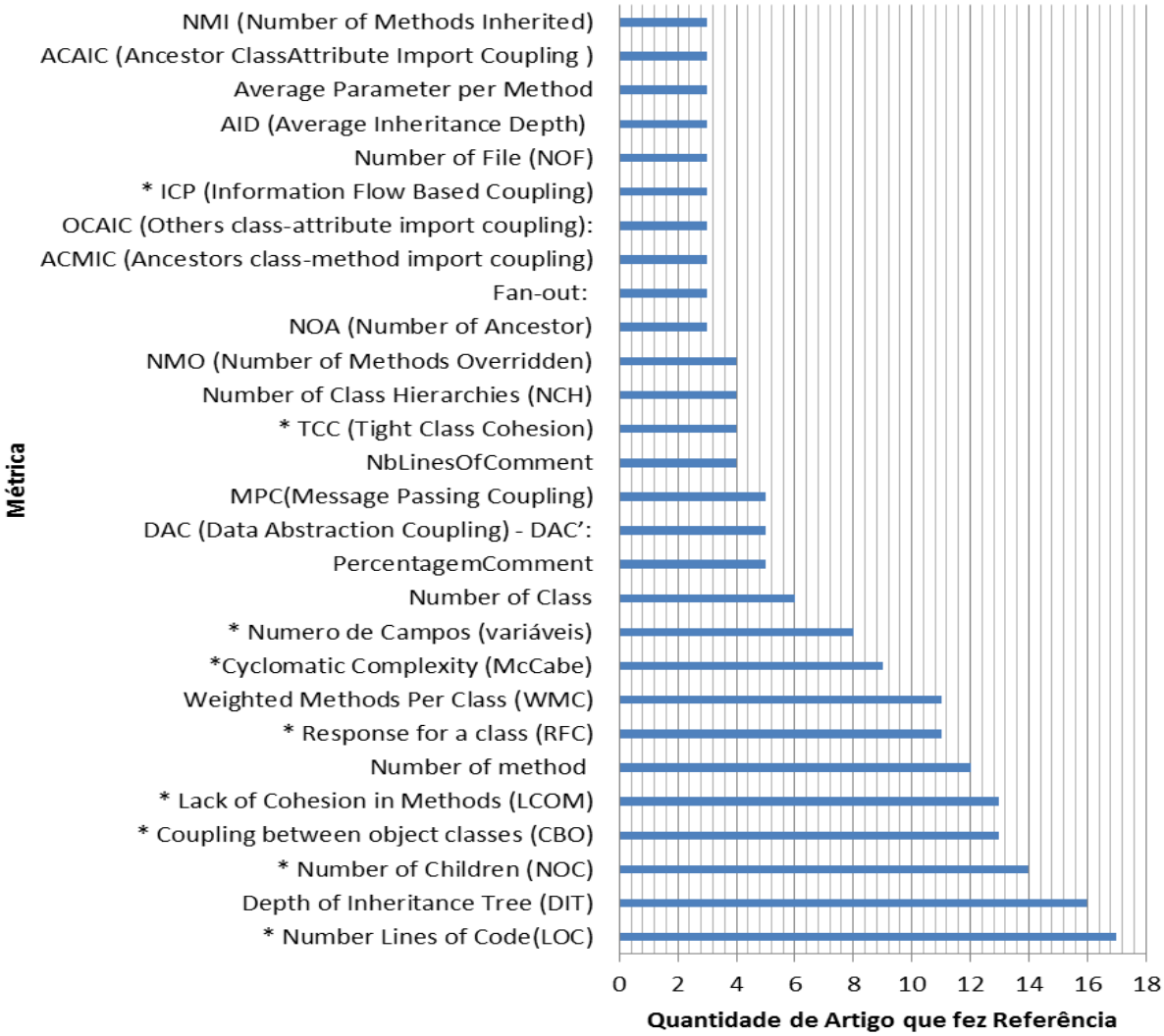
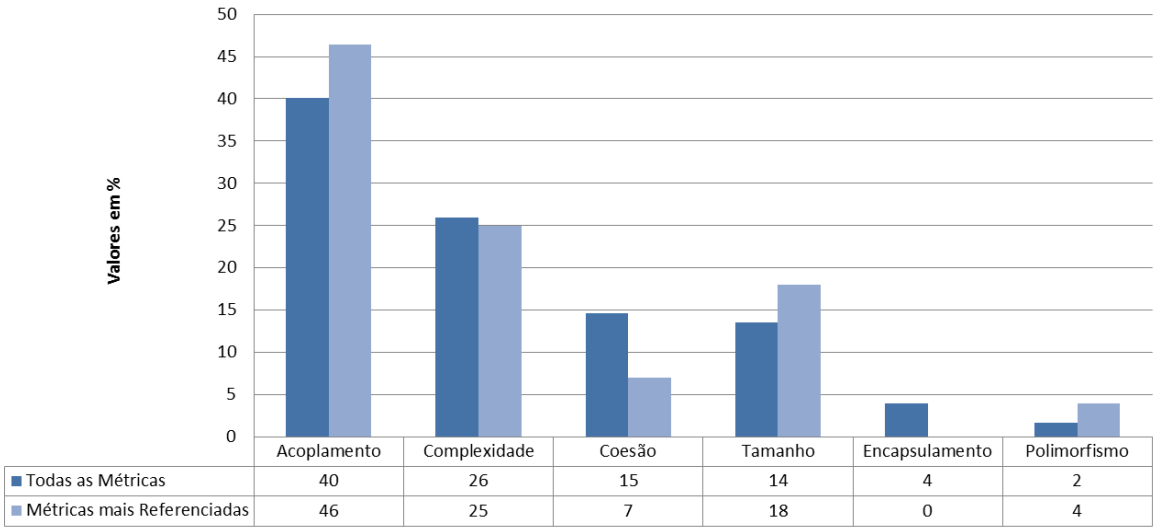


