UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS RUSSAS

CURSO DE GRADUAÇÃO EM ENGENHARIA DE SOFTWARE

FRANCISCO GUTENBERG DA SILVA FILHO

UM ESTUDO SOBRE ATRIBUTOS QUE INFLUENCIAM NA TESTABILIDADE DE SOFTWARE

RUSSAS

2019

FRANCISCO GUTENBERG DA SILVA FILHO

UM ESTUDO SOBRE ATRIBUTOS QUE INFLUENCIAM NA TESTABILIDADE DE SOFTWARE

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Software do Campus Russas da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Engenharia de Software.

Orientadora: Prof. Dra. Valéria Lelli Leitão Dantas

RUSSAS

2019

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

F498e Filho, Francisco Gutenberg da.
Um estudo sobre atributos que influenciam na testabilidade de software / Francisco Gutenberg da Filho. – 2019.
86 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Russas, Curso de Engenharia de Software, Russas, 2019.
Orientação: Prof. Dr. Valéria Lelli Leitão Dantas.

1. Testabilidade. 2. Teste de Software. 3. Qualidade de Software. I. Título.

CDD 005.1

FRANCISCO GUTENBERG DA SILVA FILHO

UM ESTUDO SOBRE ATRIBUTOS QUE INFLUENCIAM NA TESTABILIDADE DE SOFTWARE

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Software do Campus Russas da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Engenharia de Software.

Aprovada em:

BANCA EXAMINADORA

Prof. Dra. Valéria Lelli Leitão Dantas (Orientadora)
Universidade Federal do Ceará (UFC)

Profa. Dra. Anna Beatriz dos Santos Marques
Universidade Federal do Ceará (UFC)

Prof. Ms. José Osvaldo Mesquita Chaves
Universidade Federal do Ceará (UFC)

AGRADECIMENTOS

Primeiramente a Deus, por sempre me iluminar e me manter firme na realização dos meus sonhos.

A minha família, por todo o apoio, imensos esforços e educação que me deram, contribuindo com minha formação pessoal. Obrigado por confiarem em mim.

A minha namorada, por sempre ter me motivado e nunca deixar desistir. Te amo!

A minha orientadora, Profa. Dra Valéria Lelli Leitão Dantas, por ter me dado toda atenção, ensinamento e motivação durante esta pesquisa.

A todos os professores da UFC Campus Russas, em especial os professores de Engenharia de Software, por toda a dedicação dada dentro e fora de sala, com o objetivo de contribuir com a formação dos alunos.

A todos os amigos que fiz durante a graduação e que estiveram comigo nos momentos mais importantes deste ciclo.

Ao Doutorando em Engenharia Elétrica, Ednardo Moreira Rodrigues, e seu assistente, Alan Batista de Oliveira, aluno de graduação em Engenharia Elétrica, pela adequação do *template* utilizado neste trabalho para que o mesmo ficasse de acordo com as normas da biblioteca da Universidade Federal do Ceará (UFC).

“Seja a mudança que você quer ver no mundo.”

(Mahatma Gandhi)

RESUMO

A exigência por maior qualidade de software vem crescendo à medida que os sistemas estão mais presentes nas atividades do cotidiano, embora a ideia de qualidade de software seja intuitiva, ao ser analisada minuciosamente, esse conceito se torna complexo. Por ser complexo, muitos autores e normas técnicas propõem modelos, que facilitam a avaliação da qualidade de software. No geral, os modelos de qualidade (e.g., SQUARE 25000) definem um conjunto de características ou fatores que estão diretamente relacionados com a qualidade de software. Uma característica importante presente na maioria desses modelos é a testabilidade, pois ela possui relação com testes de software, que tem como objetivo evidenciar que um programa faz o que é proposto a fazer enquanto testabilidade pode ser definida como a facilidade com que um programa pode ser testado. Dessa forma, avaliar a testabilidade de um software traz informações relevantes sobre a qualidade de um sistema. Porém, a testabilidade não é uma característica intrínseca do software, e, portanto, deve se identificar ou definir medidas que deem uma indicação direta sobre testabilidade. A proposta deste estudo é investigar a testabilidade de software com o intuito de identificar quais atributos e respectivas métricas podem auxiliar na medição da testabilidade de um software orientado a objetos. Na primeira etapa do estudo foi feita uma revisão bibliográfica para identificar as características de testabilidade e as métricas que representem essas características. Com base nessa identificação, foi planejado e executado um experimento para medir a testabilidade em sistemas de código aberto. Como resultados, este trabalho apresenta um conjunto de métricas e atributos identificados na literatura, além de um relato da condução de um experimento para avaliar a testabilidade de software com base nestes atributos.

Palavras-chave: Testabilidade; Teste de Software; Qualidade de Software.

ABSTRACT

The demand for higher software quality has been growing as the systems are more present in everyday activities, although the idea of software quality is intuitive, when thoroughly analyzed, this concept becomes complex. Because it is complex, many authors and technical standards propose models that facilitate the evaluation of software quality. In general, quality models (e.g., SQUARE 25000) define a set of characteristics or factors that are directly related to software quality. An important characteristic present in most of these models is the testability, because it is related to software tests, which aims to show that a program does what it is proposed to do while testability can be defined as the ease with which a program can be tested. Thus, evaluating the testability of a software brings relevant information about the quality of a system. However, testability is not an intrinsic characteristic of software, and therefore, it should be identified or defined measures that give a direct indication about testability. The purpose of this study is to investigate the software testability in order to identify which attributes and respective metrics can help in the measurement of the testability of an object oriented software. In the first stage of the study, a literature review was performed to identify the characteristics of testability and the metrics that represent these characteristics. Based on this identification, an experiment was planned and executed to measure the testability in open source systems. As results, this work presents a set of metrics and attributes identified in the literature, in addition to a report on the conduct of an experiment to evaluate the software testability based on these attributes.

Keywords: Testability; Software Testing; Software Quality.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

AHF	Atributte Hidden Factor
CBC	Coupling Between Classes
CBO	Coupling Between Objects
COM	Cohesion Metric
CPM	Coupling Metric
DIT	Depth of Inheritance Tree
ENM	Encpsulation Metric
IFC	Information Flow Complexity
MHF	Method Hidden Factor
NOC	Number of Children
NOO	Number of Operations
REM	Reuse Metric
TAssert	Test Asserts
TCD-Complexity	Test Case Development-Complexity
TLOC	Test Line of Code
TNOO	Test Number of Operations
TRFC	Test Response for Class
TWMPC	Test Weighted Methods per Class

SUMÁRIO

1 INTRODUÇÃO

A exigência por maior qualidade de *software* vem crescendo à medida que os sistemas estão mais presentes nas atividades do cotidiano. Em 1990, muitas empresas perdiam bilhões de dólares pelo fato de não desenvolverem características e funcionalidades definidas inicialmente para o *software* (PRESSMAN, 2009). Além de perdas financeiras, outros pontos importantes instigam a necessidade de qualidade de *software*. Dayanandan e Vivekanandan (2016) considera a concorrência no mercado um outro fator relevante para que as empresas tenham um atenção especial com a qualidade de *software*.

A ideia de qualidade pode parecer intuitiva, porém, ao ser analisada minuciosamente, o conceito de qualidade se torna complexo (KOSCIANSKI; SOARES, 2007). Por ser um conceito complexo, muitos autores e normas técnicas propõem modelos, os quais são discutidos na Seção 2.1, que facilitam a avaliação da qualidade de *software*. No geral, os modelos de qualidade (e.g., SQUARE 25000) definem um conjunto de características ou fatores que podem afetar diretamente na qualidade do *software*. Entretanto, é impossível que um sistema atenda todas as características de um determinado modelo, uma vez que algumas características podem impactar negativamente em outras. Desta forma, a equipe de garantia de qualidade é responsável por definir quais são as características mais importantes para o projeto, com base na especificação do software (SOMMERVILLE, 2011).

Uma característica relevante que está presente na maioria dos modelos de qualidade é a testabilidade, que pode ser definida como a facilidade com que um programa pode ser testado. Essa característica possui uma intensa relação com testes de *software*, que é uma atividade fundamental no processo desenvolvimento de *software* que tem como objetivo evidenciar que um programa faz o que é proposto a fazer e para identificar os defeitos do programa antes mesmo do uso (SOMMERVILLE, 2011). Pressman (2009) evidencia essa relação ao afirmar que um engenheiro de *software* deve, ao projetar e implementar um sistema, levar em consideração a testabilidade, uma vez que o objetivo dos testes é encontrar erros. Dessa forma, avaliar a testabilidade de um *software* traz informações relevantes sobre a qualidade do sistema desenvolvido, porém pode ser um tanto o quanto desafiador, pelo fato da testabilidade não ser uma característica intrínseca do *software* (CRUZ, 2018). Portanto, é relevante identificar ou definir medidas que deem uma indicação direta sobre testabilidade.

Muitos trabalhos foram desenvolvidos ao longo dos anos acerca do assunto, entretanto, a maioria deles são específicos, restringindo-se à uma fase específica do processo de

desenvolvimento de *software*. Khan e Mustafa (2009), desenvolveram um modelo de testabilidade à nível de projeto (*design*) para projetos orientados a objetos, no qual era possível avaliar a testabilidade do *software* a partir do diagrama de classes. Por outro lado, Kout *et al.* (2011) basearam-se no modelo de Khan e Mustafa (2009) para criar um modelo de testabilidade também para projetos orientado objetos, entretanto, a nível de código. Khalid *et al.* (2010) utilizaram o modelo de Khan e Mustafa (2009) para analisar a testabilidade e a complexidade dos sistemas de *software* orientado a objetos. Freedman (1991) propõe medidas de testabilidade para componentes de *software* baseado em dois fatores: observabilidade e controlabilidade. Binder (1994) concorda com Freedman (1991) ao analisar a testabilidade baseada em observabilidade e controlabilidade, entretanto, Binder (1994) afirma que a testabilidade não pode ser considerada separadamente do processo de *software*.

A proposta deste trabalho é investigar a testabilidade de *software*, com o intuito de identificar um conjunto de medidas que facilite a avaliação dessa característica. O conjunto a ser desenvolvido deve identificar atributos de testabilidade, medidas que caracterizem esses atributos, bem como as fases de um processo de *software*, as quais uma medida pode ser avaliada.

A metodologia de trabalho utilizada nesta pesquisa foi dividida em duas. Na etapa 1, as atividades foram baseadas na revisão bibliográfica, onde foram identificados os atributos de testabilidade e métricas para avaliar esses atributos. Após a identificação do conjunto inicial de atributos, foram definidos dois critérios, um de inclusão e um de exclusão, que permitiu refinar a quantidade de atributos de testabilidade. Em seguida, na Etapa 2, foi conduzido um experimento para medir a testabilidade, por meio do conjunto de atributos e métricas identificados, em sistemas reais de código aberto, por fim, os dados foram coletados e analisados com o objetivo de analisar as correlações existentes entre atributos e entre métricas.

1.1 Escopo do Trabalho

O escopo deste projeto aborda a relevância de se investigar a testabilidade de *software*, uma vez que os atributos de testabilidade podem representar características importantes do processo de teste de *software* e auxiliar na avaliação da sua qualidade. Nesse contexto, este trabalho pretende responder às seguintes questões de pesquisa:

1. Quais são os atributos ou características de qualidade que podem avaliar a testabilidade de *software*?
2. Quais medidas ou métricas podem ser utilizadas para medir a testabilidade?

3. Como avaliar a testabilidade de um *software*?

1.2 Objetivos

O presente trabalho tem como objetivo evidenciar os atributos que afetam a testabilidade de um *software*, de modo a facilitar o entendimento à respeito da avaliação de testabilidade em projetos de *software*. Ao longo do desenvolvimento, a pesquisa visa alcançar os seguintes objetivos específicos:

- Identificar atributos de testabilidade
- Relacionar os atributos de testabilidade com características de teste de *software*;
- Identificar métricas para avaliar os atributos de testabilidade;
- Revelar aspectos do desenvolvimento de *software*, inerentes ao paradigma, que podem impactar na testabilidade do mesmo; e
- Realizar um experimento para avaliar a testabilidade de sistemas de *software* orientado a objetos.

1.3 Organização do Trabalho

O restante do trabalho está organizado da seguinte forma: no Capítulo 2, os principais conceitos desta pesquisa são definidos e explorados. No Capítulo 3 é realizada uma discussão sobre os principais trabalhos relacionados ao tema dessa pesquisa. A forma como este trabalho foi conduzido é apresentada no Capítulo 4 e as contribuições deste estudo são relatadas no Capítulo 5 e no Capítulo 6. Os Capítulos 7 e 8 apresenta, respectivamente, a forma como o experimento foi executado e as principais análises e discussões sobre os resultados do estudo. Por fim, o Capítulo 9 apresenta as conclusões e trabalhos futuros acerca desta pesquisa.

2 FUNDAMENTAÇÃO TEÓRICA

A utilização de testes, no contexto de desenvolvimento de *software* é essencial, pois através deles, os erros encontrados podem ser corrigidos, sempre como objetivo de obter um maior nível de qualidade. A execução das atividades de testes, no entanto, pode ser custosa e complexa, uma vez que nem todos os sistemas de *software* são fáceis de testar. Desta forma, é de fundamental importância compreender o conceito de testabilidade de *software*, pois ela preocupa-se em avaliar a capacidade de um dado sistema ser testado, explicitando uma relação direta com teste e qualidade de *software*.

Esta seção apresenta alguns conceitos importantes e fundamentais para a compreensão desta pesquisa. Nas Seções 2.1 e 2.2 são apresentados os conceitos relacionados à qualidade e testes de *software*.

2.1 Qualidade de *Software*

Segundo Pressman (2009), a definição de qualidade de *software* não é tão simples quanto parece, pode-se ter uma noção de qualidade ao vê-la, entretanto é difícil defini-la. A qualidade é resultado de um bom processo de gerenciamento de projeto complementado pela utilização de boas práticas de Engenharia de *Software*, dessa forma, o gerenciamento e utilização das boas práticas são aplicadas no contexto de 4 atividades que guiam uma equipe de *software* a atingir um alto padrão de qualidade, são elas:

- **Métodos de Engenharia de *Software*:** para construir *software*s de alta qualidade, é necessário que a equipe de desenvolvimento do projeto tenha um entendimento claro do problema a ser resolvido. A partir daí, uma solução pode ser concebida em forma de projeto de *software*. Alguns conceitos e métodos devem ser utilizados para auxiliar no entendimento do problema e na geração da solução, por exemplo: Engenharia de Requisitos, Projeto de Arquitetura, Projeto de Componentes, Projeto de Interfaces, dentre outros.
- **Técnicas de gerenciamento de *Software*:** decisões de projetos inadequadas, podem afetar negativamente a qualidade de *software*, por este motivo, deve-se estabelecer um plano de projeto contendo técnicas para gerenciamento de mudanças e qualidade. Exemplo de técnicas, são: Estimativas de Projeto, Gerência de Riscos, Cronograma de Projeto, dentre outros.

- **Controle de qualidade:** o controle de qualidade serve para garantir que um produto atinja suas metas de qualidade. Ela engloba um conjunto de ações de Engenharia de *Software* que auxiliam nessa garantia. Os modelos devem ser revisados de modo a garantir que estão completos e consistentes.
- **Garantia da qualidade:** define uma infraestrutura para apoiar os métodos de Engenharia de *Software*, fundamental para criar *softwares* de alta qualidade, além de compreender um conjunto de funções de auditorias e relatórios que permitem uma avaliação da eficiência e completude das ações de controle de qualidade.

Sommerville (2011) ao discutir sobre qualidade de *software*, afirma que o gerenciamento da qualidade para sistemas de *software*, tem três principais preocupações, a saber:

1. No nível organizacional, o objetivo do gerenciamento da qualidade é estabelecer um *framework* de processos organizacionais e padrões que possibilitem *software*s de alta qualidade. Portanto, a equipe responsável pelo gerenciamento da qualidade, deve ser capaz de definir quais os processos de desenvolvimento de *software* que serão utilizados, bem como os padrões que devem ser usados no *software*, além da documentação relacionada, incluindo os requisitos de sistema, projeto e código.
2. No nível de projeto, a preocupação do gerenciamento de qualidade está relacionado à aplicação de processos específicos de qualidade, garantindo que os processos sejam executados conforme planejado, além de garantir que as saídas de projeto estejam em conformidade com os padrões aplicáveis ao projeto.
3. Ainda no nível de projeto, o gerenciamento de qualidade também está preocupado com o estabelecimento de um plano de qualidade. O plano de qualidade deve definir as metas de qualidade para o projeto e quais processos e padrões devem ser usados.

Por ser um aspecto complexo, a qualidade de *software* não pode ser levada em consideração de forma indivisível, por este motivo, alguns autores e normas técnicas propõem modelos (e.g., SQUARE 25000) e conjuntos de características ou fatores que estão diretamente relacionados com à qualidade.

Dentre esses modelos de qualidade, podemos mencionar o sugerido por Garvin (1987). Nesse modelo a qualidade deve ser observada a partir de dimensões distintas, começando por uma avaliação da conformidade e concluindo com uma visão transcendental. Garvin (1987) define 8 dimensões de qualidade, porém, apesar de não ser definido especificamente para *software*, essas dimensões podem ser utilizadas quando se considera a qualidade de *software*. As

8 dimensões são:

- **Qualidade do desempenho:** "O *software* fornece todo o conteúdo, funções e recursos que são especificados como parte do modelo de requisitos de forma a gerar valor ao usuário final?"
- **Qualidade dos recursos:** "O *software* fornece recursos que surpreendem e encantam usuários finais que os utilizam pela primeira vez?"
- **Confiabilidade:** "O *software* fornece todos os recursos e capacidades sem falhas? Está disponível quando necessário ? Fornece funcionalidade sem a ocorrência de erros?"
- **Conformidade:** "O *software* está de acordo com os padrões de *software* locais e externos relacionados com à aplicação? Segue as convenções de projeto e codificação de fato?"
- **Durabilidade:** "O *software* pode ser mantido ou corrigido sem a geração involuntária de efeitos colaterais indesejados? As mudanças farão com que a taxa de erros ou a confiabilidade diminuam com o passar do tempo?"
- **Facilidade de Manutenção:** "O *software* pode ser mantido ou corrigido em um período de tempo aceitável e curto? O pessoal de suporte pode obter todas as informações necessárias para realizar alterações ou corrigir defeitos?"
- **Estética:** estética é uma dimensão mais subjetiva, além de estar relacionada com à abordagem baseada no usuário. Por ser subjetiva, a estética precisa de um julgamento pessoal, dessa forma, um cliente irá se interessar pelas combinações de atributos do produto que melhor correspondem às suas preferências.
- **Percepção:** assim como estética, esta dimensão também é subjetiva. Diz respeito à percepção que o cliente tem com relação à algum aspecto do desenvolvimento do produto. Por exemplo, se um *software* desenvolvido por um fornecedor que, no passado, desenvolveu um *software* de baixa qualidade, a percepção de qualidade do produto, por parte do cliente, será afetada negativamente.

Através das dimensões apresentadas, uma representação da qualidade de *software* pode ser explícita. Muitas dessas dimensões são consideradas apenas subjetivamente, assim, é necessário um conjunto de fatores de qualidade, que são classificados em duas categorias, fatores que podem ser medidos diretamente e fatores que só podem ser medidos de forma indireta (GARVIN, 1987).

Já em McCall *et al.* (1977), foi proposto uma classificação dos fatores que afetam diretamente a qualidade de *software*. Esses fatores, focalizam três aspectos importantes de um

produto de *software*: Revisão do Produto, Transição do Produto e Operação do Produto. Com relação aos fatores, McCall *et al.* (1977) definem da seguinte forma:

- **Correção:** "O quanto um programa satisfaz a sua especificação e atende aos objetivos definidos pelo cliente"
- **Confiabilidade:** "O quanto se pode esperar que um programa realize a função pretendida com a precisão exigida."
- **Eficiência:** "A quantidade de recursos computacionais e código exigidos por um programa para desempenhar sua função."
- **Integridade:** "O quanto o acesso ao *software* ou dados por pessoas não autorizadas pode ser controlado."
- **Usabilidade:** "Esforço necessário para aprender, operar, preparar a entrada de dados e interpretar a saída de um programa."
- **Facilidade de manutenção:** "Esforço necessário para localizar e corrigir um erro em um programa."
- **Flexibilidade:** "Esforço necessário para modificar um programa em operação."
- **Testabilidade:** "Esforço necessário para testar um programa de modo a garantir que ele desempenhe a função destinada."
- **Portabilidade:** "Esforço necessário para para transferir o programa de um ambiente de hardware ou *software* para outro."
- **Reusabilidade:** "O quanto um programa(ou partes) pode ser reutilizado em outras aplicações."
- **Interoperabilidade:** "Esforço necessário para integrar um sistema a outro."

