RUAN MARCOS BASILIO

FERRAMENTAS DE ANÁLISE ESTÁTICA DE SOFTWARE: UMA ANÁLISE COMPARATIVA

LAVRAS – MG

2022

RUAN MARCOS BASILIO

FERRAMENTAS DE ANÁLISE ESTÁTICA DE SOFTWARE: UMA ANÁLISE COMPARATIVA

Trabalho de Conclusão de Curso apresentado ao Colegiado do Curso de Ciência da Computação, para a obtenção do título de Bacharel.

Orientador
Dr. Antônio Maria Pereira de Resende
(DCC/UFLA)

LAVRAS – MG

2022

RUAN MARCOS BASILIO

FERRAMENTAS DE ANÁLISE ESTÁTICA DE SOFTWARE: UMA ANÁLISE COMPARATIVA

SOFTWARE STATIC ANALYSIS TOOLS: A COMPARATIVE ANALYSIS

Trabalho de Conclusão de Curso apresentado ao Colegiado do Curso de Ciência da Computação, para a obtenção do título de Bacharel.

APROVADO em 27 de abril de 2022.

Dr. Rafael Serapilha Durelli	DCC/UFLA
Bel. Edney Pereira Pinto	FUNDECC/UFLA

Dr. Antônio Maria Pereira de Resende
(Orientador - DCC/UFLA)

LAVRAS – MG

2022

RESUMO

Atualmente, há diversas ferramentas como Sonarqube, Embold, Coverity, Checkmarx, Klocwork, Veracode Application Security Plataform e Kiuwan Code Security & Insights, no mercado, capazes de extrair automaticamente análises de projetos de software. Entretanto, historicamente, tais ferramentas possuem distinções entre suas métricas, suas definições e até mesmo nas implementações dos cálculos realizados na medição. Isto impossibilita comparações das medidas quando elas são analisadas com ferramentas distintas, ainda que se compare a mesma métrica. O objetivo desse trabalho é analisar comparativamente as ferramentas de análise, aplicando-as em projetos de software e verificando se elas apresentam resultados divergentes, buscando justificar, caso haja, a diferença entre as medições. Espera-se que seja possível identificar o que causa a diferenciação ou a igualdade entre as métricas das diferentes ferramentas analisadas.

Palavras-chave: Métricas. Software. Análise. Ferramentas. Comparação.

ABSTRACT

Currently, there are several tools such as Sonarqube, Embold, Coverity, Checkmarx, Klocwork, Veracode Application Security Platform and Kiuwan Code Security & Insights, on the market, capable of automatically extracting analyzes from software projects. However, historically, such tools have distinctions between their metrics, their definitions and even in the implementations of the calculations performed in the measurement. This makes it impossible to compare the measures when they are analyzed with different tools, even when comparing the same metric. The objective of this work is to comparatively analyze the analysis tools, applying them in software projects and verifying if they present divergent results, seeking to justify, if any, the difference between the measurements. It is expected that it will be possible to identify what causes the differentiation or equality between the metrics of the different analyzed tools.

Keywords: Metrics. Software. Analyze. Tools. Comparison.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE SIGLAS

LOC - Lines of Code

LOCM - Lines of Comments

CC - Cyclomatic Complexity

CR - Comment Ratio

SÚMARIO

Ferramentas de Análise Estática de Software: Uma Análise Comparativa

Ruan Marcos Basilio, Antônio Maria

Departamento de Ciência da Computação
Universidade Federal de Lavras (UFLA) – Lavras, MG – Brasil

rmb@estudante.ufla.br, tonio@ufla.br

Abstract. *Currently, there are several tools such as Sonarqube, Embold, Coverity, Checkmarx, Klocwork, Veracode Application Security Platform and Kiuwan Code Security & Insights, on the market, capable of automatically extracting analyzes from software projects. However, historically, such tools have distinctions between their metrics, their definitions and even in the implementations of the calculations performed in the measurement. This makes it impossible to compare the measures when they are analyzed with different tools, even when comparing the same metric. The objective of this work is to comparatively analyze the analysis tools, applying them in software projects and verifying if they present divergent results, seeking to justify, if any, the difference between the measurements. It is expected that it will be possible to identify what causes the differentiation or equality between the metrics of the different analyzed tools.*

Keywords: Metrics. Software. Analyze. Tools. Comparison.

Resumo. *Atualmente, há diversas ferramentas como Sonarqube, Embold, Coverity, Checkmarx, Klocwork, Veracode Application Security Plataform e Kiuwan Code Security & Insights, no mercado, capazes de extrair automaticamente análises de projetos de software. Entretanto, historicamente, tais ferramentas possuem distinções entre suas métricas, suas definições e até mesmo nas implementações dos cálculos realizados na medição. Isto impossibilita comparações das medidas quando elas são analisadas com ferramentas distintas, ainda que se compare a mesma métrica. O objetivo desse trabalho é analisar comparativamente as ferramentas de análise, aplicando-as em projetos de software e verificando se elas apresentam resultados divergentes, buscando justificar, caso haja, a diferença entre as medições. Espera-se que seja possível identificar o que causa a diferenciação ou a igualdade entre as métricas das diferentes ferramentas analisadas.*

Palavras-chave: Métricas. Software. Análise. Ferramentas. Comparação.