Figura 10. Métricas mais Referenciadas

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

As métricas mais referenciadas seguiram a tendência das 177 métricas identificadas nesta revisão (ver figura 11), no que se refere a alta concentração de métricas relacionadas a propriedade de acoplamento, seguida por complexidade variando apenas o percentual em 6% e 1%, respectivamente. A maior diferença foi em relação a propriedade coesão que levando em consideração todas as métricas obteve 15% das métricas enquanto que quando restringida as mais referenciadas obteve somente 7% . Efeito exatamente oposto ao da propriedade

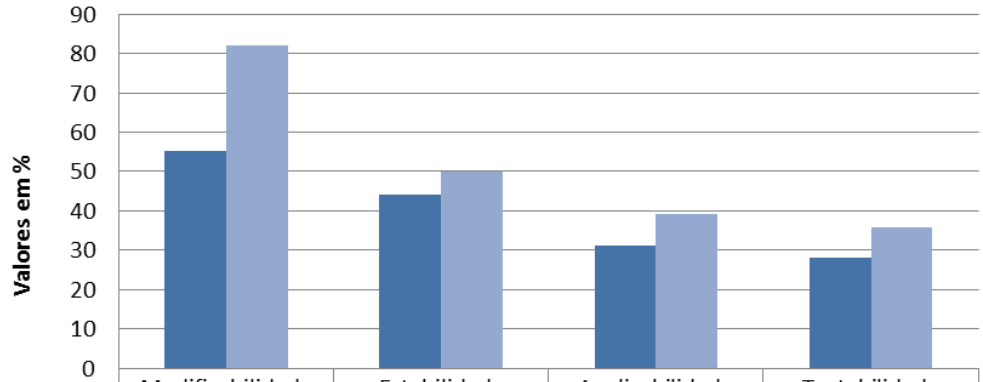
tamanho que tinha 14% de todas as métricas identificadas e das mais referenciadas alcançou 18%. Polimorfismo e encapsulamento obtiveram um baixo número de métricas em ambos universos.



	Acoplamento	Complexidade	Coesão	Tamanho	Encapsulamento	Polimorfismo
Todas as Métricas	40	26	15	14	4	2
Métricas mais Referenciadas	46	25	7	18	0	4

Figura 11. Comparação das 177 Métricas Catalogadas e as 28 mais Referenciadas com Relação as Propriedades que elas Mensuram

Nessa mesma linha uma comparação das subcaracterísticas de qualidade avaliadas pelas 177 métricas identificadas nesta revisão e as 28 mais referenciadas pode ser visualizada no gráfico da figura 12.



	Modificabilidade	Estabilidade	Analisabilidade	Testabilidade
Todas as Métricas	55	44	31	28
Métricas mais Referenciadas	82	50	39	36

Figura 12. Comparação das 177 Métricas Catalogadas e as 28 mais Referenciadas com Relação as Subcaracterísticas que elas Mensuram

Nesse gráfico (figura 12), pode-se notar que a ordem das subcaracterísticas com maior número de métrica foi a mesma ambos universos (Modificabilidade com maior número, seguida por: Estabilidade, Analisabilidade e Testabilidade). No entanto, pode-se destacar a subcaracterística Modificabilidade que não só se mantém como a que possui o maior percentual de métricas mais também o aumenta quando o universo é restrito as 28 métricas mais referenciadas, ratificando a sua importância. As outras subcaracterísticas também elevaram seu percentual, porém de maneira atenuada.

4.5 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 9. 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 dos mesmos 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 (2015) 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 [T.19] e [T.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 métrica vinculada a ela (tabela 9), 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 9. 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, 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	Metalinguagem	-
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,

Ano	Ref	Nome	Contexto	Linguagem	Métrica
					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 13, 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.

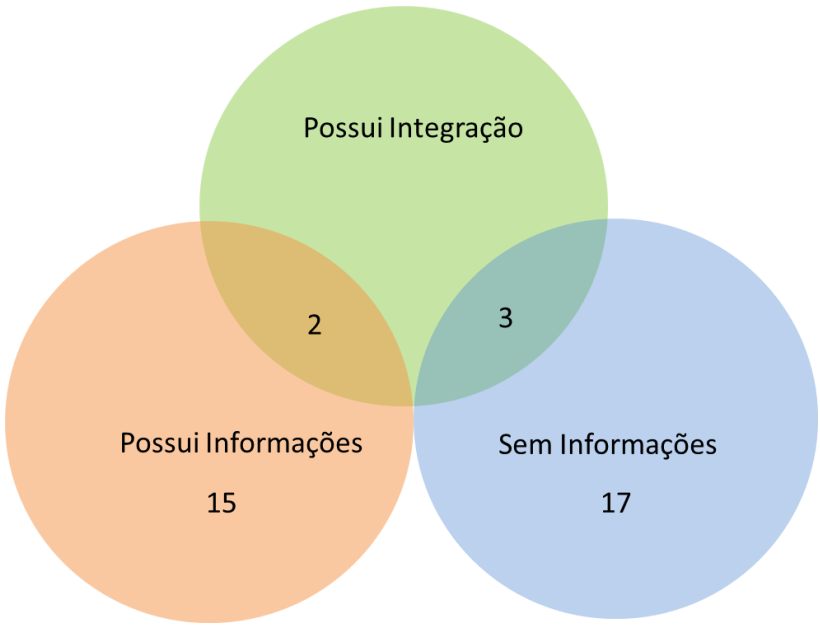


Figura 13. Distribuição dos Estudos com base nas informações sobre a ferramenta de Coleta

Conforme figura 13, 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 14, 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.