Por outro lado, a SQuaRE ISO 25000 define um modelo de qualidade hierárquico que desmembra a qualidade de *software* em seis características, que por sua vez são decompostas em subcaracterísticas e, estas, em atributos. O modelo segue o que foi definido na norma técnica 9126, sua antecessora. As características definidas no modelo são:

- **Funcionalidade:** refere-se ao que o *software* realiza quando requisitado pelo usuário.
- **Manutenibilidade:** diz respeito a quão fácil um *software* pode ser modificado.
- **Usabilidade:** indica o quão fácil é o uso do sistema.
- **Confiabilidade:** capacidade de manter um nível de desempenho específico, quando executado em um determinado contexto de uso.
- **Eficiência:** refere-se ao uso dos recurso computacionais para a operação do *software*, em

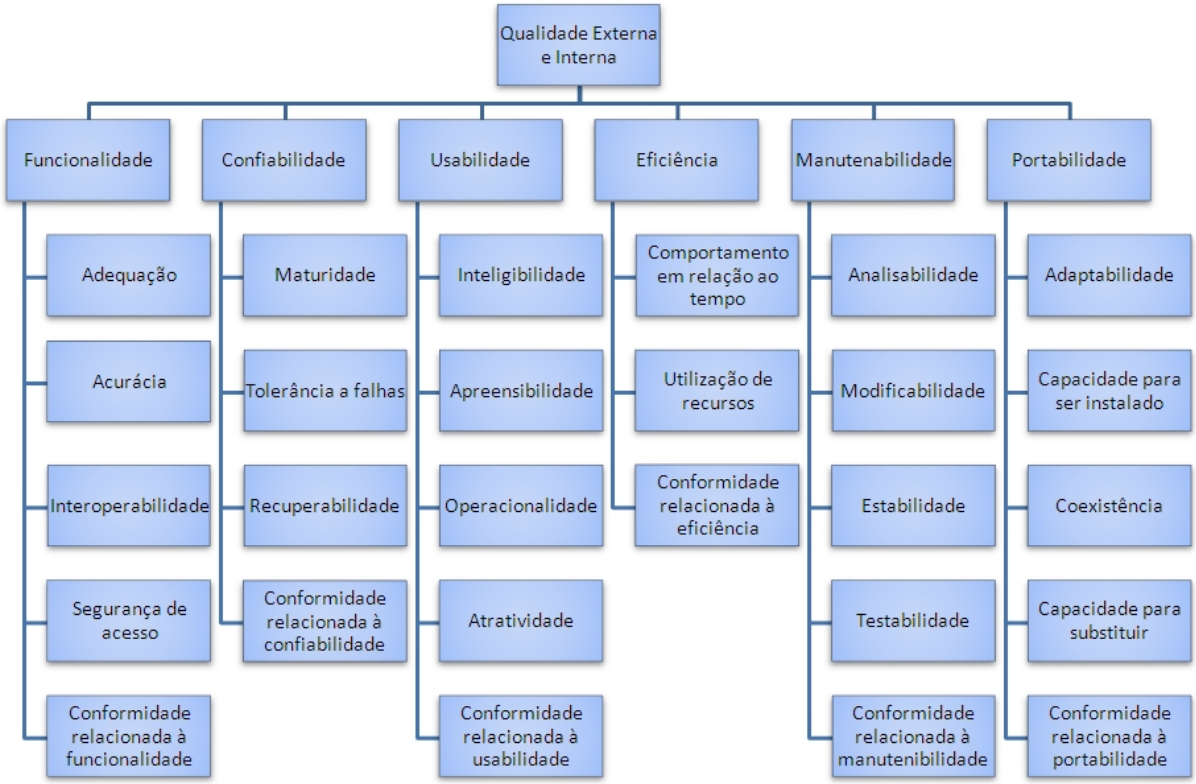
termos de tempo.

- **Portabilidade:** indica a possibilidade do sistema, ou subsistemas serem transferidos para outras plataformas de execução.

A Figura 1 ilustra as características e subcaracterísticas de qualidade conforme o modelo da SQuaRE ISO 25000. Através dela pode-se observar a estrutura hierárquica do modelo, indicando que testabilidade, característica investigada neste estudo, é uma sub-característica de Manutenibilidade, além de verificar que todas as características do modelo, possuem uma sub-característica em comum, a conformidade, pois ela pode conter atributos como padrões internos da organização ou imposição de legislação aplicáveis ao projeto ou sistema (KOSCIANSKI; SOARES, 2007).

Visto a importância da qualidade de software, é ideal que sejam desenvolvidas estratégias para a avaliação de qualidade de um sistema, no entanto, por ser complexo, essas estratégias devem ser específicas e bem definidas. Com isso, um dos resultados esperados para esta pesquisa é a identificação de atributos e métricas que facilitem a avaliação da característica testabilidade, apresentadas nos modelos acima.

Figura 1 – Modelo de Qualidade da IS0 25000



Fonte: TI (2016)

2.1.1 Métricas de Software

Pressman (2009) acredita que as métricas de *software* representam uma característica de um sistema, documentação de um sistema ou do processo de desenvolvimento, de tal forma que essa característica possa ser objetivamente medida. Pressman (2009) classifica as métricas em duas categorias: métricas de controle e métricas de previsão. As de controle apoiam processos de gerenciamento, enquanto que as métricas de previsão auxiliam a predizer características do *software*.

As métricas de *software* proporcionam uma medida para o *software* e o processo de produção de *software*, de forma a atribuir valores quantitativos aos atributos que envolvem o produto ou o processo (TU *et al.*, 2009). De acordo com a IEEE (1993) as métricas de *software* podem ser dadas como uma função, que recebe como entrada os dados de *software*, e tem como saída um valor que pode indicar como o atributo medido afeta o *software*. Assim como Pressman (2009), Tu *et al.* (2009) classifica as métricas em três tipos: métricas de procedimento, métricas de projeto e métricas de produto. As métricas de procedimento focam na forma como o *software* vai ser desenvolvido, ou seja, no processo de desenvolvimento e são utilizadas para obter o *status* da produção do *software*. Métricas de produto tem como objetivo entender a qualidade do produto, gerenciando a qualidade do mesmo a partir de características como confiabilidade, manutenção, escala do produto, complexidade do *software*, portabilidade, documentos, dentre outros. Por fim, as métricas de projeto são utilizadas para gerenciar e compreender o andamento do projeto, com o objetivo de auxiliar o projeto para evitar problemas e minimizar riscos, além de otimizar o planejamento.

Uma métrica de *software* pode ser conceituada como uma medida matemática objetiva de *software*, tal que essas medidas representem as diferentes características de um *software*. As métricas podem fornecer uma medida quantitativa de um atributo pertencente ao *software*. Elas são importantes pois as medidas numéricas do *software* podem ser transformadas em indicadores, tais como "confiabilidade"e "capacidade de manutenção", auxiliando no desenvolvimento e garantia da qualidade de *software* (GAFFNEY, 1981).

2.2 Teste de *Software*

O teste é destinado a mostrar que um programa faz o que é proposto a fazer e para descobrir os defeitos do programa antes do seu uso. Os resultados do teste também podem ser

verificados com o intuito de identificar erros, anomalias ou informações sobre os atributos não funcionais do programa (SOMMERVILLE, 2011). O autor afirma ainda, que, o processo de teste tem dois objetivos:

1. Demonstrar ao desenvolvedor e ao cliente que o *software* atende aos seus requisitos. Para *softwares* customizados, isso significa que deve haver pelo menos um teste para cada requisito do documento de requisitos. Para *softwares* genéricos, isso significa que deve haver testes para todas as características do sistema, além de suas combinações, que serão incorporadas ao *release* do produto.
2. Descobrir situações em que o *software* se comporta de maneira incorreta, indesejável ou de forma diferente das especificações. Essas são consequências de defeitos de *software*. O teste de defeitos preocupa-se com a eliminação de comportamentos indesejáveis do sistema, tais como panes, interações indesejáveis com outros sistemas, processamentos incorretos e corrupção de dados.

Para Koscianski e Soares (2007), o objetivo do teste é encontrar defeitos, de forma a expor que o comportamento do *software*, em algumas situações, pode não estar em conformidade. Por este motivo, para que um teste seja bem-sucedido, ele deve encontrar erros que estão contido no *software*, para que a equipe de desenvolvimento possa corrigir.

Leach (2016) diz que o teste de *software* não é para provar que o *software* está correto, ou seja, atende às especificações, e sim para identificar defeitos do *software*. O autor diz ainda que, isso se deve aos vários caminhos de execução possíveis de um *software*, as combinações possíveis de argumentos de funções e a complexidade de modo geral, relacionado as várias instruções do programa.

Em Myers (1979), são definidas algumas regras, que podem servir como objetivos da atividade de testes:

- Teste consiste em um processo de executar um programa com o intuito de encontrar um erro;
- Um bom pacote de testes é aquele em que há uma alta probabilidade de encontrar um erro ainda não descoberto; e
- Um teste bem-sucedido é aquele que revela um novo erro.

De forma semelhante, Davis (1995) define um conjunto de princípios de testes:

- **Principio 1:** "Todos os testes devem estar alinhados com os requisitos do cliente."
- **Principio 2:** "Os testes devem ser planejados muito antes de ser iniciados."

- **Principio 3:** "O princípio de Pareto se aplica a testes de *software*."
- **Principio 4:** Os testes devem começar "em particular"e progredir rumo ao teste "em geral".
- **Principio 5:** "Testes exaustivos são impossíveis."

2.2.1 Testabilidade

Para McCall *et al.* (1977), a testabilidade de *software* pode ser definida em termos de esforço requerido para testar um determinado *software*, de modo a garantir que ele execute a função pretendida. McCall *et al.* (1977) definem alguns atributos que podem influenciar na testabilidade, são eles: Simplicidade, Modularidade, Instrumentação e Auto-Descritividade. Independentemente do objetivo, a testabilidade é importante não só para desenvolvedores *ad hoc*, mas também para organizações com alto nível de maturidade, pois pode reduzir o custo de um processo. Binder (1994) afirma que a testabilidade é consequência de seis fatores que podem facilitar ou dificultar os testes, são eles:

- **Representação:** é necessária uma representação para o desenvolvimento dos casos de teste.
- **Implementação:** características de implementação indicam aspectos de controle e observação.
- **Construção de testes:** a capacidade de construir teste determina a observação e desacopla a capacidade de teste das características da aplicação.
- **Coleção de testes:** uma grande coleção de casos de teste é necessária para aumentar a cobertura dos testes.
- **Ferramentas de testes:** são necessárias para diminuir o tempo de realização dos testes.
- **Processo de teste:** uma abordagem organizada e bem definida pode impactar diretamente no sucesso dos testes.

No entanto, as principais características de testabilidade proposta por (BINDER, 1994; FREEDMAN, 1991), são observabilidade e controlabilidade, uma vez que, para testar componentes é necessário controlar a sua entrada e posteriormente observar a saída.

Por outro lado, Bach (1994) afirma que a testabilidade é simplesmente a facilidade com que o *software* pode ser testado, além disso, ele afirma que as características que podem indicar o nível de testabilidade de um *software*, são "Operabilidade", "Observabilidade", "Controlabilidade", "Decomponibilidade", "Simplicidade", "Estabilidade"e "Compreensibilidade". Gao

e Shih (2005) ao abordar a testabilidade no contexto de *software* baseado em componentes, afirmam que a testabilidade pode ser definida como um indicador de qualidade mensurável que pode ser utilizado para avaliar o esforço dos testadores para atingirem um objetivo de teste específico dentro do processo de teste, como um critério de cobertura de teste e a facilidade de realizar operações de teste em um determinado teste. Os autores especificam cinco fatores que podem afetar a testabilidade, sendo eles: "Compreensibilidade", "Observabilidade", "Controlabilidade", "Rastreabilidade"e "Capacidade de suporte de teste".

Embora muitos autores definam atributos ou características que permitem avaliar a testabilidade de software, pode-se observar que os atributos mais utilizados pelos autores citados foram: controlabilidade, observabilidade, simplicidade e compreensibilidade.

3 TRABALHOS RELACIONADOS

Neste capítulo são apresentados os trabalhos que motivaram e inspiraram o desenvolvimento do estudo atual. A seguir é discutido, para cada trabalho, aspectos como objetivos, metodologia e resultados desses trabalhos. Para finalizar o capítulo, são elencadas as principais lacunas dos trabalhos apresentados e os diferenciais deste trabalho.

No trabalho de Khan e Mustafa (2009) é realizado uma ampla revisão sobre testabilidade de *software* no paradigma orientado a objetos, apontando informações importantes sobre testabilidade no nível de *design*. O objetivo do estudo é conjecturar a testabilidade de *software*, utilizando o diagrama de classes, no início do ciclo de vida, a partir de um modelo proposto pelos autores, intitulado *Metric Based Testability Model for Object Oriented design* (MTMOOD). Para a elaboração do modelo, foram identificadas as principais propriedades dos *software*s orientados a objetos, que tinham influencia na testabilidade e logo após foram identificadas métricas para essas propiedades, a nivel de *design*. Como resultado do estudo, as seguintes propriedades foram identificadas: encapsulamento, herança, acoplamento e coesão. Para essas propriedades, foram identificadas, respectivamente, as métricas ENM (encapsulamento), REM (de herança de reutilização), CPM (Acoplamento) e COM(Coesão). Com isso, pode se observar que a testabilidade global estimada pelo modelo proposto indicou uma correlação estatisticamente significante, demonstrando que havia relação entre as métricas e a testabilidade.

Khalid *et al.* (2010) também analisam o nível de testabilidade e complexidade de sistemas orientado a objetos, na fase de *design*. Os autores definiram métricas baseando-se nos modelos de Khan e Mustafa (2009) e Chidamber e Kemerer (1994), criando um modelo híbrido para analisar a relação entre complexidade, testabilidade e propriedades do *design* de *software* orientado a objeto com o objetivo de prever a capacidade de teste no nível de projeto. As métricas propostas foram mapeadas e, em seguida foram calculados seus respectivos valores em quatro projetos de *software*, para cada classe desses projetos. Dentre os resultados do estudo, um relevante foi o fato de que o acoplamento de sistemas orientados a objetos tem uma relação diretamente proporcional com a complexidade e testabilidade de *software*.

Kout *et al.* (2011) apresentam um modelo de testabilidade baseado em métricas para programas orientados a objetos, também fundamentado pelo trabalho de Khan e Mustafa (2009). Os autores adaptaram o modelo de Khan e Mustafa (2009) para prever o nível de teste (em termos de esforço de teste) de classes de sistemas OO na fase de implementação. No estudo, a testabilidade foi investigada no contexto de testes de unidade, foram selecionados dois sistemas

desenvolvidos em Java que possuíam casos de teste JUnit[1]. Os autores utilizaram métricas para quantificar os casos de teste correspondentes da classe JUnit com o intuito de mensurar a testabilidade das classes em termos do esforço de teste necessário, e, para validar o modelo proposto, foi utilizado testes estatísticos usando correlação. O estudo permitiu verificar que há uma relação estatisticamente significativa entre o modelo e as métricas de caso de teste utilizadas, além de que, as medidas tiveram correlação positiva, indicando que os *ranks* da variáveldo modelo, que são coeficiente calculados a partir do coeficiente de Spearman (1904), aumentam à medida que os *ranks* das métricas de caso de teste aumentam.

Tsui e Iriele (2011) fazem uma investigação acerca da relação entre coesão, que é uma propriedade de sistemas orientado a objetos, e a complexidade de desenvolvimento de casos de teste, que pode ser considerado como um indicador de uma característica de testabilidade. No estudo foi realizada uma medição em um pequeno *software* OO, utilizando duas métricas que representam coesão, LCOM5 e ITRA-C, e uma métrica que representa a complexidade de desenvolvimento de casos de teste, TCD-Complexity, com objetivo de identificar relações entre o atributo de coesão e o atributo de complexidade do desenvolvimento de casos de teste, de modo a verificar se de fato a coesão de um *software* impacta na complexidade dos testes. As medidas foram correlacionadas através do coeficiente de correlação de Spearman e, com isso, foi observado que LCOM5 teve uma baixa correlação com a TCD-Complexity, indicando que a falta de coesão não influenciou na complexidade de desenvolvimento de casos de teste enquanto que ITRA-C correlacionou-se bem com a TCD-Complexity, representando que quando uma métricas é alterada, a outra também é de forma proporcional. Além disso, pode-se verificar que o ITRA-C, em classes e métodos grandes, pode servir como um indicador significativo para avaliar a complexidade do desenvolvimento de casos de teste.

A principal diferença entre este trabalho e os demais, citado acima, é o fato de que a testabilidade será investigada baseado em artefatos de duas fases do clico de vida distintas (*design* e implementação), além de avaliar a testabilidade a partir de atributos propostos na literatura, em uma abordagem *top-down*. Além disso, este trabalho não será fundamentado em um modelo de testabilidade específico, será realizado uma revisão bibliográfica para identificar os atributos e respectivas métricas de testabilidade com o propósito de selecionar alguns atributos e algumas métricas para avaliar a testabilidade em sistemas de *software* através de um experimento controlado.

[1] Disponível em: <https://junit.org/junit5/>

A Tabela 1 apresenta as principais métricas utilizadas no contexto de testabilidade pelos trabalhos supracitados, em seguida, é apresentada uma legenda descrevendo as métricas apresentadas e logo após é feita uma rápida discussão sobre elas, de modo a identificar as lacunas nos trabalhos apresentados.

Tabela 1 – Métricas identificadas de acordo com o trabalho relacionado

MÉTRICA	**TRABALHO**
Encpsulation Metric (ENM)	Khan e Mustafa (2009)
Reuse Metric (REM)	Khan e Mustafa (2009)
Coupling Metric (CPM)	Khan e Mustafa (2009)
Cohesion Metric (COM)	Khan e Mustafa (2009)
Atributte Hidden Factor (AHF)	Khalid *et al.* (2010)
Method Hidden Factor (MHF)	Khalid *et al.* (2010)
Depth of Inheritance Tree (DIT)	Khalid *et al.* (2010);Kout *et al.* (2011)
Number of Children (NOC)	Khalid *et al.* (2010)
Coupling Between Classes (CBC)	Khalid *et al.* (2010)
Test Case Development-Complexity (TCD-Complexity)	Tsui e Iriele (2011)
Number of Operations (NOO)	Kout *et al.* (2011)
Coupling Between Objects (CBO)	Kout *et al.* (2011)
Test Line of Code (TLOC)	Kout *et al.* (2011)
Test Asserts (TAssert)	Kout *et al.* (2011)
Test Number of Operations (TNOO)	Kout *et al.* (2011)
Test Response for Class (TRFC)	Kout *et al.* (2011)
Test Weighted Methods per Class (TWMPC)	Kout *et al.* (2011)

Descrição:

- **ENM:** contabiliza todos os métodos definidos em uma determinada classe.
- **REM:** contabiliza a quantidade de classes hierárquicas.
- **CPM:** contabiliza a quantidade de classes distintas as quais uma determinada classe está diretamente relacionada.
- **COM:** calcula o relacionamento entre os métodos de uma determinada classe, baseada nos parâmetros.
- **AHF:** total de atributos ocultos(privados ou protegidos)em relação ao número total de atributos de uma classe.
- **MHF:** total de métodos ocultos(privados ou protegidos)em relação ao número total de métodos de uma classe.
- **DIT:** determina o número de níveis ao qual uma determinada classe herda.
- **NOC:** determina o número de filhos de uma classe.
- **CBC:** contabiliza a quantidade de classes às quais uma determinada classe está diretamente

relacionada.

- **TCD-Complexity:** representa o esforço de desenvolvimento de casos de testes.
- **NOO:** indica o número de métodos de uma classe.
- **CBO:** determina o acoplamento entre objetos de uma classe.
- **TLOC:** representa o total de linhas de códigos de uma determinada classe de teste.
- **TASSERT:** indica a quantidade de chamadas aos métodos de declaração Assert da JUnit que ocorrem no código de uma classe de teste.
- **TNOO:** indica o número de métodos de uma classe de teste.
- **TRFC:** define o conjunto de respostas definidas para uma classe de teste.
- **TWMPC:** indica o total das complexidades dos métodos de uma classe de teste, em que cada método é ponderado pela sua complexidade ciclomática.

Conforme pode-se observar na 1, grande parte das métricas são utilizadas em paradigmas Orientado a Objetos, como por exemplo, ENM,DIT E CBO, isso porque os autores ao definirem métricas para testabilidade, partem de propriedades inerentes ao paradigma de programação. Kout *et al.* (2011) são os únicos que vão mais além e utilizam algumas métricas relacionadas às classes de teste, como TLOC e TASSERT, para caracterizar a testabilidade. A ideia deste estudo é partir dos atributos identificados, para então identificar métricas que caracterizem esses atributos, levando em consideração fases do desenvolvimento de software.

4 METODOLOGIA

Para alcançar os objetivos definidos, a metodologia de pesquisa deste estudo é dividida em duas etapas. Na primeira etapa, foi realizada uma revisão bibliográfica sobre testabilidade, com o propósito de identificar os atributos e métricas de testabilidade. Este conjunto de atributos e métricas foram posteriormente refinados, durante a atividade "avaliação da testabilidade", utilizando dois critérios, um de inclusão e um de exclusão, a saber:

- Critério de inclusão: Deve ser selecionados aqueles atributos que foram mais utilizados pelos autores, com base no levantamento bibliográfico
- Critério de exclusão: Deve ser excluídos aqueles atributos que não podem ser medidos com base nos artefatos disponíveis no projeto.

Estes critérios foram utilizados para definir um conjunto consistente que permite a avaliação de testabilidade de *softwares* orientado a objetos, uma vez que, se for adotado um número elevado de atributos, consequentemente o número de métricas seria grande, o que poderia tornar a avaliação complexa ou até mesmo inconsistente. Portanto, o conjunto final de atributos abrangeu aqueles atributos que eram mais utilizados e que possuíam artefatos disponíveis para que pudesse ser realizado a avaliação destes atributos.

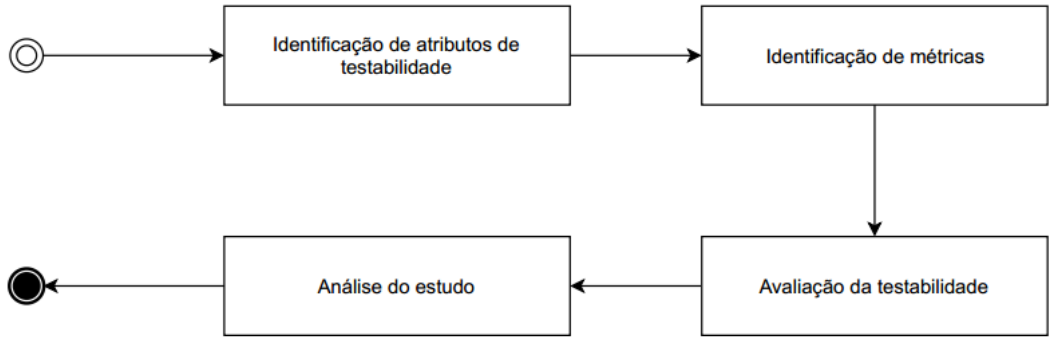
A Tabela 2 apresenta a *string* de busca utilizada para levantar os trabalhos relacionados e a quantidade de artigos retornados por base de dados e, posteriormente selecionados.

Tabela 2 – Resultados da revisão bibliográfica

BASE	STRING	RETORNADOS	SELECIONADOS
ACM	1 - testability AND (metric or measure) AND (attributes OR characteristics) AND (software OR system)	28	4
ACM	2 - testability AND (attributes OR characteristics) AND (software OR system)	41	2
IEEE	1 - testability AND (metric or measure) AND (attributes OR characteristics) AND (software OR system)	26	2
IEEE	2 - testability AND (attributes OR characteristics) AND (software OR system)	260	7

Na Etapa 2, foi realizado um experimento para avaliar a testabilidade de sistemas reais, baseado nos atributos de testabilidade identificados na Etapa 1. A Figura 2 ilustra as principais atividades da metodologia proposta e, paralelo à ela são descritas as atividades ilustradas.

Figura 2 – Metodologia de pesquisa



- **Etapa 1**
  - **Identificação de atributos de testabilidade:** nessa atividade, os atributos de testabilidade foram identificadas, com base na revisão bibliográfica, de forma a produzir um conjunto inicial de atributos.
  - **Identificação de métricas:** ainda baseado na revisão bibliográfica, essa atividade buscou identificar métricas que avaliassem a testabilidade de software, para que posteriormente pudesse ser realizada uma associação entre métricas e atributos.
- **Etapa 2**
  - **Avaliação da testabilidade:** após a elaboração do conjunto de atributos e métricas, foi realizado um experimento em sistemas de código aberto seguindo o conjunto, para avaliar a testabilidade desses sistemas. Essa atividade teve sub-atividades como planejamento, onde foi definido os sistemas a serem avaliados, bem como ferramentas que foram utilizadas, execução, que seria de fato a coleta das métricas e analise, que definem os métodos como foram analisados os dados coletados. Todas estas sub-atividades foram definidas no planejamento experimental[2] deste estudo, que está disponível no Apêndice B.
  - **Análise do estudo:** nessa atividade, após coletados e analisado os dados referentes a atividade anterior, foi realizada uma análise do estudo, onde foram identificados os pontos fortes e pontos de melhoria desta pesquisa.

[2] Disponível em: <https://drive.google.com/drive/folders/1OrsyBF6jVfRFlrtor7So6Bm-xQ3GlLBB?usp=sharing>

5 IDENTIFICAÇÃO DOS ATRIBUTOS E MÉTRICAS

O levantamento bibliográfico foi baseado nas strings de buscas apresentadas na Seção 4, e a partir dele foi realizado um estudo teórico sobre os principais estudos selecionados desta revisão. O objetivo deste estudo teórico foi identificar aqueles atributos e métricas que os autores utilizavam para avaliar a testabilidade de software. Como resultados deste estudo, foi identificado um total de 12 atributos e 30 métricas. O Quadro 1 apresenta os atributos identificados, que logo em seguida são definidos e os respectivos estudos associados.

Quadro 1 – Atributos de testabilidade identificados

ATRIBUTOS	Autores				
	McCall *et al.* (1977)	Bach (1994)	Binder (1994)	Freedman (1991)	Gao e Shih (2005)
Simplicidade	X	X			
Modularidade	X				
Observabilidade		X	X	X	X
Decomponibilidade		X			
Estabilidade		X			
Compreensibilidade		X			X
Instrumentação	X				
Controlabilidade		X	X	X	X
Operabilidade		X			
Auto-Descritividade	X				
Capacidade de suporte de teste					X
Rastreabilidade					X

- **Simplicidade:** para que um *software* seja simples, a estrutura do programa deve ser simples e objetiva, evitando, sempre que possível, o uso de estruturas que introduzam complexidade no código (BACH, 1994). McCall *et al.* (1977) abordam esse atributo em seu trabalho, definindo como o nível com que o código implementado, pode ser entendido por outra pessoa.
- **Modularidade:** estrutura de módulos com nível alto de independência (MCCALL *et al.*, 1977).
- **Instrumentação:** capacidade de auxiliar a identificação de erros (MCCALL *et al.*, 1977).
- **Auto-Descritividade:** capacidade de fornecer uma interpretação sobre a implementação de uma função (MCCALL *et al.*, 1977).
- **Controlabilidade:** de acordo com Freedman (1991), controlabilidade é a facilidade com

que um programa tem de gerar uma saída específica, a partir de uma entrada específica, de tal forma que, se um módulo não é controlável, então sua execução pode mostrar inconsistências de entrada, de forma similar, Binder (1994) afirma que um componente independente podem ser mais facilmente controlável. Controlabilidade também pode ser dada como o grau em que um componente é projetado para simplificar o controle de suas execuções durante a validação (GAO; SHIH, 2005). Além disso, para um *software* ser controlável, os estados e variáveis do *software* devem ser visíveis, podendo ser consultados durante sua execução (BACH, 1994).

- **Observabilidade:** está relacionada ao grau em que os componentes são projetados para simplificar o monitoramento e a observação das funções dos componentes e dos comportamentos dos testes de componentes (GAO; SHIH, 2005). Binder (1994) e Freedman (1991) acreditam que a observabilidade é a facilidade de determinar se as entradas específicadas em um programa afetam as saídas do mesmo. Para Bach (1994), todas as saídas possíveis de um programa podem ser geradas através de uma combinação de entradas, seguindo um formato consistente e estruturados.
- **Operabilidade:** "Quanto melhor funcionar, mais eficientemente pode ser testado."Sistemas que são implementados sem ter como objetivo garantir a qualidade, provavelmente haverá muitos defeitos (BACH, 1994).
- **Decomponibilidade:** "Controlando o escopo do teste, podemos isolar problemas mais rapidamente e executar um reteste mais racionalmente (BACH, 1994)."
- **Estabilidade:** "Quanto menos alterações, menos interrupções nos testes (BACH, 1994)."
- **Compreensibilidade:** "Quanto mais informações tivermos, mais inteligente será o teste"(BACH, 1994). Diz respeito ao grau com que um componente é específicado e projetado, de modo a facilitar o entendimento dos *stakeholders*, com o objetivo de facilitar a definição dos testes (GAO; SHIH, 2005).
- **Rastreabilidade:** Diz respeito ao aspecto que indica quão bem um componente é desenvolvido para auxiliar o monitoramento de diferente fluxos comportamentais de programas (GAO; SHIH, 2005).
- **Capacidade de suporte de teste:** É a capacidade de apoio ao processo de teste durante a validação do componente (GAO; SHIH, 2005).

Ao analisar os trabalhos que utilizam os atributos supracitados, pode-se perceber que nem todos utilizaram métricas específicas para avaliar o atributo, no entanto, outros trabalhos

propuseram o uso de métricas para avaliar a testabilidade, mas sem levar em consideração os atributos de testabilidade.

Este trabalho, por sua vez, teve como propósito identificar atributos mais utilizados para avaliar a testabilidade e com isso identificar métricas que estavam sendo utilizadas para avaliar a testabilidade, e relacioná-las com os atributos, para que estes atributos pudessem ser avaliados e posteriormente fosse feita a avaliação da testabilidade em sistemas reais. Após a identificação dos atributos, foram identificadas 30 métricas que podem ser utilizadas para avaliar a testabilidade, as quais são mostradas na Tabela 3.

Tabela 3 – Métricas identificadas

MÉTRICA	**Estudo**
ENM	Khan e Mustafa (2009)
REM	Khan e Mustafa (2009)
CPM	Khan e Mustafa (2009)
COM	Khan e Mustafa (2009)
AHF	Khalid *et al.* (2010)
MHF	Khalid *et al.* (2010)
DIT	Chidamber e Kemerer (1994)
NOC	Chidamber e Kemerer (1994)
CBC	Khalid *et al.* (2010)
TCD-Complexity	Tsui e Iriele (2011)
NOO	Chidamber e Kemerer (1994)
CBO	Kout *et al.* (2011)
TLOC	Kout *et al.* (2011)
TAssert	Kout *et al.* (2011)
TNOO	Kout *et al.* (2011)
TRFC	Kout *et al.* (2011)
TWMPC	Kout *et al.* (2011)
EC	Jureczko e Spinellis (2010)
AC	Jureczko e Spinellis (2010)
LCOM	Chidamber e Kemerer (1994)
NAK	Kudrjavets *et al.* (2006)
Information Flow Complexity (IFC)	Henry e Kafura (1981)
NTM	Nuñez-Varela *et al.* (2017)
WMC	Chidamber e Kemerer (1994)
NI	Nuñez-Varela *et al.* (2017)
IFC	Henry e Kafura (1981)
Unambiguous	Davis *et al.* (1993)
Correct	Davis *et al.* (1993)
Understandable	Davis *et al.* (1993)
Not Redundant	Davis *et al.* (1993)

6 PLANEJAMENTO DO EXPERIMENTO

Após serem identificados os atributos e suas respectivas métricas conforme apresentado no Capítulo 5, foi realizado um experimento para avaliar a testabilidade de três projetos orientado à objetos código aberto. Para a realização deste experimento, foram definidos os atributos que seriam avaliados, bem como as métricas que avaliaram esses atributos, os sistemas, e a forma como os dados foram analisados, que são discutidos nas próximas seções.

6.1 Seleção do Contexto

Este experimento foi realizado de maneira controlada, com objetivo de avaliar a testabilidade de software por meio dos atributos identificados. Foram selecionados três sistemas de código aberto, e para cada sistema foram coletados dados referente as métricas, que por sua vez determinam cada atributo. A realização deste experimento foi conduzido de maneira automática, utilizando ferramentas que automatizam, principalmente, os procedimentos de coletas e análise de dados. Como resultados do experimento, foram identificadas correlações entre atributos e entre métricas definidas para o experimento.

6.1.1 Atributos Selecionados

Quadro 2 – Atributos de testabilidade pré-selecionados

ATRIBUTOS	Autores					
	McCall *et al.* (1977)	Bach (1994)	Binder (1994)	Freedman (1991)	Gao e Shih (2005)	Total
Simplicidade	X	X				2
Modularidade	X					1
Observabilidade		X	X	X	X	4
Decomponibilidade		X				1
Estabilidade		X				1
Compreensibilidade		X			X	2
Instrumentação	X					1
Controlabilidade		X	X	X	X	4
Operabilidade		X				1
Auto-Descritividade	X					1
Capacidade de suporte de teste					X	1
Rastreabilidade					X	1

Conforme ilustrado no Quadro 2 e no critério de inclusão definido no Capítulo 4, pode-se observar que os atributos mais utilizados pelos estudos foram: controlabilidade e observabilidade, citados por quatro autores distintos, seguidos por simplicidade e compreensibilidade, citados por dois autores distintos. Com base nestes 4 atributos pré-definidos pelo critério de inclusão, foi aplicado o critério de exclusão em cima destes atributos, fazendo com que o atributo compreensibilidade fosse excluído do conjunto, ficando o grupo de atributos definidos por Simplicidade, Observabilidade e Controlabilidade, uma vez que estes são os atributos que podem ser coletados a partir dos artefatos disponíveis.

6.1.2 Métricas Selecionadas

Após a seleção dos atributos, foram selecionadas as métricas que representassem estes atributos. Assim como o conjunto de atributos, algumas métricas foram excluídas do conjunto inicial, desta vez, por algumas apresentarem interpretações ou cálculos similares. Com isso, o conjunto final de métricas utilizadas neste experimento são as apresentadas na Tabela 4.

Tabela 4 – Métricas selecionadas por atributo

ATRIBUTO	**MÉTRICA**	**ESTUDO**	**FASE**
Simplicidade	ENM	Khan e Mustafa (2009)	*Design*
Simplicidade	REM	Khan e Mustafa (2009)	*Design*
Simplicidade	AHF	Khalid *et al.* (2010)	*Design*
Simplicidade	MHF	Khalid *et al.* (2010)	*Design*
Simplicidade	NOC	Chidamber e Kemerer (1994)	Ambos
Simplicidade	DIT	Chidamber e Kemerer (1994)	Ambos
Simplicidade	NOO	Chidamber e Kemerer (1994)	Implementação
Simplicidade	WMC	Chidamber e Kemerer (1994)	Implementação
Simplicidade	EC	Jureczko e Spinellis (2010)	Implementação
Simplicidade	AC	Jureczko e Spinellis (2010)	Implementação
Simplicidade	LCOM	Chidamber e Kemerer (1994)	Implementação
Controlabilidade	CC	McCabe (1976)	Implementação
Controlabilidade	NI	Nuñez-Varela *et al.* (2017)	Implementação
Observabilidade	IFC	Henry e Kafura (1981)	Implementação

Como as métricas apresentadas na Tabela 4 foram utilizadas para avaliar os atributos,

e estes individualmente não podem dar uma indicação sobre a testabilidade do sistema, por se tratarem de métricas relacionadas as fases de *design* e implementação, um outro conjunto de métricas foi definido, desta vez, para avaliar a testabilidade. Com os dois conjuntos definidos: (1) métricas que avaliam os atributos identificados para testabilidade; e (2) as métricas que tem por objetivo principal avaliar a testabilidade, as métricas puderam ser coletadas e a partir dos resultados foi feita uma análise da relação entre as métricas desses dois conjuntos. Para fazer parte do conjunto de métricas que avaliam a testabilidade, foram selecionadas aquelas métricas que representam aspectos da atividade de testes de software, que são apresentadas na Tabela 5.

Tabela 5 – Métricas selecionadas para testabilidade

CARACTERÍSTICA	MÉTRICA	ESTUDO	FASE
Testabilidade	TLOC	Kout *et al.* (2011)	Teste
Testabilidade	TNOO	Kout *et al.* (2011)	Teste
Testabilidade	TWMC	Kout *et al.* (2011)	Teste
Testabilidade	NAK	Kudrjavets *et al.* (2006)	Teste
Testabilidade	NTM	Nuñez-Varela *et al.* (2017)	Teste

A seguir são detalhadas as métricas apresentadas na Tabela 4, que foram utilizadas no experimento, de acordo com os atributos associados e, em seguida, são descritas por atributo.

- **Simplicidade** Para obter medidas relacionadas ao atributo simplicidade dos projetos, foram analisadas as características particulares do paradigma, como encapsulamento, coesão e acoplamento, além disso, estas características foram medidas a partir do diagrama de classes, representando a fase de *design*, e também a partir do código fonte da aplicação, representado a fase de implementação. Com base nisto, as seguintes métricas foram utilizadas:
  - **ENM:** contabiliza todos os métodos definidos em uma determinada classe.
  - **REM:** contabiliza a quantidade de classes hierárquicas.
  - **AHF:** total de atributos ocultos(privados ou protegidos)em relação ao número total de atributos de uma classe.
  - **MHF:** total de métodos ocultos(privados ou protegidos)em relação ao número total de métodos de uma classe.
  - **NOC:** determina o número de filhos de uma classe.
  - **DIT:** determina o número de níveis ao qual uma determinada classe herda.
  - **NOO:** indica o número de métodos de uma classe.
  - **WMC**: somatório da complexidade de cada método da classe.

  - **EC**: média do número de tipos de dados que uma classe conhece
  - **AC**: número total de classes externas acopladas a classes de um pacote(usa a definição de acoplamento CBO)
  - **LCOM**: indica o nível de coesão de uma classe, baseado nos relacionamentos dos seus atributos e métodos.
- **Observabilidade** Para avaliar o atributo Observabilidade dos sistemas, foi utilizada a métrica definida por Henry e Kafura (1981), pelo fato de ela ser baseada em fluxo de dados.
  - **Information Flow Complexity**: A métrica Information Flow Complexity (IFC) de Henry e Kafura (1981) foi proposta para avaliar a complexidade de um componente com base no volume de informação processado por ele e pode ser dado por:

$$IFC = T * (fan - in * fan - out)^2,$$

    onde os termos *fan-in* e *fan-out* relaciona-se à quantidade de informação de entrada e saída de um dado componente e o termo T refere-se ao tamanho do componente.
- **Controlabilidade** Para definir o atributo Controlabilidade, foram utilizadas métricas que pudessem indicar tanto a complexidade lógica de um programa, que é o caso da complexidade ciclomática, como responsabilidades herdadas pelas casses, que é o caso do número de interfaces.
  - **Complexidade Ciclomática**: A complexidade ciclomática foi utilizada de forma complementar, para avaliar a simplicidade de sistemas orientados a objetos e estruturados, pois ela proporciona uma medida quantitativa da complexidade lógica de um programa (MCCABE, 1976). A complexidade ciclomática é uma métrica baseada na teoria dos grafos e pode ser calculada de três maneiras distintas, entre elas, a seguinte:

$$V(G) = E - N + 2,$$

    onde G é um grafo, E é a quantidade de arestas e N é o número de vértices.
  - **Number of Interfaces**: a métrica Number of Interfaces é uma métrica simples e direta que contabiliza o número de interfaces utilizadas por uma dada classe.
- **Testabilidade** Para definir a característica Testabilidade, foi utilizado um conjunto de métricas relacionadas as classes de testes dos projetos, para que estas pudessem ser

correlacionadas com as métricas referentes aos atributos. Desta forma, as seguintes métricas foram utilizadas:

- **TLOC**: número de linhas de código de uma dada classe de teste.
- **TNOO**: número de operações de uma dada classe de teste.
- **TWMC**: somatório da complexidade de cada método de uma dada classe de teste.
- **NAK**: número de asserts por KLOC.
- **NTM**: número de métodos de teste.

Vale ressaltar que, todas as métricas supracitadas estão disponíveis na planilha[3] com os dados referente à este estudo, nela estão presentes informações adicionais, como interpretação da métrica, ferramenta utilizada, dentre outros campos. Duas tabelas,(1) métricas associadas aos atributos;e (2) métricas associadas a testabilidade também são apresentadas, de forma resumida, no Apêndice A.

6.2 Formulação das hipóteses

- **Hipótese Nula - H0:** Não existe correlação significativa entre simplicidade e a testabilidade de um software orientado a objetos.
- **Hipótese Alternativa: - Ha:** Existe correlação significativa entre o simplicidade e a testabilidade de um software orientado a objetos.
- **Hipótese Nula - H1:** Não existe correlação significativa entre controlabilidade e a testabilidade de um software orientado a objetos.
- **Hipótese Alternativa - Ha1:** Existe correlação significativa entre controlabilidade e a testabilidade de um software orientado a objetos.
- **Hipótese Nula - H2:** Não existe correlação significativa entre observabilidade e a testabilidade de um software orientado a objetos.
- **Hipótese Alternativa: - Ha2:** Existe correlação significativa entre observabilidade e a testabilidade de um software orientado a objetos.

6.3 Seleção das variáveis

Nesta seção são apresentadas as variáveis dependentes e independentes do experimento que teve como objetivo avaliar a testabilidade do software, na Subseção 6.3.1 são

[3] Disponível em: <https://drive.google.com/drive/folders/1OrsyBF6jVfRFlrtor7So6Bm-xQ3GlLBB?usp=sharing>

apresentadas as variáveis dependentes e na Subseção 6.3.2 as variáveis independentes.

6.3.1 Variáveis Dependentes

- **Testabilidade:** A variável dependente desta pesquisa é a testabilidade, por ser o objeto de estudo, ao qual queremos fazer inferências. Como a testabilidade não pode ser medida diretamente, as variáveis dependentes deste estudo são, portanto, aquelas métricas utilizadas para avaliar a testabilidade, que são apresentadas na Tabela 6.

Tabela 6 – Variáveis dependentes

MÉTRICA
TLOC
TNOO
TWMC
NAK
NTM

6.3.2 Variáveis Independentes

- **Simplicidade:** Este atributo foi definido a partir do seguinte conjunto de métricas/variáveis apresentados na Tabela 7.

Tabela 7 – Variáveis independentes relacionadas a simplicidade

ATRIBUTO	MÉTRICA
Simplicidade	ENM
Simplicidade	REM
Simplicidade	AHF
Simplicidade	MHF
Simplicidade	NOC
Simplicidade	DIT
Simplicidade	NOO
Simplicidade	WMC
Simplicidade	EC
Simplicidade	AC
Simplicidade	LCOM

- **Observabilidade:** Este atributo foi definido a partir do seguinte conjunto de métricas/variáveis apresentados na Tabela 8.
- **Controlabilidade:** Este atributo foi definido a partir do seguinte conjunto de métricas/variáveis apresentados na Tabela 9.