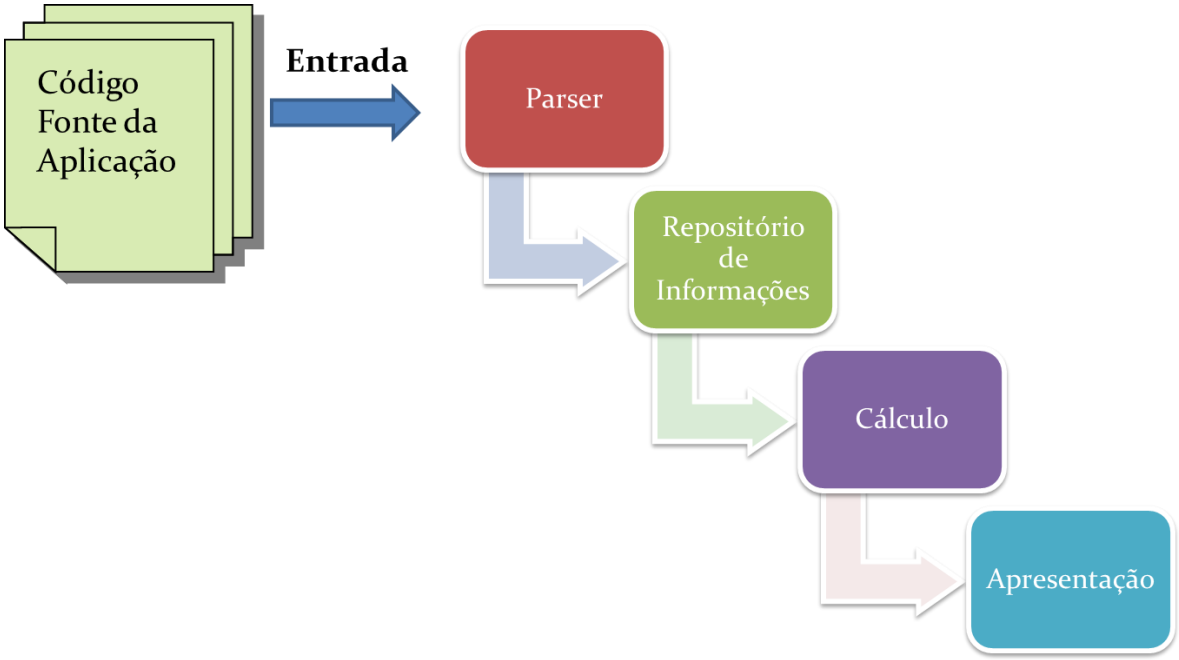


Figura 14. Componentes das Ferramentas de Coleta de Métricas

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 18 artigos que apresentaram informações sobre as suas ferramentas de coleta, 15 utilizam repositórios de informações modelados em um banco de dados relacional enquanto 2 utilizaram *Abstract Syntax Tree* e [T.1] não deixou claro como armazena essas informações. Todas as ferramentas que utilizaram um banco de dados como repositório de informações coletavam métricas somente para linguagens orientadas a objeto. Por outro lado, dos 2 artigos que utilizaram *Abstract Syntax Tree*: o estudo [T.6] realiza a coleta para linguagens orientada a objeto enquanto que em [T.26] isto é feito para linguagens procedurais.

4.6 Questão 02.1: As ferramentas possuem integração com outras do processo de software?

Cinco artigos (figura 13) relataram possuir integração com outras ferramentas utilizadas no processo de desenvolvimento de software. Os artigos [T.11] [T.22] [T.26] e [T.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 [T.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.

4.7 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 comparar os valores das métricas de várias versões do software ao longo do tempo ([T.3] [T.4] [T.25] [T.29]) ou com valores como: mínimo, máximo, média entre outros valores obtidos por meio da coleta ([T.12] [T.16] [T.18] [T.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 ([T.1] [T.5] [T.6] [T.9] [T.15] [T.20] [T.22]

[T.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 [T.33], no qual analisa a relação entre as diferentes métricas para medir tamanho e reutilização, e em [T.28] que avalia a relação de métricas de complexidade ao índice de correções do software.

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

Histogramas são usados para comparar valores de uma única métrica à medida que o número de classes do software aumentava [T.8] [T.10] e [T.12] para comparar valores entre duas métricas. Em [T.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 [T.16] [T.21] [T.31]. Os artigos [T.24] [T.35] utilizam o gráfico de linha para avaliar métricas ao longo do tempo, identificando as versões do software, enquanto que [T.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 desta seção.

4.8 Questão 02.3: Para quais linguagens de programação as ferramentas permitem a extração de métricas?

Foram encontradas sete linguagens diferentes para as quais as ferramentas de coleta, identificadas nessa 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 15, 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.

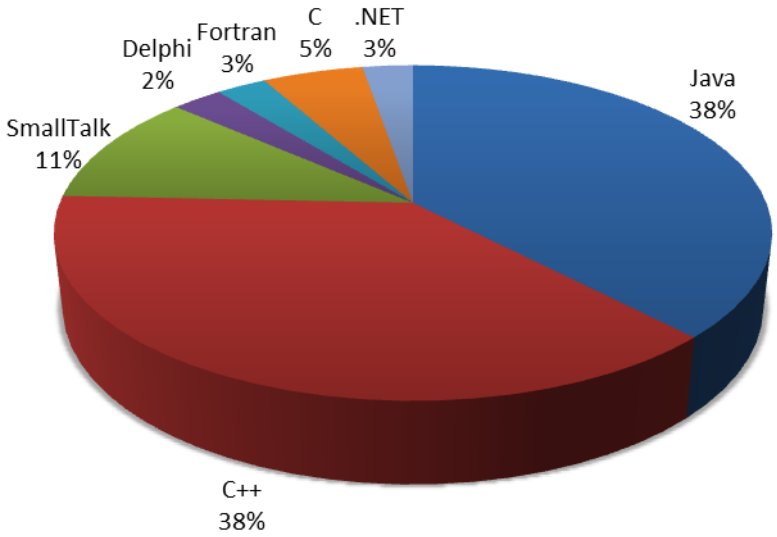


Figura 15. Quantidade de Ferramentas que Coletam Métricas por Linguagem de Programação

4.9 Considerações sobre o Capítulo

Os resultados obtidos na RSL foram apresentados nesse capítulo em forma de respostas as questões de pesquisa definidas no protocolo da revisão (capítulo 3). Esses resultados são resumidos a seguir:

- 577 estudos recuperados;
- 37 estudos selecionados;
- 177 métricas catalogadas;
- 28 métricas mais referenciadas;
- 18 ferramentas de coleta identificadas que realizam essa tarefa para 7 linguagens de programação diferentes;

- C++ e Java foram as linguagens com maior número de ferramentas de coleta.

Além disso, uma análise desses resultados foi feita constatando-se uma carência de ferramentas de coleta automática integradas com outras utilizadas no processo de desenvolvimento e um conjunto de procedimentos padrão para coleta automática de métricas foi identificado.

5. CONCLUSÃO

Neste capítulo serão apresentadas as limitações e ameaças à validade deste estudo; a principais contribuições; os trabalhos futuros e algumas considerações finais sobre este trabalho.

5.1 Limitações do Estudo e Ameaças à Validade

Uma das principais ameaças à validade para este trabalho é a seleção incompleta ou inadequada de estudos. Apesar ter sido seguida 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 que alguns artigos importantes terem utilizados termos diferentes dos que foram usados na *string* de busca o que acarretaria na não recuperação desse estudo.

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/IEC (2007), foi baseado no trabalho de Samoladas *et al.* [T.29] e isso pode ter limitado essa classificação já que em [T.29] foi restringida à característica de manutenibilidade.

5.2 Considerações Finais e Trabalhos Futuros

Este trabalho apresentou um estudo baseado em revisão sistemática o qual obteve como resultado: 177 métricas que podem ser coletadas automaticamente, das quais 28 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 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.

REFERÊNCIAS BIBLIOGRÁFICAS

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

Abreu, F. B.."Design Metrics for Object-Oriented Software Systems". UNESC/ISEG. 7th ERCIM Database Research Group Workshop on Object Oriented Darabases, Lisbon, 15-16 May (1995).

Aggarwal K., Yogesh S., Arvinder K., Ruchika M. "Empirical Study of Object-Oriented Metrics" Journal of Object Technology, (2006).

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

Basili, V.R. e Musa, J.D. "The Future Engineering of Software: A Management Perspective". IEEE Computer Society Press Los Alamitos, CA, USA, (1991).

Benlarbi, S., Melo, W.L. "Polymorphism Measures for Early Risk Prediction" Software Engineering. Proceedings of the International Conference on, (1999).

Biolchini, J. C. D. A. et al. "Scientific Research Ontology to Support Systematic Review in Software Engineering". Advanced Engineering Informatics, Amsterdam, 21, n. 2. 133-151. DOI 10.1016/j.aei.2006.11.006, (2007).

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

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

Eclipse Metrics Plugin. Disponível em: <http://eclipse-metrics.sourceforge.net/>. Acesso em 4 de Abril de 2015.

Ferreira, K. A.; Bigonha, M. A.; Bigonha, R. S.; Mendes, L. F. & Almeida, H. C. "Identifying Thresholds for Object-Oriented Software Metrics.". Journal of Systems and Software, (2012).

Filó, T. G. S., “Identificação De Valores Referência Para Métricas De Softwares Orientados Por Objetos”, Dissertação de Mestrado em Ciência da Computação, UFMG, (2014).

Garvin, D. A. “Gerenciando a Qualidade: a visão estratégica e competitiva”. Tradução de João Ferreira Bezerra de Souza. Rio de Janeiro: Qualitymark, (1992).

Harman, M. "Why Source Code Analysis and Manipulation will Always be Important". Em 10th IEEE Working Conference on Source Code Analysis and Manipulation, SCAM, (2010)

ISO/IEC 14598: Tecnologia de informação – avaliação de produto de software. Parte 5: processo para avaliadores. Rio de Janeiro, (2001).

ISO/IEC 25000, Software Engineering – Software Product Quality Requirements and Evaluation (SQuaRE) - System and Software Quality Models. Geneva, p.43, (2011).

ISO/IEC 25020, Software and System Engineering - Software Product Quality Requirements and Evaluation (SQuaRE) – Measurement Ref. Model and Guide, (2011).

ISO/IEC 9126-1 - Information Technology - Software Product Quality - Quality Model. Geneva, p. 34. 2001.

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

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

Juran, J. M. et al. “Quality Control Handbook”. New York, McGraw Hill, (1974).

Kitchenham, B.; PFLEEGER, S. L. “Software Quality: The Elusive Target”. IEEE Software, v. 1, n. 13, p. 12-21, (1996).

Lehman et al. “Metrics and laws of software evolution - the nineties view”. Fourth International Software Engineering, 13 (6):697-708, (1997).

Lima, Carla A.; Reis, Rodrigo Q. “Laboratório de Engenharia de Software e Inteligência Artificial: Construção do ambiente WebAPSEE”. Revista ProQuality (UFLA), v. 3, p. 43-48, 2007.

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

Nascimento, L. M. A., “Uma Abordagem Automatizada de Medição em Processos de Software”, Dissertação de Mestrado em Ciência da Computação, UFPA, 2007.

Pai, M. et al. “Systematic Reviews and Meta-Analyses: An Shown, Step-by-Step Guide”. The National Medical Journal Of India, 17, n. 2. 86-95, (2004).

Petersen, K.; Feldt, R. ; Mujtaba, S.; Mattsson, M. “Systematic Mapping Studies in Software Engineering”. EASE'08 Proceedings of the 12th international conference on Evaluation and Assessment in Software Engineering, (2008).

Poels, Geert. “An Analytical Evaluation of Static Coupling Measures for Domain Object Classes”. Department of Applied Economic Sciences, Katholieke Universiteit Leuven, ECOOP98-OOPM, Bélgica, (1998).

Pressman, R.S “Engenharia de Software: uma abordagem profissional”, 7ª Edição, McGraw Hill do Brasil, (2011).

Ribeiro, T. “Alinhando Perspectivas De Qualidade Em Código Fonte A Partir De Estudos Experimentais – Um Caso Na Indústria” Tese de Mestrado Coppe, (2014).