Tabela 8 – Variáveis independentes relacionadas a observabilidade

ATRIBUTO	MÉTRICA
Observabilidade	NI
Observabilidade	IFC

Tabela 9 – Variáveis independentes relacionadas a controlabilidade

ATRIBUTO	MÉTRICA
Observabilidade	CC

6.4 Seleção dos sistemas

Foram selecionados três projetos de código aberto disponível na plataforma do Github. Estes sistemas foram selecionados a partir de algumas diretrizes que o autor definiu como critério para seleção, os quais foram:

- O projeto deve ainda estar sendo mantido;
- O projeto deve ser código aberto;
- O projeto deve ter classes de testes implementadas; e
- O projeto deve ter no mínimo 50 classes e no máximo 800.

O ultimo critério foi definido para que fosse selecionado os três sistemas com número de classes nesse intervalo, de modo a representar sistemas de pequeno, médio e grande porte, de tal modo que o sistema com número de classes mais próximo de 50 foi classificado como pequeno porte, e o sistema com número de classes mais próximo a 800 foi classificado como grande porte, e o restante, foi classificado como médio porte. Com base nestes pontos, foram selecionados três projetos para este experimento, que são descritos a seguir:

- **CK-Calculator**[4]**:** O CK calcula as métricas de código em nível de classe e nível de métrica em projetos Java por meio de análise estática (ou seja, não há necessidade de código compilado). Atualmente, ele contém um grande conjunto de métricas, incluindo o famoso CK, que é a suíte de métricas proposta por Chidamber e Kemerer (1994). Este projeto foi classificado como de pequeno porte, por possuir um total de 124 classes (sendo 73 de código e 51 de teste).
- **Joda-Time**[5]**:** O Joda-Time fornece um substituto de qualidade para as classes de data e hora do Java. O Joda-Time é a biblioteca padrão de data e hora do Java antes do Java SE 8. Este projeto foi classificado como de médio porte, por possuir um total de 306 classes (sendo 147 de código e 159 de teste).

[4] Disponível em:<https://github.com/mauricioaniche/ck>
[5] Disponível em:<https://github.com/JodaOrg/joda-time>

- **JFreeChart**[6]**:** O JFreeChart é uma biblioteca de gráficos gratuita abrangente para a plataforma Java (tm) que pode ser usada no lado do cliente (JavaFX e Swing) ou no lado do servidor (com exportação para vários formatos, incluindo SVG, PNG e PDF).Este projeto foi classificado como de médio porte, por possuir um total de 657 classes (sendo 283 de código e 374 de teste).

Vale ressaltar que, no repositório[7] deste estudo, estão disponíveis mais informações sobre cada um desses sistemas, como por exemplo o número de classes, linha de código e até mesmo a classificação quanto ao porte do sistema. Além disso, os artefatos de cada projeto, que foram utilizados para realizar as medições, também estão presentes no repositório do estudo.

6.5 Instrumentação

Para a realização do experimento, foram utilizadas algumas ferramentas, detalhadas no plano experimental[8], que automatizam alguns procedimentos, com o objetivo de poupar tempo em relação a execução destes procedimentos caso fossem feitos manualmente. Para a realização da coleta de dados, foram utilizadas ferramentas para importação do código fonte dos projetos, para que, a partir daí sejam incorporadas outras ferramentas para a computação automática das métricas definidas para o experimento, e no caso das métricas relacionadas ao diagrama de classes, como nenhum dos três projetos disponibilizam o mesmo, foram utilizadas ferramentas de engenharia reversa, para fazer a elaboração deste diagrama e em seguida serem coletadas as métricas.

A análise dos dados foi baseado através do coeficiente de correlação de Pearson, este coeficiente mede o grau de relação entre duas variáveis. A análise foi realizada em duas perspectivas, a primeira foi entre métrica, onde vai ser observado a correlação individual de cada métrica dos atributos com cada métrica de testabilidade, com o objetivo de fazer uma identificação mais específica sobre o relacionamento das propriedades OO com a testabilidade. Já a segunda perspectiva, foi a análise entre atributos, desta vez mais geral e com o objetivo de validar as hipóteses deste experimento, onde foi avaliado a relação entre cada atributo e a característica de qualidade, testabilidade.

[6] Disponível em:<https://github.com/jfree/jfreechart>

[7] Disponível em: <https://drive.google.com/drive/folders/1OrsyBF6jVfRFlrtor7So6Bm-xQ3GlLBB?usp=sharing>

[8] Disponível em: <https://drive.google.com/drive/folders/1OrsyBF6jVfRFlrtor7So6Bm-xQ3GlLBB?usp=sharing>

6.6 Validação

Durante a execução deste experimento, o experimentador teve de que se atentar a alguns pontos que podem vir à ameaçar a validade do estudo. Dessa forma, foram elencados alguns pontos importantes que podem invalidar o experimento, são eles:

- Artefatos indisponíveis no projeto: O fato de não ter todos os artefatos disponíveis para a medição pode ser um fator de ameaça à validade, por este motivo, foram utilizadas ferramentas de engenharia reversa.
- Uso de ferramentas de engenharia reversa: O uso de ferramentas de engenharia reversa pode trazer resultados que não sejam muito precisos. Desta forma, os artefatos gerados por estas ferramentas só foram utilizados para o atributo simplicidade, que possui um maior número de métricas associadas à ele, incluindo métricas com o mesmo objetivo, porém, coletadas a nível de código.

7 EXECUÇÃO

Neste capítulo são apresentado os tópicos referentes à execução do experimento, explanando quais ferramentas e como estas foram utilizadas no decorrer do experimento.

7.1 Preparação do Ambiente

Antes de dar início ao experimento, todo o ambiente foi devidamente preparado seguindo as definições do plano experimental deste estudo, onde os artefatos utilizados para a coleta, diagrama de classes e código fonte das aplicações, foram organizados e preparados (Ver Apêndice B). Para a importação e configuração dos códigos das aplicações foram utilizadas as IDEs NetBeans[9] e Eclipse[10], e para IDE foram instalados *plugins* para que fossem realizadas as coletas diretamente no código importado.

Para que as métricas relacionadas aos atributos não levassem em consideração as métricas de código de teste (relacionadas às classes jUnit), a estrutura do projeto foi organizada de tal modo que fosse possível aplicar as ferramentas de coleta de forma separada: (1) em pacotes que continham apenas classes relacionadas ao código; e (2) pacotes que continham somente classes relacionadas aos testes unitários da aplicação, que seriam as classes jUnit.

Com os códigos devidamente importados, baseando-se na mesma organização estrutural dos pacotes apresentado anteriormente, foram gerados os diagramas de classes das aplicações utilizando engenharia reversa. As ferramentas utilizadas para auxiliar a Engenharia Reversa foram as ferramentas Umbrello UML[11] e Astah[12]. Ambas as ferramentas tem funcionamento similares que consiste em importar todos os arquivos .java da aplicação e em seguida gerar o diagrama arrastando e soltando cada classe relacionada ao arquivo importado, tendo como única diferença os componentes do diagrama de classes que são por cada ferramenta.

Para os sistemas CK-Calculator e Joda-Time, pequeno e médio porte, respectivamente, foi utilizada a ferramenta Umbrello UML, já para o sistema JFreeChart, de grande porte, foi utilizada a ferramenta Astah, isto porque, ao tentar utilizar a ferramenta Umbrello UML no projeto JFreeChart houve um *crash* na ferramenta devido à quantidade de classes do projeto em questão ser muito elevado.

[9] Disponível em: <https://netbeans.org/>
[10] Disponível em: <https://www.eclipse.org/>
[11] Disponível em: <https://umbrello.kde.org/>
[12] Disponível em: <http://astah.net/>

7.2 Coleta de Métricas

Para a coleta de métricas relacionadas ao código das aplicações, foram utilizados os próprios *plugins* disponíveis pelas IDEs, que apesar de coletarem, no geral, um conjunto de métricas em comum, o *plugin* Source Code Metrics[13] do NetBeans, realizava também a coleta de métricas relacionadas as classes de testes, ou seja, as classes jUnit. Após a execução destes *plugins*, todos os dados relevantes (médias e máximos) de cada métrica foram plotados na planilha de coleta deste exprimento, para posterior análise.

Para a coleta de métricas relacionadas aos diagramas de classes das aplicações, foi utilizada a ferramenta SDMetrics[14], que recebe um arquivo .xmi, em que as ferramentas utilizadas para a geração dos diagramas tem suporte a este formato, e coleta uma série de métricas a partir do diagrama de classes. Assim como na coleta de métricas relacionadas ao código, todos as métricas coletadas a partir do diagrama de classe foram também plotadas na planilha de coleta de dados.

7.3 Cálculo dos Coeficientes de Correlação

Após a coleta dos dados e a planilha devidamente preenchida, para realizar os cálculos dos coeficientes de correlação entre métricas, todos os dados referente a cada métrica foram plotados na ferramenta SPSS[15], por projeto, que serviu como amostra do experimento. Com os dados plotados, foi selecionada a opção de análise de coeficiente de Pearson, onde foram definidas as variáveis dependentes e independentes de acordo com o plano experimental.

Para realizar o cálculo do coeficiente de correlação entre atributos, cada atributo será definido pela média dos valores máximos de cada métrica que representa o atributo ou característica para que com isso, cada atributo possa ter um valor numérico associado, por projeto, e, por sua vez viabilizar a realização do cálculo do coeficiente de correlação entre atributos e a testabilidade

[13] Disponível em: <http://plugins.netbeans.org/plugin/42970/sourcecodemetrics>
[14] Disponível em: <https://www.sdmetrics.com/>
[15] Disponível em: <https://www.ibm.com/br-pt/products/spss-statistics>

8 ANÁLISES E RESULTADOS

Neste capítulo, são discutidos os principais resultados deste estudo, além de apresentar as análises mais importantes que puderam ser inferidas a partir de perspectivas diferentes, possibilitando obter uma visão geral sobre os atributos identificados e a testabilidade de software.

8.1 Resultados

Os resultados da coleta de dados estão presentes no repositório[16] do estudo, nele, estão detalhados os valores respectivos à cada métrica, separados por projeto. Vale ressaltar que, na planilha que contém os resultados da coleta, existe também uma aba contendo mais informações relacionadas as métricas coletadas, como ferramenta utilizada e calculo da métrica.

Com os dados coletados, foi realizada uma análise estatística baseada no coeficiente de correlação de Pearson, que indica o nível de correlacionamento entre variáveis, estes coeficientes podem ser vistos de forma completa nos Anexos A e B, e foram calculados a partir da coleta de cada métrica por sistema. A Tabela 10 mostra, de forma resumida, os coeficientes de correlação entre todas as métricas utilizadas neste experimento (as relacionadas aos atributos estão nas linhas e as relacionadas a testabilidade estão nas colunas).

Tabela 10 – Coeficiente de correlação entre métricas

Atributos	Testabilidade				
	TLOC	TNOO	TWMC	NAK	NTM
ENM	-0,651	-0,592	-0,668	-0,666	-0,388
REM	-0,543	-0,605	-0,524	-0,526	-0,773
AHF	0,543	0,605	0,524	0,526	0,773
MHF	-0,697	-0,641	-0,713	-0,712	-0,445
NOC	-0,115	-0,189	-0,092	-0,094	-0,412
DIT	0,397	0,327	0,419	0,417	0,099
DIT-IMP	0,954	0,929	0,961	0,96	0,817
NOC-IMP	0,896	0,927	0,885	0,886	0,989
NOO	0,478	0,543	0,458	0,46	0,723
WMC	0,696	0,748	0,679	0,681	0,882
EC	0,962	0,98	0,956	0,956	0,999
AC	0,993	0,999	0,99	0,99	0,982
LCOM	0,902	0,932	0,892	0,893	0,991
CC	0,944	0,916	0,951	0,95	0,798
NI	0,854	0,812	0,866	0,865	0,654
IFC	0,954	0,974	0,946	0,947	1

[16] Disponível em: <https://drive.google.com/drive/folders/1OrsyBF6jVfRFlrtor7So6Bm-xQ3GlLBB?usp=sharing>

Para determinar se uma métrica específica, associada à algum atributo influenciou ou não na testabilidade, foi analisado o relacionamento de cada métrica relacionada aos atributos, com as métricas relacionadas à testabilidade, baseando-se nas seguintes regras:

1) Se a métrica obteve coeficiente de correlação >= |0,6| com pelo menos outras 3 métricas de testabilidade, considera-se que houve influência da métrica na testabilidade.
2) Se a métrica obteve coeficiente de correlação <= |0,6| com pelo menos outras 3 métricas de testabilidade, considera-se que não houve influência da métrica na testabilidade.

O valor de 0,6 foi utilizado pois, segundo Mukaka (2012), um coeficiente entre 0,5 e 0,7, apresenta uma correlação moderada, já o valor 3, representa a metade mais um do total de métricas utilizadas para a testabilidade, ou seja, para que uma métrica influencie na testabilidade ela deve ter tido uma correlação moderada com pelo menos 3 das 5 métricas associadas à testabilidade. Com base nestas regras, foram gerados dois conjuntos, os de métricas que influenciaram na testabilidade, e o de métricas que não influenciaram, para que se possa ter uma visão geral sobre estas métricas. A Tabela 12, apresenta todas as métricas utilizadas no experimento que influenciaram na testabilidade, seguindo as regras definidas. Nela, estão dispostas, além de métricas que influenciaram, a fase em que a métricas influenciou e a descrição justificando a sua influência sobre a testabilidade.

Assim como na Tabela 12, a Tabela 11 apresenta uma lista de métricas que não influenciaram na testabilidade do software, seguindo os mesmos conceitos dos campos da Tabela 12.

Tabela 11 – Métricas que não influenciaram na testabilidade

MÉTRICA	**FASE**	**Descrição**
REM	Design	Apresentou coeficiente de correlação menor que |0,6| com as métricas TLOC,TWMC e NAK
AHF	Design	Apresentou coeficiente de correlação menor que |0,6| com as métricas TLOC,TWMC e NAK
DIT	Design	Apresentou coeficiente de correlação menor que |0,6| com todas as métricas de testabilidade
NOC	Design	Apresentou coeficiente de correlação menor que |0,6| com todas as métricas de testabilidade
NOO	Implementação	Apresentou coeficiente de correlação menor que |0,6| com as métricas TLOC, TNO, TWMC e NAK

Ao analisar as duas Tabelas (12 e 11), pode se observar que a maioria das métricas selecionadas tiveram influência positiva sobre a testabilidade de software, além de que, boa

Tabela 12 – Métricas que influenciaram na testabilidade

MÉTRICA	FASE	Descrição
ENM	Design	Apresentou coeficiente de correlação maior que |0,6| com as métricas TLOC,TWMC e NAK
MHF	Design	Apresentou coeficiente de correlação maior que |0,6| com as métricas TLOC,TNOO,TWMC e NAK
DIT	Implementação	Apresentou coeficiente de correlação maior que |0,6| com as métricas TLOC, TNOO, TWMC, NAK e NTM
NOC	Implementação	presentou coeficiente de correlação maior que |0,6| com as métricas TLOC, TNOO, TWMC, NAK e NTM
WMC	Implementação	Apresentou coeficiente de correlação maior que |0,6| com todas as métricas de testabilidade
EC	Implementação	Apresentou coeficiente de correlação maior que |0,6| com todas as métricas de testabilidade
AC	Implementação	Apresentou coeficiente de correlação maior que |0,6| com todas as métricas de testabilidade
LCOM	Implementação	Apresentou coeficiente de correlação maior que |0,6| com todas as métricas de testabilidade
NOM	Implementação	Apresentou coeficiente de correlação maior que |0,6| com todas as métricas de testabilidade
NI	Implementação	Apresentou coeficiente de correlação maior que |0,6| com todas as métricas de testabilidade
IFC	Implementação	Apresentou coeficiente de correlação maior que |0,6| com todas as métricas de testabilidade

parte das métricas que influenciaram, obtiveram um bom coeficiente de correlação com todas as métricas de testabilidade. Da mesma forma, todas as métricas que não influenciaram obtiveram um baixo coeficiente de correlação com todas as métricas de testabilidade utilizadas.

8.2 Análise

Com base nos resultados deste experimento, apresentado na subseção 8.1, foram realizadas duas análises distintas: (1) uma análise entre métricas, onde foi observado a correlação individual de cada métrica dos atributos com cada métrica de testabilidade, com o objetivo de fazer uma identificação mais específica sobre o relacionamento das propriedades do paradigma orientado a objetos com a testabilidade; e (2) uma análise entre atributos, desta vez mais geral e com o objetivo de validar as hipóteses deste experimento, onde foi avaliado a relação entre cada atributo e a característica de qualidade, testabilidade.

8.2.1 Análise entre métricas

Com base nos resultados, observou que a maioria da métricas que não influenciaram na testabilidade foram relacionadas à Simplicidade e coletadas a partir do diagrama de classes, como por exemplo, *Reuse Metric (REM)* e *Attribute Hidden Factor (AHF)*, enquanto que a maioria das métricas que influenciaram foram relacionadas a simplicidade, por ter um conjunto maior de métricas, e foram coletadas a partir do código dos projetos. Isto pode indicar que, na fase de design, os diagramas de classe foram modelados de uma forma mais simples do que a forma como foi codificada, ou simplesmente indicar que as ferramentas utilizadas para realizar a engenharia reversa não recuperou totalmente os componentes, o que é mais provável, visto que, se for feita uma observação da coleta destas métricas que não influenciaram, por projeto, será possível observar que, o projeto com os maiores valores, foram o do projeto de médio porte (Joda-Time), superando as métricas do sistema de grande porte(JFreeChart). Além disso, se formos analisar o diagrama de classes de cada projeto, disponível no repositório[17] deste estudo, pode se verificar que o diagrama mais completo(com mais classes e associações), são justamente os do projeto Joda-Time.

As únicas métricas coletadas a partir do diagrama de classes que influenciaram na testabilidade foram *Encapslulation Metric (ENM)* e *Methods Hidden Factor (MHF)*, ambas estão relacionadas a contagem de métodos das classes, indicando que a quantidade de métodos impacta na testabilidade. No entanto, quando fazemos uma comparação com a métrica *Number of Operations (NOO)*, que não influenciou na testabilidade, mas que também está relacionada a contagem de métodos, porém, coletada a partir do código, pode-se observar que há uma divergência, no diagrama de classe o número de métodos influenciou, mas no código não influenciou. Este comportamento pode ser justificado pelo fato de que, no código, muitos métodos que servem apenas como métodos auxiliares (como getters e setters ou metódos de API/Framework), foram contabilizados, mas não impactaram de fato na testabilidade, uma vez que para esses tipos de métodos não haviam sido criados testes unitários no JUnit.

A métrica *Information Flow Complexity (IFC)* obteve, em relação à métrica *Number of Test Methods (NTM)*, o maior coeficiente de correlação do experimento, sendo considerada uma correlação perfeita (1,0). Isto significa que sempre que o valor da métrica *Number of Test Methods* aumentou, a métrica *Information Flow Complexity* também aumentou, podendo dar

[17] Disponível em: <https://drive.google.com/drive/folders/1OrsyBF6jVfRFlrtor7So6Bm-xQ3GlLBB?usp=sharing>

uma representação de que a utilização de muitos métodos podem tornar os projetos menos controláveis.

Outro importante ponto de análise é o fato de que as métricas *Depth Inheritance Tree (DIT)* e *Number of Childrens (NOC)* tiveram valores muito divergentes, quando comparado implementação vs design. Esta divergência também pode ser fundamentada pela recuperação de componentes por parte das ferramentas de engenharia de reversa, uma vez que, estas métricas são calculadas a partir de relacionamentos entre classes e como foi discutido anteriormente, alguns diagramas não possuem todas as associações, ou pelo menos a maioria, no entanto, estas relações que não foram capturadas no diagrama de classes, foram capturadas a partir do código fonte, por meio de expressões como *extends* e *implements*. Continuando na mesma linha de análise, como esperado, ambas as métricas tiveram coeficiente de correlação similares, seja no diagrama de classes ou no código fonte, já que as duas métricas estão relacionadas com a profundidade das associações, diferenciando apenas o sentido, enquanto uma conta o número de sucessores, a outra conta o número de antecessores.

As métricas *Efferent Coupling (EC)* e *Afferent Coupling (AC)* tiveram, com quase todas as outras métricas de testabilidade uma correlação perfeita (1,0), indicando que o nível de acoplamento aumentou de um projeto para outro, impactando negativamente na testabilidade. Este cenário pode abrir portas para algumas discussões interessantes, como por exemplo, "Como modelar e utilizar hierarquias ou interfaces?", "Como modelar ou implementar classes sem muitas dependências entre elas?"ou até mesmo "Quando e como reutilizar código?". Estas discussões são importantes, pois podem ser um fator de peso na testabilidade de um software. Além disso, o que é possível observar nos projetos sob análise deste estudo é que todos eles possuem muita dependências, tanto no uso de hierarquia, quanto no uso de métodos de outras classes e implementação de métodos de interfaces.

Por fim, mas não menos importante, ao analisar a coesão dos projetos, um outro pilar da orientação a objetos, por meio da métrica *Lack of Cohesion Metric (LCOM)*, é possível verificar que houve uma correlação significativa, ou seja, maior que 0,6 com todas as métricas de testabilidade. Isto indica que, quanto menos coeso foram os projetos, menos testáveis eles foram considerados. Assim como o acoplamento, a falta de coesão nos instiga à alguns questionamentos, como por exemplo "Quais responsabilidades devem ser de uma classe x?"ou "Quais os objetivos de determinada classe?". A partir do código foi possível observar que muitas classes possuem métodos que realizam operações que vão além de suas responsabilidades, o que pode ter feito

com que o índice de LCOM tenha aumentado de um projeto para outro, porém, por outro lado, esta métrica contabiliza todos os métodos, incluindo os getters e setters, o que pode ter aumentado ainda mais o nível da LCOM.

8.2.2 Análise entre atributos

Nesta seção será tratada a análise do relacionamento dos atributos com a característica sob estudo desta pesquisa, a testabilidade, além disso, será analisado as validade das hipóteses definidas no plano experimental[18]. A Tabela 13 abaixo, mostra, de forma resumida, os coeficientes de correlação entre os três atributos e a testabilidade, onde, a linha representa a testabilidade e as colunas representam os atributos.

Tabela 13 – Coeficiente de correlação entre atributos

Caracteristica	Atributos		
	Simplicidade	Controlabilidade	Observabilidade
Testabilidade	0,793	0,928	0,955

Com base nestes valores, seguindo Mukaka (2012), pode-se garantir que todos os atributos tiveram uma correlação forte com a testabilidade, no entanto, estes valores evidencia alguns pontos de análises, como no caso do atributo simplicidade, que foi o que obteve menor coeficiente de correlação (0,793) dentre os demais atributos, isto se justifica porque este atributo é definido por várias métricas, dentre elas as coletadas a partir do diagrama de classes e as coletadas a partir do código, e como discutido anteriormente, na subseção 8.2.1, algumas métricas coletadas a partir do diagrama de classes podem ter influenciada pela recuperação de componentes das ferramentas de engenharia reversa. Os outros dois atributos, observabilidade e controlabilidade, obtiveram correlação quase que perfeita,ou seja, bem próximo de 1, com a testabilidade, pois as métricas que definem estes dois atributos tiveram impacto muito positivo nas métricas de testabilidade. Um ponto em comum entre os dois atributos é que ambos lidam com o fluxo de informações e dados, indicando que a complexidade envolvendo o fluxo das informações nos projetos de software pode afetar negativamente a testabilidade, introduzindo mais complexidade na atividade de testes.

Assim como no estudo de Kout *et al.* (2011), as hipóteses deste estudo foram analisadas a partir do coeficiente de correlação, diferenciando no sentido de que, enquanto no

[18] Disponível em: <https://drive.google.com/drive/folders/1OrsyBF6jVfRFlrtor7So6Bm-xQ3GlLBB?usp=sharing>

trabalho de Kout *et al.* (2011) foram avaliados os coeficientes de correlações das métricas, este estudo avaliou os coeficientes de correlações dos atributos. Baseado nisso, as seguintes hipóteses foram aceitas:

- **Hipótese Alternativa: - Ha:** Existe correlação significativa entre o simplicidade e a testabilidade de um software orientado a objetos. Sim, o atributo simplicidade obteve um coeficiente de correlação de 0,793, apresentando uma forte correlação positiva com a testabilidade.
- **Hipótese Alternativa - Ha1:** Existe correlação significativa entre controlabilidade e a testabilidade de um software orientado a objetos. Sim, o atributo controlabilidade obteve um coeficiente de correlação de 0,928, apresentando uma correlação positiva quase que perfeita com a testabilidade.
- **Hipótese Alternativa: - Ha2:** Existe correlação significativa entre observabilidade e a testabilidade de um software orientado a objetos. Sim, o atributo observabilidade obteve o maior coeficiente de correlação com a testabilidade, de 0,955, dentre os outros atributos, apresentando uma correlação positiva quase que perfeita com a testabilidade.

As demais hipóteses levantadas neste estudo, que afirma que para cada atributo não existe correlação significativa com a testabilidade, por sua vez, foram rejeitadas, uma vez que as hipóteses nulas afirmam que não existe correlação significativa entre cada atributo e a testabilidade de software.

Além de examinar as hipóteses, a análise entre atributos permitiu que as questões de pesquisa estabelecidas na Seção 1.1 fossem respondidas. A partir do experimento, foi possível chegar nas seguintes conclusões:

- **RQ1:** Foram identificados um total de 12 atributos que podem ser utilizados para avaliar a testabilidade, no entanto, este estudo se restringiu à analisar 3 atributos em específico, sendo eles a controlabilidade, observabilidade e simplicidade, por serem atributos que os autores mais utilizaram e por ter sido possível coletar dados refentes a estes atributos por meio dos artefatos disponíveis por projeto. Baseado no experimento realizado, pode-se observar que estes três atributos influenciaram positivamente na testabilidade, podendo ser utilizados em futuras avaliações.
- **RQ2:** Conforme discutido na Seção 8.2.1, foram identificadas as métricas que influenciaram e as que não influenciaram na testabilidade. Por sua vez, algumas métricas podem ter sido descartadas devido à coleta ter sido realizada a partir de um artefato gerado utilizando

engenharia reversa, o que faz com que um estudo mais aprofundado seja realizada para que evidências significativas possam ser obtidas.

- **RQ3:** Baseado no experimento realizado, para avaliar a testabilidade de um software, é necessário que se tenha um bom planejamento estratégico, de modo que fique bem definidos que atributos e métricas devem ser utilizadas, de acordo com o sistema. Este trabalho apresentou uma avaliação a partir de um contexto específico, baseando-se em três atributos e restringindo-se ao paradigma orientado a objetos, apesar das limitações foi possível realizar o experimento e obter análises a partir dos resultados, a cerca da testabilidade do software.

9 CONCLUSÃO E TRABALHOS FUTUROS

A partir deste estudo, foi possível preencher algumas lacunas existentes sobre o tema, como o fato de utilizar atributos para medir a testabilidade de um software e a avaliação da testabilidade a partir de artefatos de fases distintas (design e implementação). O estudo teórico permitiu obter uma visão geral do estado da arte com relação ao uso de atributos de testabilidade, o que proporcionou também a identificação de métricas baseado nos atributos. No entanto, o estudo também permitiu evidenciar algumas lacunas ainda existentes. Os resultados obtidos indicaram que os atributos identificados possuem uma relação direta com a testabilidade, no entanto, como mostrado nas discussões anteriores e no própio planejamento experimental[19].

Durante as análises e discussões do estudo, foi possível enfatizar alguns aspectos do ciclo de vida de desenvolvimento do software que podem influenciar na testabilidade de software, o que pode servir como base para que futuros projetos baseiem-se nestes pontos durante as fases iniciais do projeto, o que é de suma relevância, visto que atualmente as empresas estão cada vez mais buscando aprimorar seus processos, utilizando métodos, ferramentas e abordagens que possam alavancar a produtividade do time responsável pelo projeto. Além disso, boas decisões nas fases iniciais do projeto aliado à experiência da equipe podem contribuir solidamente como um fator de sucesso de um projeto de software.

Por meio de algumas limitações durante o estudo, ficou constatado que, avaliar a testabilidade de um software pode não ser uma tarefa tão trivial, exigindo rigor e boas decisões e estratégias durante o planejamento da medição desta característica. Como limitações desta pesquisa, pode se elencar alguns pontos, como, o fato da testabilidade ter sido avaliada de uma perspectiva de testes unitários, por meio de classes JUnit, a falta de artefatos disponíveis para medições mais precisas sobre esta característica e a necessidade de apoio ferramental para a coleta de métricas. No entanto, estas limitações permitiram que fosse observado que a testabilidade de software é uma característica complexa, e que, de fato, não pode ser medida diretamente, exigindo que, ao avaliar esta característica, estudos e investigações sejam realizadas acerca do tema para que sejam definidos os melhores procedimentos para avaliação da testabilidade.

A testabilidade de software se mostrou um fator crucial em projetos de software, pelo fato de poder nos dar uma representação sobre a complexidade da atividade de testes de um projeto. Além disso, identificar os aspectos que estão impactando, tanto negativamente, quanto

[19] Disponível em: <https://drive.google.com/drive/folders/1OrsyBF6jVfRFlrtor7So6Bm-xQ3GlLBB?usp=sharing>

positivamente pode enriquecer e melhorar cada vez mais esta atividade e consequentemente atingir um maior nível de qualidade nos projetos de software.

Com base nas contribuições e limitações deste estudo, foi possível observar que vários caminhos podem ser seguidos como trabalhos futuros desta pesquisa. Dentre estes caminhos possíveis, há alguns interessantes de serem citados, como a realização de um estudo de caso na indústria, no qual esteja disponível todos os artefatos necessários para a realização de uma medição completa desta característica; avaliar a testabilidade não só da perspectiva de testes unitários, mas também de testes de integração ou até mesmo de sistemas;e avaliar a testabilidade não só a partir do código das aplicações, mas também a partir da especificação de testes, por exemplo. É válido ressaltar que, não foi encontrado até então abordagens mais sólidas, como um checklist para a seleção de atributos para avaliar a testabilidade, baseado em um conjunto de atributos, o que seria interessante, visto que, alguns atributos podem ser mais relevantes à algum domínio específico, necessitando de um estudo mais aprofundado sobre o assunto.

REFERÊNCIAS

BACH, J. **Heuristics of Software Testability**. 1994. Disponível em: <newsgroupcomp.software.eng>. Acesso em: 28 ago. 2018.

BINDER, R. Design for Testability in Object-Oriented Systems. **Communication of ACM**, v. 37, n. 9, p. 87–100, 1994.

CHIDAMBER, S.; KEMERER, C. A metrics suite for object oriented design. **IEEE Transactions on Software Engineering**, v. 20, n. 6, p. 476–493, June 1994.

CRUZ, R. C. da. **Análise empírica sobre a influência das métricas CK na testabilidade de software orientado a objetos**. Dissertação (Mestrado em Ciências) — Universidade de São Paulo, São Paulo, 2018.

DAVIS, A. **201 Principles of Software Development**. 4th. ed. McGraw-Hill, 1995. ISBN 9780070158405. Disponível em: <https://books.google.com.br/books?id=8HAhAQAAIAAJ>. Acesso em: 24 ago. 2018.

DAVIS, F. D.; REYNOLDS, A. D.; KINCAID. Identifying and measuring quality in a software requirements specification. In: IEEE. **Proceedings First International Software Metrics Symposium**. Baltimore, MD, USA, USA, 1993. p. 141–152.

DAYANANDAN, U.; VIVEKANANDAN, K. An empirical evaluation model for software architecture maintainability for object oriented design. In: ACM DL. **Proceedings of the International Conference on Informatics and Analytics**. New York, NY, USA: ACM, 2016. (ICIA-16), p. 98:1–98:4. ISBN 978-1-4503-4756-3.

FREEDMAN, R. S. Testability of software components. **IEEE Transactions on Software Engineering**, v. 17, n. 6, p. 553–564, June 1991.

GAFFNEY, J. E. Metrics in software quality assurance. In: **Proceedings of the ACM '81 Conference**. New York, NY, USA: ACM, 1981. (ACM '81), p. 126–130.

GAO, J.; SHIH, M. C. A component testability model for verification and measurement. In: IEEE. **29th Annual International Computer Software and Applications Conference (COMPSAC'05)**. [S.l.], 2005. v. 2, p. 211–218 Vol. 1.

GARVIN, D. Competing on the eight dimensions of quality. **Harvard Business Review 65**, n. 6, p. 101–109, 1987.

HENRY, S.; KAFURA, K. Software structure metrics based on information flow. **IEEE Transactions on Software Engineering**, SE-7, n. 5, p. 510–518, Sep. 1981.

IEEE. Standard for a software quality metrics methodology. **IEEE Std 1061-1992**, p. 1–96, March 1993.

JURECZKO, M.; SPINELLIS, D. Using object-oriented design metrics to predict software defects. **Models and Methods of System Dependability. Oficyna Wydawnicza Politechniki Wrocławskiej**, p. 69–81, 2010.

KHALID, S.; ZEHRA, S.; ARIF, F. Analysis of object oriented complexity and testability using object oriented design metrics. In: **Proceedings of the 2010 National Software Engineering Conference**. New York, NY, USA: ACM, 2010. (NSEC '10), p. 4:1–4:8.

KHAN, R. A.; MUSTAFA, K. Metric based testability model for object oriented design (mtmood). **SIGSOFT Softw. Eng. Notes**, ACM, New York, NY, USA, v. 34, n. 2, p. 1–6, fev. 2009.

KOSCIANSKI, A.; SOARES, M. dos S. **Qualidade de Software: Aprenda as metodologias e técnicas mais modernas para o desenvolvimento de software**. 2ª. ed. Novatec, 2007. ISBN 9788575221129. Disponível em: <https://books.google.com.br/books?id=O9aWoUq6L88C>. Acesso em: 05 ago. 2018.

KOUT, A.; TOURE, F.; BADRI, M. An empirical analysis of a testability model for object-oriented programs. **SIGSOFT Softw. Eng. Notes**, ACM, New York, NY, USA, v. 36, n. 4, p. 1–5, ago. 2011. ISSN 0163-5948.

KUDRJAVETS, G.; NAGAPPAN, N.; BALL, T. Assessing the relationship between software assertions and faults: An empirical investigation. In: IEEE. **Proceedings - International Symposium on Software Reliability Engineering, ISSRE**. [S.l.], 2006. p. 204–212.

LEACH, R. **Introduction to Software Engineering**. 2ª. ed. 2016. ISBN 9781498705288. Disponível em: <https://books.google.com.br/books?id=8W2mCwAAQBAJ>. Acesso em: 20 ago. 2018.

MCCABE, T. J. A complexity measure. **IEEE Transactions on Software Engineering**, SE-2, n. 4, p. 308–320, Dec 1976.

MCCALL, J. A.; RICHARDS, P. K.; WALTERS, G. F. Factors in software quality. **US Rome Air development center reports**, GENERAL ELECTRIC CO SUNNYVALE CA, v. 1, n. 3, p. 77–369, 1977.

MUKAKA, M. Statistics corner: A guide to appropriate use of correlation coefficient in medical research. **Malawi medical journal : the journal of Medical Association of Malawi**, v. 24, p. 69–71, Sep 2012.

MYERS, G. **The Art of Software Testing**. 3th. ed. [S.l.]: Wiley, 1979. (A Wiley-Interscience publication). ISBN 9780471043287.

NUñEZ-VARELA, A. S.; PéREZ-GONZALEZ, H. G.; MARTíNEZ-PEREZ, F. E.; SOUBERVIELLE-MONTALVO, C. Source code metrics: A systematic mapping study. **Journal of Systems and Software**, v. 128, p. 164 – 197, 2017.

PRESSMAN, R. **Engenharia de Software**. 7th. ed. [S.l.]: McGraw Hill Brasil, 2009. ISBN 9788580550443.

SOMMERVILLE, I. **Engenharia de software**. 9th. ed. [S.l.]: PEARSON BRASIL, 2011. ISBN 9788579361081.

SPEARMAN, C. The proof and measurement of association between two things. **The American Journal of Psychology**, University of Illinois Press, v. 15, n. 1, p. 72–101, 1904. ISSN 00029556.

TI, E. em. **Métricas de Qualidade de Software**. 2016. Disponível em: <https://www.tiespecialistas.com.br/metricas-de-qualidade-de-software/>. Acesso em: 02 jul. 2018.

TSUI, F.; IRIELE, S. Analysis of software cohesion attribute and test case development complexity. In: ACM DL. **Proceedings of the 49th Annual Southeast Regional Conference**. New York, NY, USA: ACM, 2011. (ACM-SE '11), p. 237–242.

TU, H.; SUN, W.; ZHANG, Y. The research on software metrics and software complexity metrics. In: IEEE. **IFCSTA 2009 Proceedings - 2009 International Forum on Computer Science-Technology and Applications**. [S.l.], 2009. v. 1, p. 131–136.

APÊNDICE A – TABELAS DE ASSOCIAÇÃO DAS MÉTRICAS

ASSOCIAÇÃO ENTRE MÉTRICAS E ATRIBUTOS

Atributo	Métrica	Cálculo da Métrica	Interpretação
SIMPLICIDADE	ENM	Contagem do número de operações (métodos) da classe	Quanto maior, maior o nível de encapsulamento
SIMPLICIDADE	REM	Contagem do número de hierarquia de classes	Quanto maior, mais dependente a classe será.
SIMPLICIDADE	AHF	Contagem do número de atributos ocultos dividido pelo número de total de atributos	Quanto maior, menos complexo será o código.
SIMPLICIDADE	MHF	Contagem do número de métodos ocultos dividido pelo número de total de métodos	Quanto maior, menos complexo será o código.
SIMPLICIDADE	NOC	Contagem do número de sucessores de uma classe	Quanto maior, mais dependente a classe será.
SIMPLICIDADE	DIT	Contagem do número de antecessores de uma classe	Quanto maior, mais dependente a classe será.
SIMPLICIDADE	NOO	Contagem do número de operações de uma classe	Quanto maior, maior o nível de encapsulamento
SIMPLICIDADE	WMC	Somatório da complexidade de cada método da classe.	Quanto maior, mais complexa será a classe.
SIMPLICIDADE	EC	Média do número de tipos de dados que uma classe conhece	Quanto maior, menos especifica e mais responsabilidade terá o componente
SIMPLICIDADE	AC	Número total de classes externas acopladas a classes de um pacote (usa a definição de acoplamento CBO)	Quanto maior, mais classes externas acopladas terá a classe
SIMPLICIDADE	LCOM	LCOM = P = NP – Q, onde: P é o número de métodos que não compartilham atributos, Q é o número de métodos que compartilham atributos e NP é dado pela função M! / 2! . (M – 2)!, sendo M o número de métodos existentes na classe.	Quanto maior, menos coeso será o código
OBSERVABILIDADE	CC	V(G) = E −N +2, onde G é um grafo, E é a quantidade de arestas e N é o número de vértices	Geralmente, um valor máximo entre 15-10 é aceitável
OBSERVABILIDADE	NI	Contagem do número de interfaces utilizadas por uma dada classe	Quanto maior, mais responsabilidade de implementação terá a classe
CONTROLABILIDADE	IFC	$T * (f\,an{-}in * f\,an{-}out)^2$, fan-in e fan-out relaciona-se à quantidade de informação de I/O de um dado componente e T refere-se ao tamanho do componente.	Quanto maior, maior será a complexidade envolvendo o fluxo de dados.

ASSOCIAÇÃO ENTRE MÉTRICAS E TESTABILIDADE

Característica	Métrica	Cálculo da Métrica	Interpretação
TESTABILIDADE	TLOC	Contagem de linhas de uma classe de teste	Quanto maior, mais grande será o código de teste
TESTABILIDADE	TNOO	Contagem do número de operações das classes de teste	Quanto maior, mais operações terá a classe de teste
TESTABILIDADE	TWMC	Somatório da complexidade de cada método do caso de teste.	Quanto maior, mais complexa será a classe de teste.
TESTABILIDADE	NAK	Contagem do número de *asserts* por KLOC	Quanto maior, mais testes estão sendo feitos por KLOC, por uma classe
TESTABILIDADE	NTM	Contagem do número de métodos de teste	Quanto maior, mais testes estão sendo feitos por KLOC, por uma classe

APÊNDICE B – PLANEJAMENTO EXPERIMENTAL

Estudo sobre atributos que influenciam da testabilidade de software

Planejamento Experimental

Aluno: Francisco Gutenberg da Silva Filho

Orientadora: Valéria Lelli Leitão Dantas

07 de Junho de 2019

Sumário

1. Introdução

A exigência por maior qualidade de software vem crescendo à medida que os sistemas estão mais presentes nas atividades do cotidiano, embora a ideia de qualidade de software seja intuitiva, ao ser analisada minuciosamente, esse conceito se torna complexo. Por ser complexo, muitos autores e normas técnicas propõem modelos, que facilitam a avaliação da qualidade de software. No geral, os modelos de qualidade (e.g., SQUARE 25000) definem um conjunto de características ou fatores que estão diretamente relacionados com a qualidade de software. Uma característica importante presente na maioria desses modelos é a testabilidade, pois ela possui relação com testes de software, que tem como objetivo evidenciar que um programa faz o que é proposto a fazer enquanto testabilidade pode ser definida como a facilidade com que um programa pode ser testado. Dessa forma, avaliar a testabilidade de um software traz informações relevantes sobre a qualidade de um sistema. Porém, a testabilidade não é uma característica intrínseca do software, e, portanto, deve se identificar ou definir medidas que deem uma indicação direta sobre testabilidade. A proposta deste estudo é investigar a testabilidade de software com o intuito de propor um guia que auxilie na avaliação dessa característica. Na primeira etapa do estudo foi feita uma revisão bibliográfica para identificar as características de testabilidade e as métricas que representam essas características. Na segunda etapa, com base na identificação feita na etapa anterior, foi realizado um experimento em sistemas de código aberto, para avaliar a testabilidade desses sistemas. Como resultados, este trabalho apresenta algumas métricas e atributos identificados na literatura para avaliar a testabilidade de software.