Ribeiro, T. V. “Medição Em Processos De Software: Da Aplicação Prática À Melhoria De Uma Ferramenta De Planejamento E Análise”, Trabalho de Conclusão de Curso de Ciência da Computação, UFPA, 2011.

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

SEI. CMMI for Development (CMMI-DEV). Carnegie Mellon University. Pittsburgh, p. 482.. (CMU/SEI-2010-TR-033). Versão 1.3., (2010).

SOFTEX. Associação Para Promoção Da Excelência Do Software Brasileiro. MPS.BR – Guia de Implementação de Software – Parte 1: Nível G:2013, 2013**.** Disponível em: <http://www.softex.br/mpsbr/guias/>. Acesso em 8 de Abril de 2015.

Travassos, G. H. et al. An Environment to Support Large Scale Experimentation in Software Engineering. Proceeedings of the 13th IEEE International Conference on Engineering of Complex Computer Systems. Belfast: IEEE. 2008. p. 193-202. DOI 10.1109/ICECCS..30. (2008).

Travassos, G. H.; Kalinowski, M. iMPS 2013: Evidências sobre o Desempenho das Empresas. SOFTEX. Campinas, p. 102. 2014. 978-85-99334-75-1. Disponível em: http://www.softex.br/wp-content/uploads/2013/08/Livro_iMPS-2013-PT_v3.pdf, (2013).

REFERÊNCIAS BIBLIOGRÁFICAS REVISÃO SISTEMÁTICA DA LITERATURA

T1. Aguero, M. and Madou, F. et al. “Enhancing source code metrics scope through artificial intelligence” IMETI 2010 - 3rd International Multi-Conference on Engineering and Technological Innovation, Proceedings (2010).

T2. Ajmal, O. and Missen, M.M.S. et al. “EPlag: A two layer source code plagiarism detection system” Digital Information Management (ICDIM), Eighth International Conference on (2013).

T3. Al Dallal, J. “Constructing models for predicting extract subclass refactoring opportunities using object-oriented quality metrics” Information and Software Technology (2012).

T4. Alikacem, H. and Sahraoui, A. “A metric extraction framework based on a high-level description language” IEEE International Working Conference on Source Code Analysis and Manipulation, SCAM (2009).

T5. Banker, D., and Kauffman, J. et al. “Automating output size and reuse metrics in a repository-based computer-aided software engineering (CASE) environment” Software Engineering, IEEE Transactions (1994).

T6. Bansiya, J. and Davis, C “Automated metrics and object-oriented development” Dr. Dobb's Journal (1997).

T7. Bavota, G. and De Lucia, A. et al. “Supporting extract class refactoring in Eclipse: The ARIES Project” Software Engineering (ICSE), 2012 34th International Conference on (2012).

T8. Briand, L. and Wust, J. et al. “Using coupling measurement for impact analysis in object-oriented systems” Software Maintenance (1999).

T9. Brooks, C. and Buell, C. “A tool for automatically gathering object-oriented metrics” Aerospace and Electronics Conference (1994).

T10. Chidamber, R. and Kemerer, F. “A Metrics Suite for Object-Oriented Design”. IEEE Trans.software Eng., Vol. 20, No. 6 (1994).

T11. Erdemir, U. and Tekin, U. et al. “E-Quality: A graph based object oriented software quality visualization tool” Visualizing Software for Understanding and Analysis (VISSOFT) (2011).

T12. Ferenc, R. and Siket, et al. “Extracting facts from open source software” Software Maintenance (2004).

T13. Hamou-Lhadj, A., Lethbridge, T. “Summarizing the content of large traces to facilitate the understanding of the behaviour of a software system” IEEE International Conference on Program Comprehension (2006).

T14. Hericko, M. and Rozman, I. et al. “OO metrics data gathering environment” Technology of Object-Oriented Languages (1997).

T15. Kaur, K., Singh, H. “Towards a valid metric for class cohesion at design level” International Conference on Emerging Applications of Information Technology, EAIT (2011).

T16. Kayarvizhy, N. and Kanmani, S. “Analysis of quality of object oriented systems using object oriented metrics” Electronics Computer Technology (ICECT) (2011).

T17. Kozak, C. and Squire, M. “A secondary data archive for code-level debian metrics” International Workshop on Replication in Empirical Software Engineering Research, RESER (2011).

T18. Kwiatkowsk, M. and Verhoef, C. “Recovering management information from source code” Science of Computer Programming (2013).

T19. Lee, J. and Jung, W. “Automated metric visualizations for analyzing source code repositories” International Conference on Information Science and Applications, ICISA (2013).

T20. Marinescu, C. and Marinescu, Ret al. “Towards a simplified implementation of object-oriented design metrics” Software Metrics (2005).

T21. Mayrand, J. and Leblanc, C. “Experiment on the automatic detection of function clones in a software system using metrics” Software Maintenance (1996).

T22. McQuillan, A. and Power, F. “A metamodel for the measurement of object-oriented systems: An analysis using alloy” Conference on Software Testing, Verification and Validation, ICST (2008).

T23. Mihancea, F., Marinescu, R. “Discovering comprehension pitfalls in class hierarchies” Conference on Software Maintenance and Reengineering, CSMR (2009).

T24. Mihancea, P. and Marinescu, R. “Towards the Optimization of Automatic Detection of Design Flaws in Object-Oriented Software Systems” Conference Software Maintenance and Reengineering (2005).

T25. Pereira, M. and Mellado, R “Software product measurement and analysis in a continuous integration environment” International Conference on Information Technology: New Generations, ITNG (2010).

T26. Plosch, R. and Gruber, H. et al. “Tool support for expert-centred code assessments” International Conference on Software Testing, Verification and Validation, ICST (2008).

T27. Roubtsov, S. and Telea, A. “SQuAVisiT: A software quality assessment and visualisation toolset” IEEE International Working Conference on Source Code Analysis and Manipulation, SCAM (2007).

T28. Saboe, M. “The use of software quality metrics in the materiel release process experience report” Quality Software (2001).