2. Planejamento Experimental

a. Seleção do Contexto

Este experimento foi realizado de maneira ad-hoc, com objetivo de avaliar a testabilidade de software por meio dos atributos identificados. Serão selecionados três sistemas de código aberto, e para cada sistema serão coletados dados referente as métricas, que por sua vez determinam cada atributo. A realização deste experimento foi conduzido de maneira automática, utilizando ferramentas que automatizam, principalmente, os procedimentos de coletas e análise de dados. Como resultados do experimento, foram identificadas correlações entre atributos e entre métricas definidas para o experimento.

b. Formulação das hipóteses

i. **Hipótese Nula - H0:** Não existe correlação significativa entre simplicidade e a testabilidade de um software orientado a objetos.

ii. **Hipótese Alternativa - Ha:** Existe correlação significativa entre o simplicidade e a testabilidade de um software orientado a objetos.

iii. **Hipótese Nula - H1:** Não existe correlação significativa entre controlabilidade e a testabilidade de um software orientado a objetos.

iv. **Hipótese Alternativa - Ha1:** Existe correlação significativa entre controlabilidade e a testabilidade de um software orientado a objetos.

v. **Hipótese Nula - H2:** Não existe correlação significativa entre observabilidade e a testabilidade de um software orientado a objetos.

vi. **Hipótese Alternativa - Ha2:** Existe correlação significativa entre observabilidade e a testabilidade de um software orientado a objetos.

c. Seleção das variáveis

i. Dependentes

1. Testabilidade

   Esta característica será definido a partir do seguinte conjunto de métricas/variáveis:

   a. TNOO
   b. TLOC
   c. TWMC
   d. NAK
   e. NTM

ii. Independentes

1. Simplicidade

   Este atributo será definido a partir do seguinte conjunto de métricas/variáveis:

   a. ENM
   b. REM
   c. AHF
   d. MHF
   e. NOC
   f. DIT
   g. NOC
   h. NOO

i. DIT (Implementação)

j. NOC (Implementação)

k. NOO

l. WMC

m. EC

n. AC

o. LCOM

2. Observabilidade

   Este atributo será definido a partir do seguinte conjunto de métricas/variáveis:

   a. IFC
   b. NI

3. Controlabilidade

   Este atributo será definido a partir do seguinte conjunto de métricas/variáveis:

   a. CC

Estas métricas, estão disponíveis no repositório do experimento, na seção **Informações Gerais**. Nela, as métricas são definidas e classificadas.

d. Seleção dos Sistemas

Foram selecionados três projetos de código aberto disponível na plataforma do Github. Estes sistemas foram selecionados a partir de alguns pontos que o autor definiu como critério para seleção, estes pontos foram:

1. O projeto deve ainda estar sendo mantido
2. O projeto deve ser open source
3. O projeto deve ter classes de testes implementadas
4. O projeto deve ter no mínimo 50 classes e no máximo 800.

Com base nestes pontos, foram selecionados três projetos para este experimento, que são descritos a seguir:

i. ***CK-Calculator***

O CK calcula as métricas de código em nível de classe e nível de métrica em projetos Java por meio de análise estática (ou seja, não há necessidade de código compilado). Atualmente, ele contém um grande conjunto de métricas, incluindo o famoso CK.

Informações gerais:

Núm Classes	**Funções**	**Linhas de Código**	**Porte**
124(73 código/51 teste)	1008	9763	Pequeno

ii. ***Joda-Time***

O Joda-Time fornece um substituto de qualidade para as classes de data e hora do Java. O Joda-Time é a biblioteca padrão de data e hora do Java antes do Java SE 8. Agora, os usuários são solicitados a migrar para o java.time (JSR-310).

Informações gerais:

Núm Classes	**Funções**	**Linhas de Código**	**Porte**
306(147 código/159 teste)	8046	144.079	Médio

iii. ***JFreeChart***

O JFreeChart é uma biblioteca de gráficos gratuita abrangente para a plataforma Java (tm) que pode ser usada no lado do cliente (JavaFX e Swing) ou no lado do servidor (com exportação para vários formatos, incluindo SVG, PNG e PDF).

Informações gerais:

Núm Classes	**Funções**	**Linhas de Código**	**Porte**
657(283 código/374 teste)	9369	279.106	Grande

e. Instrumentação

Para a realização deste experimento, foram utilizados algumas ferramentas que automatizam alguns procedimentos, com o objetivo de poupar tempo em relação a execução destes procedimentos caso fossem feitos manualmente. Estas ferramentas se aplicam à etapas diferentes do experimento, sendo elas:

1) Coleta dos dados
2) Análise de resultados

Para a realização da coleta de dados, serão utilizadas ferramentas para importação do código fonte dos projetos, para que, a partir daí sejam incorporadas outras ferramentas para a computação automática das métricas definidas para o experimento, e no caso das métricas relacionadas ao diagrama de classes, como nenhum dos três projetos disponibilizam o mesmo, serão utilizadas ferramentas de engenharia reversa, para fazer a elaboração deste diagrama e em seguida serem coletadas as métricas.

A análise dos dados será baseado através do coeficiente de correlação de pearson, este coeficiente mede o grau de relação entre duas variáveis. A análise será realizada em duas perspectivas, a primeira será entre métrica,

onde vai ser observado a correlação individual de cada métrica dos atributos com cada métrica de testabilidade, com o objetivo de fazer uma identificação mais específica sobre o relacionamento das propriedades OO com a testabilidade. Já a segunda perspectiva, será a análise entre atributos, desta vez mais geral e com o objetivo de validar as hipóteses deste experimento, onde será avaliado a relação entre cada atributo e a característica de qualidade, testabilidade. Para a primeira perspectiva, o cálculo do coeficiente se dará após a coleta dos dados, será colocado todos os valores máximos de cada métrica em um planilha e calculado a correlação entre elas. Já na segunda perspectiva, antes de calcular a correlação, cada atributo será definido pela média dos valores máximos de cada métrica que representa o atributo ou característica, da seguinte forma:

$$\text{MED\_Atributo} = \sum_{i=1} Mi \text{ /NumM,}$$

onde Mi é o valor máximo de cada métrica utilizada para representar o atributo, e NumM é o número total de métricas utilizadas. Para a análise dos coeficientes, será utilizado como base a classificação especificada na Tabela 1, proposta por Mukaka (2012):

0.9 a 1.0	Indica uma correlação muito forte, quase perfeita
07 a 0.9	Positivo ou negativo indica uma correlação forte
0.5 a 0.7	Positivo ou negativo indica uma correlação moderada
0.3 a 0.5	Positivo ou negativo indica uma correlação fraca
0 a 0.3	Positivo ou negativo Indica uma correlação desprezível

Tabela 1 - Interpretação do coeficiente de Pearson

A Tabela 2 mostra as ferramentas utilizadas no experimento, classificando-as de acordo com seu tipo e finalidade.

Ferramenta	Contexto	Finalidade
Google docs	Planejamento	Registrar informações acerca da condução do experimento
CCCC Tools	Coleta de dados	Ferramenta para automatizar a coleta da métrica IFC
Understand	Coleta de dados	Ferramenta para coletar informações gerais do sistema
Eclipse Metrics	Coleta de dados	Ferramenta para coletar métricas de código OO, relacionadas aos principais pilares da OO(Herança, Encapsulamento, Coesão, Acoplamento)
Eclipse IDE	Coleta de dados	Ferramenta para importar o código fonte da aplicação
SDMetrics	Coleta de dados	Ferramenta para automatizar a coleta de métricas do diagrama de classe
Astah Professional	Coleta de dados	Ferramenta para auxiliar a geração de diagrama de classes a partir do código
Umbrello UML	Coleta de dados	Ferramenta para auxiliar a geração de diagrama de classes a

		partir do código
Google planilhas	Armazenamento dos dados	Armazenar e rotular os dados coletados
SPSS Statistics	Análise de resultados	Plotar todos os dados coletados para fazer cálculos estatísticos, entre eles, o cálculo do coeficiente de correlação

Tabela 2 - Ferramentas utilizadas

f. Validação

Durante a execução deste experimento, o experimentador deve-se atentar a alguns pontos que podem vir à ameaçar a validade do estudo. Pensando nisso, foram elencados alguns pontos importantes que podem invalidar o experimento, são eles:

1) Artefatos indisponíveis no projeto: O fato de não ter todos os artefatos disponíveis para a medição pode ser um fator de ameaça à validade
2) Uso de ferramentas de engenharia reversa: O uso de ferramentas de engenharia reversa pode trazer resultados que não sejam muito precisos

ANEXO A – COEFICIENTE DE CORRELAÇÃO ENTRE MÉTRICAS

Correlações

		MED_Simp	MED_Contr	MED_Obs	MED_Testab
MED_Simp	Correlação de Pearson	1	,510	,938	,793
	Sig. (2 extremidades)		,660	,225	,417
	N	3	3	3	3
MED_Contr	Correlação de Pearson	,510	1	,776	,928
	Sig. (2 extremidades)	,660		,434	,242
	N	3	3	3	3
MED_Obs	Correlação de Pearson	,938	,776	1	,955
	Sig. (2 extremidades)	,225	,434		,192
	N	3	3	3	3
MED_Testab	Correlação de Pearson	,793	,928	,955	1
	Sig. (2 extremidades)	,417	,242	,192	
	N	3	3	3	3

ANEXO B – COEFICIENTE DE CORRELAÇÃO ENTRE MÉTRICAS

Correlações

		ENM	REM	AHF	MHF	NOC
ENM	Correlação de Pearson	1	-,284	,284	,998*	-,680
	Sig. (2 extremidades)		,817	,817	,040	,524
	N	3	3	3	3	3
REM	Correlação de Pearson	-,284	1	-1,000**	-,223	,896
	Sig. (2 extremidades)	,817		,000	,857	,293
	N	3	3	3	3	3
AHF	Correlação de Pearson	,284	-1,000**	1	,223	-,896
	Sig. (2 extremidades)	,817	,000		,857	,293
	N	3	3	3	3	3
MHF	Correlação de Pearson	,998*	-,223	,223	1	-,633
	Sig. (2 extremidades)	,040	,857	,857		,564
	N	3	3	3	3	3
NOC	Correlação de Pearson	-,680	,896	-,896	-,633	1
	Sig. (2 extremidades)	,524	,293	,293	,564	
	N	3	3	3	3	3
DIT	Correlação de Pearson	-,955	,554	-,554	-,935	,866
	Sig. (2 extremidades)	,191	,626	,626	,231	,333
	N	3	3	3	3	3
DIT_IMP	Correlação de Pearson	-,849	-,266	,266	-,880	,189
	Sig. (2 extremidades)	,355	,828	,828	,315	,879
	N	3	3	3	3	3
NOC_IMP	Correlação de Pearson	-,246	-,860	,860	-,306	-,544
	Sig. (2 extremidades)	,842	,341	,341	,802	,634
	N	3	3	3	3	3
NOO	Correlação de Pearson	,356	-,997*	,997*	,297	-,927
	Sig. (2 extremidades)	,769	,048	,048	,808	,244
	N	3	3	3	3	3
EMC	Correlação de Pearson	,093	-,981	,981	,030	-,793
	Sig. (2 extremidades)	,941	,124	,124	,981	,417
	N	3	3	3	3	3
EC	Correlação de Pearson	-,419	-,752	,752	-,475	-,381
	Sig. (2 extremidades)	,725	,458	,458	,685	,751
	N	3	3	3	3	3
AC	Correlação de Pearson	-,554	-,641	,641	-,605	-,235
	Sig. (2 extremidades)	,627	,557	,557	,587	,849
	N	3	3	3	3	3

Correlações

		DIT	DIT_IMP	NOC_IMP	NOO	EMC
ENM	Correlação de Pearson	-,955	-,849	-,246	,356	,093
	Sig. (2 extremidades)	,191	,355	,842	,769	,941
	N	3	3	3	3	3
REM	Correlação de Pearson	,554	-,266	-,860	-,997*	-,981
	Sig. (2 extremidades)	,626	,828	,341	,048	,124
	N	3	3	3	3	3
AHF	Correlação de Pearson	-,554	,266	,860	,997*	,981
	Sig. (2 extremidades)	,626	,828	,341	,048	,124
	N	3	3	3	3	3
MHF	Correlação de Pearson	-,935	-,880	-,306	,297	,030
	Sig. (2 extremidades)	,231	,315	,802	,808	,981
	N	3	3	3	3	3
NOC	Correlação de Pearson	,866	,189	-,544	-,927	-,793
	Sig. (2 extremidades)	,333	,879	,634	,244	,417
	N	3	3	3	3	3
DIT	Correlação de Pearson	1	,655	-,052	-,616	-,382
	Sig. (2 extremidades)		,546	,967	,578	,750
	N	3	3	3	3	3
DIT_IMP	Correlação de Pearson	,655	1	,721	,192	,448
	Sig. (2 extremidades)	,546		,487	,877	,704
	N	3	3	3	3	3
NOC_IMP	Correlação de Pearson	-,052	,721	1	,819	,942
	Sig. (2 extremidades)	,967	,487		,390	,217
	N	3	3	3	3	3
NOO	Correlação de Pearson	-,616	,192	,819	1	,964
	Sig. (2 extremidades)	,578	,877	,390		,172
	N	3	3	3	3	3
EMC	Correlação de Pearson	-,382	,448	,942	,964	1
	Sig. (2 extremidades)	,750	,704	,217	,172	
	N	3	3	3	3	3
EC	Correlação de Pearson	,132	,836	,983	,700	,865
	Sig. (2 extremidades)	,916	,370	,117	,507	,334
	N	3	3	3	3	3
AC	Correlação de Pearson	,283	,910	,943	,582	,778
	Sig. (2 extremidades)	,817	,272	,215	,605	,432
	N	3	3	3	3	3

Correlações

		EC	AC	LCOM	CC	NI
ENM	Correlação de Pearson	-,419	-,554	-,260	-,865	-,951
	Sig. (2 extremidades)	,725	,627	,833	,334	,200
	N	3	3	3	3	3
REM	Correlação de Pearson	-,752	-,641	-,852	-,235	-,027
	Sig. (2 extremidades)	,458	,557	,350	,849	,983
	N	3	3	3	3	3
AHF	Correlação de Pearson	,752	,641	,852	,235	,027
	Sig. (2 extremidades)	,458	,557	,350	,849	,983
	N	3	3	3	3	3
MHF	Correlação de Pearson	-,475	-,605	-,319	-,895	-,968
	Sig. (2 extremidades)	,685	,587	,793	,295	,161
	N	3	3	3	3	3
NOC	Correlação de Pearson	-,381	-,235	-,532	,220	,419
	Sig. (2 extremidades)	,751	,849	,643	,859	,725
	N	3	3	3	3	3
DIT	Correlação de Pearson	,132	,283	-,037	,679	,817
	Sig. (2 extremidades)	,916	,817	,976	,525	,391
	N	3	3	3	3	3
DIT_IMP	Correlação de Pearson	,836	,910	,731	,999*	,971
	Sig. (2 extremidades)	,370	,272	,478	,020	,154
	N	3	3	3	3	3
NOC_IMP	Correlação de Pearson	,983	,943	1,000**	,699	,534
	Sig. (2 extremidades)	,117	,215	,009	,508	,642
	N	3	3	3	3	3
NOO	Correlação de Pearson	,700	,582	,810	,161	-,049
	Sig. (2 extremidades)	,507	,605	,399	,897	,969
	N	3	3	3	3	3
EMC	Correlação de Pearson	,865	,778	,938	,419	,220
	Sig. (2 extremidades)	,334	,432	,226	,725	,859
	N	3	3	3	3	3
EC	Correlação de Pearson	1	,988	,986	,818	,679
	Sig. (2 extremidades)		,098	,108	,390	,524
	N	3	3	3	3	3
AC	Correlação de Pearson	,988	1	,948	,897	,784
	Sig. (2 extremidades)	,098		,206	,292	,426
	N	3	3	3	3	3

Correlações

		IFC	TLOC	TNOO	TWMC	NAK
ENM	Correlação de Pearson	-,392	-,651	-,592	-,668	-,666
	Sig. (2 extremidades)	,744	,549	,597	,534	,536
	N	3	3	3	3	3
REM	Correlação de Pearson	-,771	-,543	-,605	-,524	-,526
	Sig. (2 extremidades)	,440	,634	,586	,649	,648
	N	3	3	3	3	3
AHF	Correlação de Pearson	,771	,543	,605	,524	,526
	Sig. (2 extremidades)	,440	,634	,586	,649	,648
	N	3	3	3	3	3
MHF	Correlação de Pearson	-,449	-,697	-,641	-,713	-,712
	Sig. (2 extremidades)	,704	,509	,557	,494	,496
	N	3	3	3	3	3
NOC	Correlação de Pearson	-,408	-,115	-,189	-,092	-,094
	Sig. (2 extremidades)	,732	,927	,879	,942	,940
	N	3	3	3	3	3
DIT	Correlação de Pearson	,103	,397	,327	,419	,417
	Sig. (2 extremidades)	,934	,740	,788	,725	,726
	N	3	3	3	3	3
DIT_IMP	Correlação de Pearson	,819	,954	,929	,961	,960
	Sig. (2 extremidades)	,389	,194	,242	,179	,181
	N	3	3	3	3	3
NOC_IMP	Correlação de Pearson	,988	,896	,927	,885	,886
	Sig. (2 extremidades)	,098	,293	,245	,308	,306
	N	3	3	3	3	3
NOO	Correlação de Pearson	,720	,478	,543	,458	,460
	Sig. (2 extremidades)	,488	,682	,635	,697	,696
	N	3	3	3	3	3
EMC	Correlação de Pearson	,880	,696	,748	,679	,681
	Sig. (2 extremidades)	,315	,510	,462	,525	,523
	N	3	3	3	3	3
EC	Correlação de Pearson	1,000*	,962	,980	,956	,956
	Sig. (2 extremidades)	,019	,176	,128	,191	,189
	N	3	3	3	3	3
AC	Correlação de Pearson	,983	,993	,999*	,990	,990
	Sig. (2 extremidades)	,117	,078	,030	,092	,091
	N	3	3	3	3	3

Correlações

		NTM
ENM	Correlação de Pearson	-,388
	Sig. (2 extremidades)	,746
	N	3
REM	Correlação de Pearson	-,773
	Sig. (2 extremidades)	,437
	N	3
AHF	Correlação de Pearson	,773
	Sig. (2 extremidades)	,437
	N	3
MHF	Correlação de Pearson	-,445
	Sig. (2 extremidades)	,706
	N	3
NOC	Correlação de Pearson	-,412
	Sig. (2 extremidades)	,730
	N	3
DIT	Correlação de Pearson	,099
	Sig. (2 extremidades)	,937
	N	3
DIT_IMP	Correlação de Pearson	,817
	Sig. (2 extremidades)	,391
	N	3
NOC_IMP	Correlação de Pearson	,989
	Sig. (2 extremidades)	,096
	N	3
NOO	Correlação de Pearson	,723
	Sig. (2 extremidades)	,485
	N	3
EMC	Correlação de Pearson	,882
	Sig. (2 extremidades)	,313
	N	3
EC	Correlação de Pearson	,999*
	Sig. (2 extremidades)	,021
	N	3
AC	Correlação de Pearson	,982
	Sig. (2 extremidades)	,120
	N	3

Correlações

		ENM	REM	AHF	MHF	NOC
LCOM	Correlação de Pearson	-,260	-,852	,852	-,319	-,532
	Sig. (2 extremidades)	,833	,350	,350	,793	,643
	N	3	3	3	3	3
CC	Correlação de Pearson	-,865	-,235	,235	-,895	,220
	Sig. (2 extremidades)	,334	,849	,849	,295	,859
	N	3	3	3	3	3
NI	Correlação de Pearson	-,951	-,027	,027	-,968	,419
	Sig. (2 extremidades)	,200	,983	,983	,161	,725
	N	3	3	3	3	3
IFC	Correlação de Pearson	-,392	-,771	,771	-,449	-,408
	Sig. (2 extremidades)	,744	,440	,440	,704	,732
	N	3	3	3	3	3
TLOC	Correlação de Pearson	-,651	-,543	,543	-,697	-,115
	Sig. (2 extremidades)	,549	,634	,634	,509	,927
	N	3	3	3	3	3
TNOO	Correlação de Pearson	-,592	-,605	,605	-,641	-,189
	Sig. (2 extremidades)	,597	,586	,586	,557	,879
	N	3	3	3	3	3
TWMC	Correlação de Pearson	-,668	-,524	,524	-,713	-,092
	Sig. (2 extremidades)	,534	,649	,649	,494	,942
	N	3	3	3	3	3
NAK	Correlação de Pearson	-,666	-,526	,526	-,712	-,094
	Sig. (2 extremidades)	,536	,648	,648	,496	,940
	N	3	3	3	3	3
NTM	Correlação de Pearson	-,388	-,773	,773	-,445	-,412
	Sig. (2 extremidades)	,746	,437	,437	,706	,730
	N	3	3	3	3	3

Correlações

		DIT	DIT_IMP	NOC_IMP	NOO	EMC
LCOM	Correlação de Pearson	-,037	,731	1,000**	,810	,938
	Sig. (2 extremidades)	,976	,478	,009	,399	,226
	N	3	3	3	3	3
CC	Correlação de Pearson	,679	,999*	,699	,161	,419
	Sig. (2 extremidades)	,525	,020	,508	,897	,725
	N	3	3	3	3	3
NI	Correlação de Pearson	,817	,971	,534	-,049	,220
	Sig. (2 extremidades)	,391	,154	,642	,969	,859
	N	3	3	3	3	3
IFC	Correlação de Pearson	,103	,819	,988	,720	,880
	Sig. (2 extremidades)	,934	,389	,098	,488	,315
	N	3	3	3	3	3
TLOC	Correlação de Pearson	,397	,954	,896	,478	,696
	Sig. (2 extremidades)	,740	,194	,293	,682	,510
	N	3	3	3	3	3
TNOO	Correlação de Pearson	,327	,929	,927	,543	,748
	Sig. (2 extremidades)	,788	,242	,245	,635	,462
	N	3	3	3	3	3
TWMC	Correlação de Pearson	,419	,961	,885	,458	,679
	Sig. (2 extremidades)	,725	,179	,308	,697	,525
	N	3	3	3	3	3
NAK	Correlação de Pearson	,417	,960	,886	,460	,681
	Sig. (2 extremidades)	,726	,181	,306	,696	,523
	N	3	3	3	3	3
NTM	Correlação de Pearson	,099	,817	,989	,723	,882
	Sig. (2 extremidades)	,937	,391	,096	,485	,313
	N	3	3	3	3	3

Correlações

		EC	AC	LCOM	CC	NI
LCOM	Correlação de Pearson	,986	,948	1	,709	,546
	Sig. (2 extremidades)	,108	,206		,498	,632
	N	3	3	3	3	3
CC	Correlação de Pearson	,818	,897	,709	1	,978
	Sig. (2 extremidades)	,390	,292	,498		,134
	N	3	3	3	3	3
NI	Correlação de Pearson	,679	,784	,546	,978	1
	Sig. (2 extremidades)	,524	,426	,632	,134	
	N	3	3	3	3	3
IFC	Correlação de Pearson	1,000*	,983	,990	,800	,658
	Sig. (2 extremidades)	,019	,117	,089	,409	,543
	N	3	3	3	3	3
TLOC	Correlação de Pearson	,962	,993	,902	,944	,854
	Sig. (2 extremidades)	,176	,078	,284	,215	,349
	N	3	3	3	3	3
TNOO	Correlação de Pearson	,980	,999*	,932	,916	,812
	Sig. (2 extremidades)	,128	,030	,236	,262	,396
	N	3	3	3	3	3
TWMC	Correlação de Pearson	,956	,990	,892	,951	,866
	Sig. (2 extremidades)	,191	,092	,299	,200	,334
	N	3	3	3	3	3
NAK	Correlação de Pearson	,956	,990	,893	,950	,865
	Sig. (2 extremidades)	,189	,091	,297	,201	,335
	N	3	3	3	3	3
NTM	Correlação de Pearson	,999*	,982	,991	,798	,654
	Sig. (2 extremidades)	,021	,120	,087	,412	,546
	N	3	3	3	3	3

Correlações

		IFC	TLOC	TNOO	TWMC	NAK
LCOM	Correlação de Pearson	,990	,902	,932	,892	,893
	Sig. (2 extremidades)	,089	,284	,236	,299	,297
	N	3	3	3	3	3
CC	Correlação de Pearson	,800	,944	,916	,951	,950
	Sig. (2 extremidades)	,409	,215	,262	,200	,201
	N	3	3	3	3	3
NI	Correlação de Pearson	,658	,854	,812	,866	,865
	Sig. (2 extremidades)	,543	,349	,396	,334	,335
	N	3	3	3	3	3
IFC	Correlação de Pearson	1	,954	,974	,946	,947
	Sig. (2 extremidades)		,195	,147	,209	,208
	N	3	3	3	3	3
TLOC	Correlação de Pearson	,954	1	,997*	1,000*	1,000*
	Sig. (2 extremidades)	,195		,048	,015	,013
	N	3	3	3	3	3
TNOO	Correlação de Pearson	,974	,997*	1	,995	,995
	Sig. (2 extremidades)	,147	,048		,063	,061
	N	3	3	3	3	3
TWMC	Correlação de Pearson	,946	1,000*	,995	1	1,000**
	Sig. (2 extremidades)	,209	,015	,063		,001
	N	3	3	3	3	3
NAK	Correlação de Pearson	,947	1,000*	,995	1,000**	1
	Sig. (2 extremidades)	,208	,013	,061	,001	
	N	3	3	3	3	3
NTM	Correlação de Pearson	1,000**	,952	,973	,945	,946
	Sig. (2 extremidades)	,003	,197	,149	,212	,211
	N	3	3	3	3	3

Correlações

		NTM
LCOM	Correlação de Pearson	,991
	Sig. (2 extremidades)	,087
	N	3
CC	Correlação de Pearson	,798
	Sig. (2 extremidades)	,412
	N	3
NI	Correlação de Pearson	,654
	Sig. (2 extremidades)	,546
	N	3
IFC	Correlação de Pearson	1,000**
	Sig. (2 extremidades)	,003
	N	3
TLOC	Correlação de Pearson	,952
	Sig. (2 extremidades)	,197
	N	3
TNOO	Correlação de Pearson	,973
	Sig. (2 extremidades)	,149
	N	3
TWMC	Correlação de Pearson	,945
	Sig. (2 extremidades)	,212
	N	3
NAK	Correlação de Pearson	,946
	Sig. (2 extremidades)	,211
	N	3
NTM	Correlação de Pearson	1
	Sig. (2 extremidades)	
	N	3

*. A correlação é significativa no nível 0,05 (2 extremidades).

**. A correlação é significativa no nível 0,01 (2 extremidades).

REFERÊNCIAS

BACH, J. **Heuristics of Software Testability**. 1994. Disponível em: newsgroupcomp. software.eng. Acesso em: 28 ago. 2018.

BINDER, R. Design for Testability in Object-Oriented Systems. **Communication of ACM**, v. 37, n. 9, p. 87–100, 1994.

CHIDAMBER, S.; KEMERER, C. A metrics suite for object oriented design. **IEEE Transactions on Software Engineering,** v. 20, n. 6, p. 476–493, jun. 1994.

CRUZ, R. C. da. **Análise empírica sobre a influência das métricas CK na testabilidade de software orientado a objetos**. 2018. Dissertação (Mestrado em Ciências) — Universidade de São Paulo, São Paulo, 2018.

DAVIS, A. 201 **Principles of Software Development**. 4. ed. [S. l.]: McGraw-Hill, 1995. Disponível em: https://books.google.com.br/books?id=8HAhAQAAIAAJ. Acesso em: 24 ago. 2018.

DAVIS, F. D.; REYNOLDS, A. D.; KINCAID. Identifying and measuring quality in a software requirements specification. *In*: INTERNATIONAL SOFTWARE METRICS SYMPOSIUM, 1., 1993, Baltimore. **Proceedings** […]. Baltimore: IEEE, 1993. p. 141–152.

DAYANANDAN, U.; VIVEKANANDAN, K. An empirical evaluation model for software architecture maintainability for object oriented design. *In*: INTERNATIONAL CONFERENCE ON INFORMATICS AND ANALYTICS, 2016, New York. **Proceedings** […]. New York: ACM, 2016. p. 98:1–98:4.

FREEDMAN, R. S. Testability of software components. **IEEE Transactions on Software Engineering**, v. 17, n. 6, p. 553–564, jun. 1991.

GAFFNEY, J. E. Metrics in software quality assurance. *In*: ACM 81 CONFERENCE, 1981, New York. **Proceedings** […]. New York: ACM, 1981. p. 126–130.

GAO, J.; SHIH, M. C. A component testability model for verification and measurement. *In:* ANNUAL INTERNATIONAL COMPUTER SOFTWARE AND APPLICATIONS CONFERENCE, 29., 2005, Edinburgh. **Proceedings** […]. [S.l.]: IEEE, 2005. p. 211–218.

GARVIN, D. Competing on the eight dimensions of quality. **Harvard Business Review 65**, n. 6, p. 101–109, 1987.

HENRY, S.; KAFURA, K. Software structure metrics based on information flow. **IEEE Transactions on Software Engineering**, SE-7, n. 5, p. 510–518, Sep. 1981.

IEEE. Standard for a software quality metrics methodology. **IEEE Std 1061-1992**, p. 1–96, Mar. 1993.

JURECZKO, M.; SPINELLIS, D. Using object-oriented design metrics to predict software defects. *In*: MODELS AND METHODS OF SYSTEM DEPENDABILITY. OFICYNA

WYDAWNICZA POLITECHNIKI WROCLAWSKIEJ, 2010, p. 69–81.

KHALID, S.; ZEHRA, S.; ARIF, F. Analysis of object oriented complexity and testability using object oriented design metrics. *In*: NATIONAL SOFTWARE ENGINEERING CONFERENCE, 10., 2010, New York. **Proceedings** [...]. New York: ACM, 2010. p. 4:1–4:8.

KHAN, R. A.; MUSTAFA, K. Metric based testability model for object oriented design (mtmood). **SIGSOFT Softw. Eng. Notes**, New York, v. 34, n. 2, p. 1–6, fev. 2009.

KOSCIANSKI, A.; SOARES, M. dos S. **Qualidade de Software**: aprenda as metodologias e técnicas mais modernas para o desenvolvimento de software. 2. ed. [S. l.]: Novatec, 2007. Disponível em: https://books.google.com.br/books?id=O9aWoUq6L88C. Acesso em: 05 ago. 2018.

KOUT, A.; TOURE, F.; BADRI, M. An empirical analysis of a testability model for object-oriented programs. **SIGSOFT Softw. Eng. Notes**, New York, v. 36, n. 4, p. 1–5, ago. 2011.

KUDRJAVETS, G.; NAGAPPAN, N.; BALL, T. Assessing the relationship between software assertions and faults: an empirical investigation. *In*: INTERNATIONAL SYMPOSIUM ON SOFTWARE RELIABILITY ENGINEERING, 17., 2006, Raleigh. **Proceedings** […]. [S.l.]: IEEE, 2006. p. 204–212.

LEACH, R. **Introduction to Software Engineering**. 2. ed. Washington: CRC Press, 2016. Disponível em: <https://books.google.com.br/books?id=8W2mCwAAQBAJ>. Acesso em: 20 ago. 2018.

MCCABE, T. J. A complexity measure. **IEEE Transactions on Software Engineering**, SE-2, n. 4, p. 308–320, dec. 1976.

MCCALL, J. A.; RICHARDS, P. K.; WALTERS, G. F. Factors in software quality. **US Rome Air development center reports**, GENERAL ELECTRIC CO SUNNYVALE CA, v. 1, n. 3, p. 77–369, 1977.

MUKAKA, M. Statistics corner: a guide to appropriate use of correlation coefficient in medical research. **Malawi medical journal**, v. 24, p. 69–71, sep. 2012.

MYERS, G. **The Art of Software Testing**. 3. ed. [S.l.]: Wiley, 1979.

NUñEZ-VARELA, A. S.; PéREZ-GONZALEZ, H. G.; MARTíNEZ-PEREZ, F. E.; SOUBERVIELLE-MONTALVO, C. Source code metrics: a systematic mapping study. **Journal of Systems and Software**, v. 128, p. 164 – 197, 2017.

PRESSMAN, R. **Engenharia de Software**. 7. ed. [S.l.]: McGraw Hill Brasil, 2009.

SOMMERVILLE, I. **Engenharia de software**. 9. ed. [S.l.]: PEARSON BRASIL, 2011.

SPEARMAN, C. The proof and measurement of association between two things. **The American Journal of Psychology**, v. 15, n. 1, p. 72–101, 1904.

TI, E. em. **Métricas de Qualidade de Software**. 2016. Disponível em: https://www.tiespecialistas.com.br/metricas-de-qualidade-de-software/. Acesso em: 02 jul. 2018.

TSUI, F.; IRIELE, S. Analysis of software cohesion attribute and test case development complexity. *In*: ANNUAL SOUTHEAST REGIONAL CONFERENCE, 49., 2011, New York. **Proceedings** […]. New York: ACM, 2011. p. 237–242.

TU, H.; SUN, W.; ZHANG, Y. The research on software metrics and software complexity metrics. *In*: INTERNATIONAL FORUM ON COMPUTER SCIENCE TECHNOLOGY AND APPLICATIONS, 2009, Chongqing . **Proceedings** […]. [S.l.]: IEEE, 2009. v. 1, p. 131–136.

APÊNDICE A – TABELAS DE ASSOCIAÇÃO DAS MÉTRICAS

ASSOCIAÇÃO ENTRE MÉTRICAS E ATRIBUTOS

Atributo	Métrica	Cálculo da Métrica	Interpretação
SIMPLICIDADE	ENM	Contagem do número de operações (métodos) da classe	Quanto maior, maior o nível de encapsulamento
SIMPLICIDADE	REM	Contagem do número de hierarquia de classes	Quanto maior, mais dependente a classe será.
SIMPLICIDADE	AHF	Contagem do número de atributos ocultos dividido pelo número de total de atributos	Quanto maior, menos complexo será o código.
SIMPLICIDADE	MHF	Contagem do número de métodos ocultos dividido pelo número de total de métodos	Quanto maior, menos complexo será o código.
SIMPLICIDADE	NOC	Contagem do número de sucessores de uma classe	Quanto maior, mais dependente a classe será.
SIMPLICIDADE	DIT	Contagem do número de antecessores de uma classe	Quanto maior, mais dependente a classe será.
SIMPLICIDADE	NOO	Contagem do número de operações de uma classe	Quanto maior, maior o nível de encapsulamento
SIMPLICIDADE	WMC	Somatório da complexidade de cada método da classe.	Quanto maior, mais complexa será a classe.
SIMPLICIDADE	EC	Média do número de tipos de dados que uma classe conhece	Quanto maior, menos especifica e mais responsabilidade terá o componente
SIMPLICIDADE	AC	Número total de classes externas acopladas a classes de um pacote (usa a definição de acoplamento CBO)	Quanto maior, mais classes externas acopladas terá a classe
SIMPLICIDADE	LCOM	LCOM = P = NP – Q, onde: P é o número de métodos que não compartilham atributos, Q é o número de métodos que compartilham atributos e NP é dado pela função M! / 2! . (M – 2)!, sendo M o número de métodos existentes na classe.	Quanto maior, menos coeso será o código
OBSERVABILIDADE	CC	V(G) = E −N +2, onde G é um grafo, E é a quantidade de arestas e N é o número de vértices	Geralmente, um valor máximo entre 15-10 é aceitável
OBSERVABILIDADE	NI	Contagem do número de interfaces utilizadas por uma dada classe	Quanto maior, mais responsabilidade de implementação terá a classe
CONTROLABILIDADE	IFC	$T * (f\,an{-}in * f\,an{-}out)^2$, fan-in e fan-out relaciona-se à quantidade de informação de I/O de um dado componente e T refere-se ao tamanho do componente.	Quanto maior, maior será a complexidade envolvendo o fluxo de dados.

ASSOCIAÇÃO ENTRE MÉTRICAS E TESTABILIDADE

Característica	Métrica	Cálculo da Métrica	Interpretação
TESTABILIDADE	TLOC	Contagem de linhas de uma classe de teste	Quanto maior, mais grande será o código de teste
TESTABILIDADE	TNOO	Contagem do número de operações das classes de teste	Quanto maior, mais operações terá a classe de teste
TESTABILIDADE	TWMC	Somatório da complexidade de cada método do caso de teste.	Quanto maior, mais complexa será a classe de teste.
TESTABILIDADE	NAK	Contagem do número de *asserts* por KLOC	Quanto maior, mais testes estão sendo feitos por KLOC, por uma classe
TESTABILIDADE	NTM	Contagem do número de métodos de teste	Quanto maior, mais testes estão sendo feitos por KLOC, por uma classe

APÊNDICE B – PLANEJAMENTO EXPERIMENTAL

Estudo sobre atributos que influenciam da testabilidade de software

Planejamento Experimental

Aluno: Francisco Gutenberg da Silva Filho

Orientadora: Valéria Lelli Leitão Dantas

07 de Junho de 2019

Sumário

1. Introdução

A exigência por maior qualidade de software vem crescendo à medida que os sistemas estão mais presentes nas atividades do cotidiano, embora a ideia de qualidade de software seja intuitiva, ao ser analisada minuciosamente, esse conceito se torna complexo. Por ser complexo, muitos autores e normas técnicas propõem modelos, que facilitam a avaliação da qualidade de software. No geral, os modelos de qualidade (e.g., SQUARE 25000) definem um conjunto de características ou fatores que estão diretamente relacionados com a qualidade de software. Uma característica importante presente na maioria desses modelos é a testabilidade, pois ela possui relação com testes de software, que tem como objetivo evidenciar que um programa faz o que é proposto a fazer enquanto testabilidade pode ser definida como a facilidade com que um programa pode ser testado. Dessa forma, avaliar a testabilidade de um software traz informações relevantes sobre a qualidade de um sistema. Porém, a testabilidade não é uma característica intrínseca do software, e, portanto, deve se identificar ou definir medidas que deem uma indicação direta sobre testabilidade. A proposta deste estudo é investigar a testabilidade de software com o intuito de propor um guia que auxilie na avaliação dessa característica. Na primeira etapa do estudo foi feita uma revisão bibliográfica para identificar as características de testabilidade e as métricas que representam essas características. Na segunda etapa, com base na identificação feita na etapa anterior, foi realizado um experimento em sistemas de código aberto, para avaliar a testabilidade desses sistemas. Como resultados, este trabalho apresenta algumas métricas e atributos identificados na literatura para avaliar a testabilidade de software.

2. Planejamento Experimental

a. Seleção do Contexto

Este experimento foi realizado de maneira ad-hoc, com objetivo de avaliar a testabilidade de software por meio dos atributos identificados. Serão selecionados três sistemas de código aberto, e para cada sistema serão coletados dados referente as métricas, que por sua vez determinam cada atributo. A realização deste experimento foi conduzido de maneira automática, utilizando ferramentas que automatizam, principalmente, os procedimentos de coletas e análise de dados. Como resultados do experimento, foram identificadas correlações entre atributos e entre métricas definidas para o experimento.

b. Formulação das hipóteses

i. **Hipótese Nula - H0:** Não existe correlação significativa entre simplicidade e a testabilidade de um software orientado a objetos.

ii. **Hipótese Alternativa - Ha:** Existe correlação significativa entre o simplicidade e a testabilidade de um software orientado a objetos.

iii. **Hipótese Nula - H1:** Não existe correlação significativa entre controlabilidade e a testabilidade de um software orientado a objetos.

iv. **Hipótese Alternativa - Ha1:** Existe correlação significativa entre controlabilidade e a testabilidade de um software orientado a objetos.

v. **Hipótese Nula - H2:** Não existe correlação significativa entre observabilidade e a testabilidade de um software orientado a objetos.

vi. **Hipótese Alternativa - Ha2:** Existe correlação significativa entre observabilidade e a testabilidade de um software orientado a objetos.

c. Seleção das variáveis

i. Dependentes

1. Testabilidade

   Esta característica será definido a partir do seguinte conjunto de métricas/variáveis:

   a. TNOO
   b. TLOC
   c. TWMC
   d. NAK
   e. NTM

ii. Independentes

1. Simplicidade

   Este atributo será definido a partir do seguinte conjunto de métricas/variáveis:

   a. ENM
   b. REM
   c. AHF
   d. MHF
   e. NOC
   f. DIT
   g. NOC
   h. NOO

i. DIT (Implementação)

j. NOC (Implementação)

k. NOO

l. WMC

m. EC

n. AC

o. LCOM

2. Observabilidade

   Este atributo será definido a partir do seguinte conjunto de métricas/variáveis:

   a. IFC
   b. NI

3. Controlabilidade

   Este atributo será definido a partir do seguinte conjunto de métricas/variáveis:

   a. CC

Estas métricas, estão disponíveis no repositório do experimento, na seção **Informações Gerais**. Nela, as métricas são definidas e classificadas.

d. Seleção dos Sistemas

Foram selecionados três projetos de código aberto disponível na plataforma do Github. Estes sistemas foram selecionados a partir de alguns pontos que o autor definiu como critério para seleção, estes pontos foram:

1. O projeto deve ainda estar sendo mantido
2. O projeto deve ser open source
3. O projeto deve ter classes de testes implementadas
4. O projeto deve ter no mínimo 50 classes e no máximo 800.

Com base nestes pontos, foram selecionados três projetos para este experimento, que são descritos a seguir:

i. ***CK-Calculator***

O CK calcula as métricas de código em nível de classe e nível de métrica em projetos Java por meio de análise estática (ou seja, não há necessidade de código compilado). Atualmente, ele contém um grande conjunto de métricas, incluindo o famoso CK.

Informações gerais:

Núm Classes	**Funções**	**Linhas de Código**	**Porte**
124(73 código/51 teste)	1008	9763	Pequeno

ii. ***Joda-Time***

O Joda-Time fornece um substituto de qualidade para as classes de data e hora do Java. O Joda-Time é a biblioteca padrão de data e hora do Java antes do Java SE 8. Agora, os usuários são solicitados a migrar para o java.time (JSR-310).

Informações gerais:

Núm Classes	**Funções**	**Linhas de Código**	**Porte**
306(147 código/159 teste)	8046	144.079	Médio

iii. ***JFreeChart***

O JFreeChart é uma biblioteca de gráficos gratuita abrangente para a plataforma Java (tm) que pode ser usada no lado do cliente (JavaFX e Swing) ou no lado do servidor (com exportação para vários formatos, incluindo SVG, PNG e PDF).

Informações gerais:

Núm Classes	**Funções**	**Linhas de Código**	**Porte**
657(283 código/374 teste)	9369	279.106	Grande

e. Instrumentação

Para a realização deste experimento, foram utilizados algumas ferramentas que automatizam alguns procedimentos, com o objetivo de poupar tempo em relação a execução destes procedimentos caso fossem feitos manualmente. Estas ferramentas se aplicam à etapas diferentes do experimento, sendo elas:

1) Coleta dos dados
2) Análise de resultados

Para a realização da coleta de dados, serão utilizadas ferramentas para importação do código fonte dos projetos, para que, a partir daí sejam incorporadas outras ferramentas para a computação automática das métricas definidas para o experimento, e no caso das métricas relacionadas ao diagrama de classes, como nenhum dos três projetos disponibilizam o mesmo, serão utilizadas ferramentas de engenharia reversa, para fazer a elaboração deste diagrama e em seguida serem coletadas as métricas.