T29. Samoladas, I. and Gousios, G. et al. “The SQO-OSS quality model: Measurement based open source software evaluation” International Federation for Information Processing, IFIP (2008).

T30. Sandhu, P. and, Kaur, H. “Modeling of reusability of object oriented software system” World Academy of Science, Engineering and Technology (2009).

T31. Sato, D. and Goldman, A. et al. “Tracking the evolution of object-oriented quality metrics on agile projects” Lecture Notes in Computer Scienc (2005).

T32. Scotto, M. and Sillitti, A. “A Relational Approach to Software Metrics” Proceedings of the ACM Symposium on Applied Computing, 2, 1536 – 1540 (2004).

T33. Subramanian, G. and Corbin, W. “An empirical study of certain object-oriented software metrics” Journal of Systems and Software (2001).

T34. Taibi, F. “Filtered mining in program code repositories” Information Retrieval Knowledge Management (CAMP), 2012 International Conference on (2012).

T35. Welker, D. and Oman, W. et al. “Development and application of an automated source code maintainability index” Journal of Software Maintenance and Evolution (1997).

T36. Xiaojing, W. and Contreras, A. et al. “Interval-based algorithms to extract fuzzy measures for Software Quality Assessment” Fuzzy Information Processing Society (NAFIPS), 2012 Annual Meeting of the North American (2012).

T37. Xie, T. and Yuan, W. et al. “JBOOMT: Jade Bird Object-Oriented Metrics Tool” Chinese Journal of Electronics, 9, 202-20 (2000).

APÊNDICE A – LISTA DE MÉTRICAS IDENTIFICADAS

#	**Métricas**	**Referência**	**Subcaracterísticas**	**Propriedade**
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	Tamanho
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	Acoplamento
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	Tamanho

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	Acoplamento
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	Coesão
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	Tamanho
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	Complexidade
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	Complexidade
9	* (McCabe) Cyclomatic Complexity	A.18,A.19,A.2,A.25, A.27, A.28, A.29, A.34, A.35	Analisabilidade, Testabilidade	Complexidade
10	* Number of Field	A.11, A.12, A.16, A.2, A.25, A.3, A.4, A.6	Analisabilidade, Testabilidade	Complexidade
11	Number of Class	A.12, A.17, A.36, A.37, A.4, A.6	Analisabilidade, Testabilidade	Tamanho

12	Percentage of Comment	A.1, A.17, A.25, A.29, A.34	Analisabilidade	Complexidade
13	(DAC) Data Abstraction Coupling	A.16, A.22, A.3, A.6, A.8	Modificabilidade, Estabilidade	Acoplamento
14	(MPC) Message Passing Coupling	A.16, A.22, A.3, A.7, A.8	Modificabilidade, Estabilidade	Acoplamento
15	NbLinesOfComment	A.17, A.2, A.25, A.35	Analisabilidade	Complexidade
16	*(TCC) Tight Class Cohesion	A.16, A.22, A.24, A.3	Modificabilidade	Coesão
17	(NCH) Number of Class Hierarchies	A.29, A.36, A.6, A.9	Analisabilidade	Acoplamento
18	(NMO) Number of Methods Overridden	A.11, A.16, A.24, A.4	Modificabilidade, Estabilidade	Polimorfismo
19	(NOA) Number of Ancestor	A.16, A.4, A.6	Estabilidade, Testabilidade	Acoplamento
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	Acoplamento
21	(ACMIC) Ancestors class-method import coupling	A.22, A.4, A.8	Modificabilidade, Estabilidade	Acoplamento
22	(OCAIC) Others class-attribute import	A.22, A.4,A.8	Modificabilidade, Estabilidade	Acoplamento

	coupling			
23	*(ICP) Information Flow Based Coupling	A.16, A.22, A.8	Modificabilidade, Estabilidade	Acoplamento
24	(NOF) Number of File	A.17, A.37, A.4	Analisabilidade, Testabilidade	Tamanho
25	(AID) Average Inheritance Depth	A.16, A.4, A.6	Modificabilidade, Estabilidade	Acoplamento
26	Average Parameter per Method	A.37, A.4, A.6	Analisabilidade, Testabilidade	Complexidade
27	(ACAIC) Ancestor ClassAttribute Import Coupling	A.22, A.4, A.8	Modificabilidade, Estabilidade	Acoplamento
28	(NMI) Number of Methods Inherited	A.16, A.33, A.4	Modificabilidade, Estabilidade	Acoplamento
29	(OCMIC) Coupling to other classes, i.e., 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	Acoplamento
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	A.22,A.8	Modificabilidade, Estabilidade	Acoplamento

	parameter (function pointer) to a method of class C.			
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	Acoplamento
32	(OCMEC) Number of distinct classes used as types of the parameters of the methods in the class	A.22, A.3	Modificabilidade, Estabilidade	Acoplamento
33	DAM (Data Acess Metric)	A.36, A.6	Modificabilidade, Estabilidade	Encapsulamento
34	Number of Module (NOM)	A.37, A.4	Analisabilidade, Testabilidade	Tamanho
35	DCC (Direct Class Coupling)	A.36, A.6	Modificabilidade, Estabilidade	Acoplamento
36	CIS (Class Interface Size)	A.36, A.6	Modificabilidade	Encapsulamento
37	Number of Polymorphic Methods	A.36, A.6	Modificabilidade, Estabilidade	Polimorfismo

38	NPA (Number of PublicAttributes)	A.24, A.6	Modificabilidade, Estabilidade	Encapsulamento
39	Number of Calls (NOCALLS)	A.23, A.27	Modificabilidade, Estabilidade	Complexidade
40	NOD (Number Of Descendants)	A.16, A.4	Estabilidade, Testabilidade	Acoplamento
41	ANA (A measure of generalization-specialization aspect of design)	A.36, A.6	Estabilidade, Testabilidade	Acoplamento
42	Maintainability index	A.28, A.35		Complexidade
43	program effort (E)	A.28, A.35	Analisabilidade, Testabilidade	Complexidade
44	* Average Derived Classes per Class	A.23, A.6	Estabilidade, Testabilidade	Acoplamento
45	Numero de instruções	A.2, A.25	Analisabilidade	Tamanho
46	Afferent coupling at type level (TypeCa)	A.25, A.31	Modificabilidade, Estabilidade	Acoplamento
47	DCMEC (Descendants class-method export coupling)	A.22, A.4	Modificabilidade, Estabilidade	Acoplamento
48	Efferent coupling at type level (TypeCe)	A.25, A.31	Modificabilidade, Estabilidade	Acoplamento
49	DCAEC (Descendants class-attribute export	A.22, A.4	Modificabilidade, Estabilidade	Acoplamento

	coupling)			
50	NIC(número de classes independentes)	A.4, A.6	Analisabilidade	Acoplamento
51	CAM (Coupling among method)	A.11, A.36	Modificabilidade, Estabilidade	Acoplamento
52	MFA (Measure of funcional abstraction)	A.36, A.6	Modificabilidade	Acoplamento
53	NOP (Number of Parent)	A.16, A.4	Estabilidade, Testabilidade	Acoplamento
54	CLD (Class to Leaf Depth)	A.16, A.4	Estabilidade, Testabilidade	Acoplamento
55	TBI (Total base interfaces of system)	A.23, A.4	Analisabilidade	Acoplamento
56	program volume (v)	A.28, A.35	Analisabilidade, Testabilidade	Complexidade
57	CndNbr - Number of Decisions	A.21, A.29	Estabilidade, Testabilidade	Complexidade
58	StmDecNbr - Number of Declaration statements	A.21, A.29	Analisabilidade	Tamanho
59	StrBrcNbr – Number of breaches of structure	A.21	Estabilidade, Testabilidade	Complexidade
60	Number of nested levels	A.29	Modificabilidade, Testabilidade	Acoplamento

61	Number of unconditional jumps	A.29	Modificabilidade, Testabilidade	Complexidade
62	Vocabulary frequency	A.29	Modificabilidade	Coesão
63	NMN (Number of Methods New)	A.4	Modificabilidade, Estabilidade	Tamanho
64	Class comments frequency	A.29	Analisabilidade	Complexidade
65	Average Size of Statements	A.29	Analisabilidade,Modificabilidade	Tamanho
66	ATFD (Access To Foreign Data)	A.24	Modificabilidade, Estabilidade	Acoplamento
67	INAG(Indirect aggregation coupling)	A.8	Modificabilidade, Estabilidade	Acoplamento
68	NEM (Number of external called method)	A.4	Modificabilidade, Estabilidade	Acoplamento
69	NEA (Number of external used Attribute)	A.4	Modificabilidade,Estabilidade	Acoplamento
70	PIMAS	A.8	Modificabilidade,Estabilidade	Polimorfismo
71	aveVG	A.35	Analisabilidade, Testabilidade	Complexidade
72	INTERNAL REUSE	A.5	Modificabilidade	Acoplamento
73	aveV(media de volume por módulo)	A.35	Analisabilidade, Testabilidade	Complexidade

74	aveLOC (média de linhas de código por módulo)	A.35	Analisabilidade, Testabilidade	Tamanho
75	Subroutine averages	A.35		Tamanho
76	extended cyclomatic complexity	A.35	Analisabilidade, Testabilidade	Complexidade
77	NstLvlAvg (Average nesting level)	A.21	Modificabilidade, Estabilidade	Acoplamento
78	PIM ()	A.8	Modificabilidade, Estabilidade	Acoplamento
79	REUSE VALUE	A.5	Modificabilidade	Acoplamento
80	SIMAS -	A.8	Modificabilidade, Estabilidade	Acoplamento
81	EXTERNAL REUSE	A.5	Modificabilidade	Acoplamento
82	REUSE LEVERAGE	A.5	Modificabilidade	Acoplamento
83	NEW OBJECT PERCENT	A.5	Modificabilidade	Tamanho
84	Changing Method (CM)	A.20	Modificabilidade	Acoplamento
85	StmCtlNbr (Number of Control Statements)	A.21	Estabilidade, Testabilidade	Complexidade

86	Directly called components	A.29	Estabilidade	Acoplamento
87	PthIndNbr (Number of Independents Paths)	A.21	Estabilidade, Testabilidade	Complexidade
88	C3 (Conceptual Cohesion of Classes)	A.7	Modificabilidade	Coesão
89	v22	A.1	Analisabilidade	Coesão
90	v11	A.1	Analisabilidade	Complexidade
91	v10	A.1	Analisabilidade	Acoplamento
92	v6	A.1	Analisabilidade	Encapsulamento
93	PCCC (Path Connectivity Class Cohesion)	A.3	Modificabilidade	Coesão
94	Oln	A.3	Modificabilidade	Coesão
95	ICBMC (Improved Cohesion Based on Member Connectivity)	A.3	Modificabilidade	Coesão
96	CBMC (Cohesion Based on Member Connectivity)	A.3	Modificabilidade	Coesão
97	DC1	A.3	Modificabilidade	Coesão

98	DCd	A.3	Modificabilidade	Coesão
99	Coh	A.3	Modificabilidade	Coesão
100	SCOM (Class Cohesion Metric)	A.3	Modificabilidade	Coesão
101	Number of entry nodes	A.29	Estabilidade	Complexidade
102	LSCC (Low-level design Similarity-based Class Cohesion)	A.3	Modificabilidade	Coesão
103	Number of exit nodes	A.29	Estabilidade	Complexidade
104	MOA	A.36	Modificabilidade	Acoplamento
105	Type Rank	A.25		Coesão
106	method total reuse	A.33	Modificabilidade	Acoplamento
107	method same subclass reuse	A.33	Modificabilidade	Acoplamento
108	method same class reuse	A.33	Modificabilidade	Acoplamento
109	Count of Blank lines	A.2		Tamanho