A análise dos dados será baseado através do coeficiente de correlação de pearson, este coeficiente mede o grau de relação entre duas variáveis. A análise será realizada em duas perspectivas, a primeira será entre métrica,

onde vai ser observado a correlação individual de cada métrica dos atributos com cada métrica de testabilidade, com o objetivo de fazer uma identificação mais específica sobre o relacionamento das propriedades OO com a testabilidade. Já a segunda perspectiva, será a análise entre atributos, desta vez mais geral e com o objetivo de validar as hipóteses deste experimento, onde será avaliado a relação entre cada atributo e a característica de qualidade, testabilidade. Para a primeira perspectiva, o cálculo do coeficiente se dará após a coleta dos dados, será colocado todos os valores máximos de cada métrica em um planilha e calculado a correlação entre elas. Já na segunda perspectiva, antes de calcular a correlação, cada atributo será definido pela média dos valores máximos de cada métrica que representa o atributo ou característica, da seguinte forma:

$$\text{MED\_Atributo} = \sum_{i=1} Mi \text{ /NumM,}$$

onde Mi é o valor máximo de cada métrica utilizada para representar o atributo, e NumM é o número total de métricas utilizadas. Para a análise dos coeficientes, será utilizado como base a classificação especificada na Tabela 1, proposta por Mukaka (2012):

0.9 a 1.0	Indica uma correlação muito forte, quase perfeita
07 a 0.9	Positivo ou negativo indica uma correlação forte
0.5 a 0.7	Positivo ou negativo indica uma correlação moderada
0.3 a 0.5	Positivo ou negativo indica uma correlação fraca
0 a 0.3	Positivo ou negativo Indica uma correlação desprezível

Tabela 1 - Interpretação do coeficiente de Pearson

A Tabela 2 mostra as ferramentas utilizadas no experimento, classificando-as de acordo com seu tipo e finalidade.

Ferramenta	Contexto	Finalidade
Google docs	Planejamento	Registrar informações acerca da condução do experimento
CCCC Tools	Coleta de dados	Ferramenta para automatizar a coleta da métrica IFC
Understand	Coleta de dados	Ferramenta para coletar informações gerais do sistema
Eclipse Metrics	Coleta de dados	Ferramenta para coletar métricas de código OO, relacionadas aos principais pilares da OO(Herança, Encapsulamento, Coesão, Acoplamento)
Eclipse IDE	Coleta de dados	Ferramenta para importar o código fonte da aplicação
SDMetrics	Coleta de dados	Ferramenta para automatizar a coleta de métricas do diagrama de classe
Astah Professional	Coleta de dados	Ferramenta para auxiliar a geração de diagrama de classes a partir do código
Umbrello UML	Coleta de dados	Ferramenta para auxiliar a geração de diagrama de classes a

		partir do código
Google planilhas	Armazenamento dos dados	Armazenar e rotular os dados coletados
SPSS Statistics	Análise de resultados	Plotar todos os dados coletados para fazer cálculos estatísticos, entre eles, o cálculo do coeficiente de correlação

Tabela 2 - Ferramentas utilizadas

f. Validação

Durante a execução deste experimento, o experimentador deve-se atentar a alguns pontos que podem vir à ameaçar a validade do estudo. Pensando nisso, foram elencados alguns pontos importantes que podem invalidar o experimento, são eles:

1) Artefatos indisponíveis no projeto: O fato de não ter todos os artefatos disponíveis para a medição pode ser um fator de ameaça à validade
2) Uso de ferramentas de engenharia reversa: O uso de ferramentas de engenharia reversa pode trazer resultados que não sejam muito precisos

ANEXO A – COEFICIENTE DE CORRELAÇÃO ENTRE MÉTRICAS

Correlações

		MED_Simp	MED_Contr	MED_Obs	MED_Testab
MED_Simp	Correlação de Pearson	1	,510	,938	,793
	Sig. (2 extremidades)		,660	,225	,417
	N	3	3	3	3
MED_Contr	Correlação de Pearson	,510	1	,776	,928
	Sig. (2 extremidades)	,660		,434	,242
	N	3	3	3	3
MED_Obs	Correlação de Pearson	,938	,776	1	,955
	Sig. (2 extremidades)	,225	,434		,192
	N	3	3	3	3
MED_Testab	Correlação de Pearson	,793	,928	,955	1
	Sig. (2 extremidades)	,417	,242	,192	
	N	3	3	3	3

ANEXO B – COEFICIENTE DE CORRELAÇÃO ENTRE MÉTRICAS

Correlações

		ENM	REM	AHF	MHF	NOC
ENM	Correlação de Pearson	1	-,284	,284	,998*	-,680
	Sig. (2 extremidades)		,817	,817	,040	,524
	N	3	3	3	3	3
REM	Correlação de Pearson	-,284	1	-1,000**	-,223	,896
	Sig. (2 extremidades)	,817		,000	,857	,293
	N	3	3	3	3	3
AHF	Correlação de Pearson	,284	-1,000**	1	,223	-,896
	Sig. (2 extremidades)	,817	,000		,857	,293
	N	3	3	3	3	3
MHF	Correlação de Pearson	,998*	-,223	,223	1	-,633
	Sig. (2 extremidades)	,040	,857	,857		,564
	N	3	3	3	3	3
NOC	Correlação de Pearson	-,680	,896	-,896	-,633	1
	Sig. (2 extremidades)	,524	,293	,293	,564	
	N	3	3	3	3	3
DIT	Correlação de Pearson	-,955	,554	-,554	-,935	,866
	Sig. (2 extremidades)	,191	,626	,626	,231	,333
	N	3	3	3	3	3
DIT_IMP	Correlação de Pearson	-,849	-,266	,266	-,880	,189
	Sig. (2 extremidades)	,355	,828	,828	,315	,879
	N	3	3	3	3	3
NOC_IMP	Correlação de Pearson	-,246	-,860	,860	-,306	-,544
	Sig. (2 extremidades)	,842	,341	,341	,802	,634
	N	3	3	3	3	3
NOO	Correlação de Pearson	,356	-,997*	,997*	,297	-,927
	Sig. (2 extremidades)	,769	,048	,048	,808	,244
	N	3	3	3	3	3
EMC	Correlação de Pearson	,093	-,981	,981	,030	-,793
	Sig. (2 extremidades)	,941	,124	,124	,981	,417
	N	3	3	3	3	3
EC	Correlação de Pearson	-,419	-,752	,752	-,475	-,381
	Sig. (2 extremidades)	,725	,458	,458	,685	,751
	N	3	3	3	3	3
AC	Correlação de Pearson	-,554	-,641	,641	-,605	-,235
	Sig. (2 extremidades)	,627	,557	,557	,587	,849
	N	3	3	3	3	3

Correlações

		DIT	DIT_IMP	NOC_IMP	NOO	EMC
ENM	Correlação de Pearson	-,955	-,849	-,246	,356	,093
	Sig. (2 extremidades)	,191	,355	,842	,769	,941
	N	3	3	3	3	3
REM	Correlação de Pearson	,554	-,266	-,860	-,997*	-,981
	Sig. (2 extremidades)	,626	,828	,341	,048	,124
	N	3	3	3	3	3
AHF	Correlação de Pearson	-,554	,266	,860	,997*	,981
	Sig. (2 extremidades)	,626	,828	,341	,048	,124
	N	3	3	3	3	3
MHF	Correlação de Pearson	-,935	-,880	-,306	,297	,030
	Sig. (2 extremidades)	,231	,315	,802	,808	,981
	N	3	3	3	3	3
NOC	Correlação de Pearson	,866	,189	-,544	-,927	-,793
	Sig. (2 extremidades)	,333	,879	,634	,244	,417
	N	3	3	3	3	3
DIT	Correlação de Pearson	1	,655	-,052	-,616	-,382
	Sig. (2 extremidades)		,546	,967	,578	,750
	N	3	3	3	3	3
DIT_IMP	Correlação de Pearson	,655	1	,721	,192	,448
	Sig. (2 extremidades)	,546		,487	,877	,704
	N	3	3	3	3	3
NOC_IMP	Correlação de Pearson	-,052	,721	1	,819	,942
	Sig. (2 extremidades)	,967	,487		,390	,217
	N	3	3	3	3	3
NOO	Correlação de Pearson	-,616	,192	,819	1	,964
	Sig. (2 extremidades)	,578	,877	,390		,172
	N	3	3	3	3	3
EMC	Correlação de Pearson	-,382	,448	,942	,964	1
	Sig. (2 extremidades)	,750	,704	,217	,172	
	N	3	3	3	3	3
EC	Correlação de Pearson	,132	,836	,983	,700	,865
	Sig. (2 extremidades)	,916	,370	,117	,507	,334
	N	3	3	3	3	3
AC	Correlação de Pearson	,283	,910	,943	,582	,778
	Sig. (2 extremidades)	,817	,272	,215	,605	,432
	N	3	3	3	3	3

Correlações

		EC	AC	LCOM	CC	NI
ENM	Correlação de Pearson	-,419	-,554	-,260	-,865	-,951
	Sig. (2 extremidades)	,725	,627	,833	,334	,200
	N	3	3	3	3	3
REM	Correlação de Pearson	-,752	-,641	-,852	-,235	-,027
	Sig. (2 extremidades)	,458	,557	,350	,849	,983
	N	3	3	3	3	3
AHF	Correlação de Pearson	,752	,641	,852	,235	,027
	Sig. (2 extremidades)	,458	,557	,350	,849	,983
	N	3	3	3	3	3
MHF	Correlação de Pearson	-,475	-,605	-,319	-,895	-,968
	Sig. (2 extremidades)	,685	,587	,793	,295	,161
	N	3	3	3	3	3
NOC	Correlação de Pearson	-,381	-,235	-,532	,220	,419
	Sig. (2 extremidades)	,751	,849	,643	,859	,725
	N	3	3	3	3	3
DIT	Correlação de Pearson	,132	,283	-,037	,679	,817
	Sig. (2 extremidades)	,916	,817	,976	,525	,391
	N	3	3	3	3	3
DIT_IMP	Correlação de Pearson	,836	,910	,731	,999*	,971
	Sig. (2 extremidades)	,370	,272	,478	,020	,154
	N	3	3	3	3	3
NOC_IMP	Correlação de Pearson	,983	,943	1,000**	,699	,534
	Sig. (2 extremidades)	,117	,215	,009	,508	,642
	N	3	3	3	3	3
NOO	Correlação de Pearson	,700	,582	,810	,161	-,049
	Sig. (2 extremidades)	,507	,605	,399	,897	,969
	N	3	3	3	3	3
EMC	Correlação de Pearson	,865	,778	,938	,419	,220
	Sig. (2 extremidades)	,334	,432	,226	,725	,859
	N	3	3	3	3	3
EC	Correlação de Pearson	1	,988	,986	,818	,679
	Sig. (2 extremidades)		,098	,108	,390	,524
	N	3	3	3	3	3
AC	Correlação de Pearson	,988	1	,948	,897	,784
	Sig. (2 extremidades)	,098		,206	,292	,426
	N	3	3	3	3	3

Correlações

		IFC	TLOC	TNOO	TWMC	NAK
ENM	Correlação de Pearson	-,392	-,651	-,592	-,668	-,666
	Sig. (2 extremidades)	,744	,549	,597	,534	,536
	N	3	3	3	3	3
REM	Correlação de Pearson	-,771	-,543	-,605	-,524	-,526
	Sig. (2 extremidades)	,440	,634	,586	,649	,648
	N	3	3	3	3	3
AHF	Correlação de Pearson	,771	,543	,605	,524	,526
	Sig. (2 extremidades)	,440	,634	,586	,649	,648
	N	3	3	3	3	3
MHF	Correlação de Pearson	-,449	-,697	-,641	-,713	-,712
	Sig. (2 extremidades)	,704	,509	,557	,494	,496
	N	3	3	3	3	3
NOC	Correlação de Pearson	-,408	-,115	-,189	-,092	-,094
	Sig. (2 extremidades)	,732	,927	,879	,942	,940
	N	3	3	3	3	3
DIT	Correlação de Pearson	,103	,397	,327	,419	,417
	Sig. (2 extremidades)	,934	,740	,788	,725	,726
	N	3	3	3	3	3
DIT_IMP	Correlação de Pearson	,819	,954	,929	,961	,960
	Sig. (2 extremidades)	,389	,194	,242	,179	,181
	N	3	3	3	3	3
NOC_IMP	Correlação de Pearson	,988	,896	,927	,885	,886
	Sig. (2 extremidades)	,098	,293	,245	,308	,306
	N	3	3	3	3	3
NOO	Correlação de Pearson	,720	,478	,543	,458	,460
	Sig. (2 extremidades)	,488	,682	,635	,697	,696
	N	3	3	3	3	3
EMC	Correlação de Pearson	,880	,696	,748	,679	,681
	Sig. (2 extremidades)	,315	,510	,462	,525	,523
	N	3	3	3	3	3
EC	Correlação de Pearson	1,000*	,962	,980	,956	,956
	Sig. (2 extremidades)	,019	,176	,128	,191	,189
	N	3	3	3	3	3
AC	Correlação de Pearson	,983	,993	,999*	,990	,990
	Sig. (2 extremidades)	,117	,078	,030	,092	,091
	N	3	3	3	3	3

Correlações

		NTM
ENM	Correlação de Pearson	-,388
	Sig. (2 extremidades)	,746
	N	3
REM	Correlação de Pearson	-,773
	Sig. (2 extremidades)	,437
	N	3
AHF	Correlação de Pearson	,773
	Sig. (2 extremidades)	,437
	N	3
MHF	Correlação de Pearson	-,445
	Sig. (2 extremidades)	,706
	N	3
NOC	Correlação de Pearson	-,412
	Sig. (2 extremidades)	,730
	N	3
DIT	Correlação de Pearson	,099
	Sig. (2 extremidades)	,937
	N	3
DIT_IMP	Correlação de Pearson	,817
	Sig. (2 extremidades)	,391
	N	3
NOC_IMP	Correlação de Pearson	,989
	Sig. (2 extremidades)	,096
	N	3
NOO	Correlação de Pearson	,723
	Sig. (2 extremidades)	,485
	N	3
EMC	Correlação de Pearson	,882
	Sig. (2 extremidades)	,313
	N	3
EC	Correlação de Pearson	,999*
	Sig. (2 extremidades)	,021
	N	3
AC	Correlação de Pearson	,982
	Sig. (2 extremidades)	,120
	N	3

Correlações

		ENM	REM	AHF	MHF	NOC
LCOM	Correlação de Pearson	-,260	-,852	,852	-,319	-,532
	Sig. (2 extremidades)	,833	,350	,350	,793	,643
	N	3	3	3	3	3
CC	Correlação de Pearson	-,865	-,235	,235	-,895	,220
	Sig. (2 extremidades)	,334	,849	,849	,295	,859
	N	3	3	3	3	3
NI	Correlação de Pearson	-,951	-,027	,027	-,968	,419
	Sig. (2 extremidades)	,200	,983	,983	,161	,725
	N	3	3	3	3	3
IFC	Correlação de Pearson	-,392	-,771	,771	-,449	-,408
	Sig. (2 extremidades)	,744	,440	,440	,704	,732
	N	3	3	3	3	3
TLOC	Correlação de Pearson	-,651	-,543	,543	-,697	-,115
	Sig. (2 extremidades)	,549	,634	,634	,509	,927
	N	3	3	3	3	3
TNOO	Correlação de Pearson	-,592	-,605	,605	-,641	-,189
	Sig. (2 extremidades)	,597	,586	,586	,557	,879
	N	3	3	3	3	3
TWMC	Correlação de Pearson	-,668	-,524	,524	-,713	-,092
	Sig. (2 extremidades)	,534	,649	,649	,494	,942
	N	3	3	3	3	3
NAK	Correlação de Pearson	-,666	-,526	,526	-,712	-,094
	Sig. (2 extremidades)	,536	,648	,648	,496	,940
	N	3	3	3	3	3
NTM	Correlação de Pearson	-,388	-,773	,773	-,445	-,412
	Sig. (2 extremidades)	,746	,437	,437	,706	,730
	N	3	3	3	3	3

Correlações

		DIT	DIT_IMP	NOC_IMP	NOO	EMC
LCOM	Correlação de Pearson	-,037	,731	1,000**	,810	,938
	Sig. (2 extremidades)	,976	,478	,009	,399	,226
	N	3	3	3	3	3
CC	Correlação de Pearson	,679	,999*	,699	,161	,419
	Sig. (2 extremidades)	,525	,020	,508	,897	,725
	N	3	3	3	3	3
NI	Correlação de Pearson	,817	,971	,534	-,049	,220
	Sig. (2 extremidades)	,391	,154	,642	,969	,859
	N	3	3	3	3	3
IFC	Correlação de Pearson	,103	,819	,988	,720	,880
	Sig. (2 extremidades)	,934	,389	,098	,488	,315
	N	3	3	3	3	3
TLOC	Correlação de Pearson	,397	,954	,896	,478	,696
	Sig. (2 extremidades)	,740	,194	,293	,682	,510
	N	3	3	3	3	3
TNOO	Correlação de Pearson	,327	,929	,927	,543	,748
	Sig. (2 extremidades)	,788	,242	,245	,635	,462
	N	3	3	3	3	3
TWMC	Correlação de Pearson	,419	,961	,885	,458	,679
	Sig. (2 extremidades)	,725	,179	,308	,697	,525
	N	3	3	3	3	3
NAK	Correlação de Pearson	,417	,960	,886	,460	,681
	Sig. (2 extremidades)	,726	,181	,306	,696	,523
	N	3	3	3	3	3
NTM	Correlação de Pearson	,099	,817	,989	,723	,882
	Sig. (2 extremidades)	,937	,391	,096	,485	,313
	N	3	3	3	3	3

Correlações

		EC	AC	LCOM	CC	NI
LCOM	Correlação de Pearson	,986	,948	1	,709	,546
	Sig. (2 extremidades)	,108	,206		,498	,632
	N	3	3	3	3	3
CC	Correlação de Pearson	,818	,897	,709	1	,978
	Sig. (2 extremidades)	,390	,292	,498		,134
	N	3	3	3	3	3
NI	Correlação de Pearson	,679	,784	,546	,978	1
	Sig. (2 extremidades)	,524	,426	,632	,134	
	N	3	3	3	3	3
IFC	Correlação de Pearson	1,000*	,983	,990	,800	,658
	Sig. (2 extremidades)	,019	,117	,089	,409	,543
	N	3	3	3	3	3
TLOC	Correlação de Pearson	,962	,993	,902	,944	,854
	Sig. (2 extremidades)	,176	,078	,284	,215	,349
	N	3	3	3	3	3
TNOO	Correlação de Pearson	,980	,999*	,932	,916	,812
	Sig. (2 extremidades)	,128	,030	,236	,262	,396
	N	3	3	3	3	3
TWMC	Correlação de Pearson	,956	,990	,892	,951	,866
	Sig. (2 extremidades)	,191	,092	,299	,200	,334
	N	3	3	3	3	3
NAK	Correlação de Pearson	,956	,990	,893	,950	,865
	Sig. (2 extremidades)	,189	,091	,297	,201	,335
	N	3	3	3	3	3
NTM	Correlação de Pearson	,999*	,982	,991	,798	,654
	Sig. (2 extremidades)	,021	,120	,087	,412	,546
	N	3	3	3	3	3

Correlações

		IFC	TLOC	TNOO	TWMC	NAK
LCOM	Correlação de Pearson	,990	,902	,932	,892	,893
	Sig. (2 extremidades)	,089	,284	,236	,299	,297
	N	3	3	3	3	3
CC	Correlação de Pearson	,800	,944	,916	,951	,950
	Sig. (2 extremidades)	,409	,215	,262	,200	,201
	N	3	3	3	3	3
NI	Correlação de Pearson	,658	,854	,812	,866	,865
	Sig. (2 extremidades)	,543	,349	,396	,334	,335
	N	3	3	3	3	3
IFC	Correlação de Pearson	1	,954	,974	,946	,947
	Sig. (2 extremidades)		,195	,147	,209	,208
	N	3	3	3	3	3
TLOC	Correlação de Pearson	,954	1	,997*	1,000*	1,000*
	Sig. (2 extremidades)	,195		,048	,015	,013
	N	3	3	3	3	3
TNOO	Correlação de Pearson	,974	,997*	1	,995	,995
	Sig. (2 extremidades)	,147	,048		,063	,061
	N	3	3	3	3	3
TWMC	Correlação de Pearson	,946	1,000*	,995	1	1,000**
	Sig. (2 extremidades)	,209	,015	,063		,001
	N	3	3	3	3	3
NAK	Correlação de Pearson	,947	1,000*	,995	1,000**	1
	Sig. (2 extremidades)	,208	,013	,061	,001	
	N	3	3	3	3	3
NTM	Correlação de Pearson	1,000**	,952	,973	,945	,946
	Sig. (2 extremidades)	,003	,197	,149	,212	,211
	N	3	3	3	3	3

Correlações

		NTM
LCOM	Correlação de Pearson	,991
	Sig. (2 extremidades)	,087
	N	3
CC	Correlação de Pearson	,798
	Sig. (2 extremidades)	,412
	N	3
NI	Correlação de Pearson	,654
	Sig. (2 extremidades)	,546
	N	3
IFC	Correlação de Pearson	1,000**
	Sig. (2 extremidades)	,003
	N	3
TLOC	Correlação de Pearson	,952
	Sig. (2 extremidades)	,197
	N	3
TNOO	Correlação de Pearson	,973
	Sig. (2 extremidades)	,149
	N	3
TWMC	Correlação de Pearson	,945
	Sig. (2 extremidades)	,212
	N	3
NAK	Correlação de Pearson	,946
	Sig. (2 extremidades)	,211
	N	3
NTM	Correlação de Pearson	1
	Sig. (2 extremidades)	
	N	3

*. A correlação é significativa no nível 0,05 (2 extremidades).

**. A correlação é significativa no nível 0,01 (2 extremidades).