110	(ABC) Association Between Class	A.25	Analisabilidade,Modificabilidade Estabilidade,Testabilidade	Acoplamento
111	PercentageCoverage	A.25	Testabilidade	Complexidade
112	Count of Keywords	A.2		Tamanho
113	Obsolete language constructs.	A.18		Complexidade
114	Average cyclomatic complexity per method	A.29	Analisabilidade, Testabilidade	Complexidade
115	NIS (Number of interface)	A.4	Analisabilidade, Testabilidade	Tamanho
116	NdsNbr (Number of Nodes)	A.21	Estabilidade	Tamanho
117	CC (Class Cohesion)	A.3	Modificabilidade	Coesão
118	ANU (Average Non Uniformity)	A.23	Modificabilidade, Estabilidade	Coesão
119	perCM	A.35	Analisabilidade	Complexidade
120	OAM (Operation Acess Metric)	A.6	Modificabilidade, Estabilidade	Encapsulamento
121	CndCplAvg - Average Complexity of	A.21	Analisabilidade, Testabilidade	Complexidade

	Decisions			
122	NOI (Number of Inline (Trivial) Methods)	A.6	Modificabilidade, Estabilidade	Complexidade
123	CalUnq - Unique Calls to Other Functions	A.21	Modificabilidade, Estabilidade	Complexidade
124	NAD (Number of Abstract Data Types)	A.6	Modificabilidade	Acoplamento
125	NRA (Number of Reference Attributes)	A.6	Analisabilidade, Testabilidade	Tamanho
126	CalNbr - Total Calls to other functions	A.21	Modificabilidade, Estabilidade	Complexidade
127	CSB (Class Size in Bytes)	A.6	Analisabilidade, Testabilidade	Tamanho
128	CEC (Class Entropy Complexity)	A.6	Analisabilidade, Testabilidade	Complexidade
129	Average of- Weak Uniformity (AWU)	A.23	Modificabilidade, Estabilidade	Coesão
130	Average of -Strong Uniformity (ASU)	A.23	Modificabilidade, Estabilidade	Coesão
131	MAA (Measure of Attribute Abstraction)	A.6	Modificabilidade	Acoplamento
132	VarLenAvg - Average variable name length	A.21	Analisabilidade	Complexidade
133	v1	A.1	Analisabilidade, Testabilidade	Tamanho

134	ComLogNbr - Number of non-blank lines	A.21	Analisabilidade, Testabilidade	Tamanho
135	ComStrVol - Volume of Control Comments	A.21	Analisabilidade	Complexidade
136	In-Degree (Number of Incoming Edges)	A.11	Modificabilidade, Estabilidade	Acoplamento
137	NOSF (Number of Static Fields)	A.11	Analisabilidade, Testabilidade	Encapsulamento
138	NOSM (Number of Static Methods)	A.33	Analisabilidade, Testabilidade	Encapsulamento
139	Out-Degree (Number of outgoing edges)	A.11	Modificabilidade, Estabilidade	Acoplamento
140	Specialization Index	A.11		Acoplamento
141	ntrn	A.11		Tamanho
142	NHD	A.15	Modificabilidade	Coesão
143	SNHD	A.15	Modificabilidade	Coesão
144	NHDM	A.15	Modificabilidade	Coesão
145	TODO Count	A.17	Analisabilidade	Complexidade
146	WMC (Average Weighted Methods Per Class)	A.17	Analisabilidade	Complexidade

147	TA (Type Affinity)	A.23	Modificabilidade, Estabilidade	Coesão
148	FMMEC	A.22	Modificabilidade, Estabilidade	Acoplamento
149	Essential complexity	A.28	Analisabilidade, Testabilidade	Complexidade
150	ComDecVol (Volume of declarations comments)	A.21	Analisabilidade	Complexidade
151	Program Length	A.28	Analisabilidade, Testabilidade	Tamanho
152	Program Vocabulary	A.28	Analisabilidade, Testabilidade	Complexidade
153	Difficulty	A.28	Analisabilidade, Testabilidade	Complexidade
154	IMNU (ImportNotUsed)	A.26		Complexidade
155	ICH (Information-flow based cohesion)	A.22	Modificabilidade	Coesão
156	COF (Coupling factor)	A.22	Modificabilidade, Estabilidade	Acoplamento
157	IFCAIC	A.22	Modificabilidade, Estabilidade	Acoplamento
158	NdsExtNbr (Number of Exits in a function)	A.21	Estabilidade, Testabilidade	Complexidade

159	IFCMIC	A.22	Modificabilidade, Estabilidade	Acoplamento
160	LopNbr (Number of Loops)	A.21	Analisabilidade, Testabilidade	Complexidade
161	StmExeNbr - Number of executable statements	A.21	Analisabilidade	Tamanho
162	IFMMIC	A.22	Modificabilidade, Estabilidade	Acoplamento
163	NLC	A.6	Estabilidade, Testabilidade	Acoplamento
164	DMMEC	A.22	Modificabilidade, Estabilidade	Acoplamento
165	OMMEC	A.22	Modificabilidade, Estabilidade	Acoplamento
166	FCMEC	A.22	Modificabilidade, Estabilidade	Acoplamento
167	CndSpnAvg (Average Decision Span)	A.21	Estabilidade, Testabilidade	Complexidade
168	OCAEC	A.22	Modificabilidade, Estabilidade	Acoplamento
169	FCAEC	A.22	Modificabilidade, Estabilidade	Acoplamento
170	Co	A.22	Modificabilidade	Coesão

171	NewCo	A.22	Modificabilidade	Coesão
172	NSI (Numbero of Single Inheterance)	A.6	Modificabilidade, Estabilidade	Acoplamento
173	NMI (Number of Multiple Inherited)	A.6	Modificabilidade, Estabilidade	Acoplamento
174	NNC	A.6	Modificabilidade, Estabilidade	Acoplamento
175	NAC	A.6	Modificabilidade	Acoplamento
176	ArcNbr (Number of Arcs)	A.21	Testabilidade	Complexidade
177	KntNbr (Number of Knots)	A.21	Testabilidade	Complexidade