1. Introdução

Segundo Sommerville e Ian (2007, p. 3) “a engenharia de software é um ramo da engenharia cujo foco é o desenvolvimento dentro de custos adequados de sistemas de software de alta qualidade”. Com o crescimento da engenharia de software as aplicações começaram a se tornar cada vez maiores e com maior grau de complexidade, as linhas de códigos foram crescendo, maior número de funções, arquivos, entre outros artefatos que constituem um software. Desta forma, surgiu a necessidade de medição, seja da qualidade ou complexidade dos softwares, dando origem às métricas de software.

“As métricas de software são uma forma de medir e avaliar a qualidade e/ou complexidade de um software. As métricas constituem uma boa maneira para entender, monitorar, controlar, prever e testar softwares em desenvolvimento e manutenção de

projetos" (TOMAS; ESCALONA; MEJIAS, 2013, p. 245). Como lembra Pádua (2009), as métricas de software e seu gerenciamento tem objetivos como aumentar a precisão nas previsões dos custos e prazos do projeto e redução dos defeitos do produto.

Tendo em vista que as métricas são uma forma de medir o software, muitas delas feitas a partir de diversos cálculos com base no conteúdo dos arquivos, fica inviável a realização desta tarefa por um humano, pois, levaria muito tempo para finalizar e estaria mais suscetível ao erro, além de não possuir integração direta com o código sendo necessário refazer manualmente a cada modificação no software. As ferramentas de medição de softwares foram criadas para realizar essa tarefa, de forma eficaz e confiável.

Com isso temos diversas ferramentas de análise de métricas disponíveis no mercado, podendo ser gratuitas ou pagas, cada uma com suas respectivas métricas e definições de análise. Desta forma tais ferramentas irão possuir métricas distintas, definições distintas para suas métricas em comum e, até mesmo, diferentes implementações do cálculo da métrica, impossibilitando uma comparação direta das medidas com ferramentas distintas.

Esse trabalho tem como objetivo comparar e analisar diferentes ferramentas, aplicando-as nos mesmos projetos, a fim de, verificar se elas apresentam diferentes resultados para as métricas em comum, buscando justificar a diferença entre as medidas. Espera-se identificar a causa das desigualdades dos resultados, caso haja.

O restante deste artigo está estruturado da seguinte forma: a seção 2 contém os elementos teóricos necessários à compreensão do artigo. A seção 3 apresenta a metodologia utilizada para o desenvolvimento do trabalho. A seção 4, mostra o passo a passo do desenvolvimento do trabalho. A seção 5, com a discussão do resultado. Finalmente, a seção 6, apresenta as conclusões da análise comparativa realizada.

2. Referencial Teórico

O referencial teórico do presente trabalho foi estruturado em três tópicos, a saber: engenharia de software, métricas na engenharia de software e ferramentas de medição de métricas.

2.1. Engenharia de software

> A engenharia de software descreve o conjunto de técnicas que aplicam uma abordagem de engenharia para a construção e suporte de produtos de software. As atividades de engenharia de software incluem gerenciamento, custos, planejamento, modelagem analisando, especificando, projetando, implementando, testando e mantendo. Por "abordagem de engenharia", queremos dizer que cada atividade é compreendida e controlada, para que haja poucas surpresas à medida que o software é especificado, projetado, construído, e mantido (FENTON; BIEMAN, 2015, p. 11).

"As disciplinas de engenharia usam métodos que são sustentados por modelos e teorias. A base do processo é a medição: medir as variáveis para diferenciar casos, medindo as mudanças de comportamento e medindo as causas e efeitos" (FENTON; BIEMAN, 2015, p. 12). Assim como em outras engenharias, a medição é importante na

engenharia de software indicando como serão os resultados. Sem medições diversos problemas podem ocorrer no desenvolvimento de um software. Dessa forma, a realização da medição para a área da engenharia de software, faz com que, de fato, a engenharia de software seja eficaz.

2.2. Métricas na engenharia de software

De acordo com Rosenberg e Sheppard (1994, p. 10), "métricas são as ferramentas de avaliação aplicadas aos dados brutos". Métrica, em sua definição, é uma interpretação de uma ou mais medidas obtidas. Conforme Henderson-Sellers (1999, p. 1) "As métricas nos ajudam a entender o processo e, uma vez entendido, começar a controlá-lo".

Segundo Rosenberg e Sheppard (1994, p. 10), "a medição é o processo usado para coletar informações sobre o processo e o produto". Como lembra Fenton e Bieman (2015), a medição de software é um componente essencial para uma boa prática da engenharia de software, tornando possível desenvolvedores de software medir as características de seus softwares para saber se os requisitos estão consistentes e completos, e se o código está pronto para ser lançado.

Métrica de software é o processo que obtém, através da medição de software, um resultado que reflete o estado e/ou condição em que se encontra o software naquele determinado momento.

> O uso de métricas tem representado uma ferramenta essencial à gerência de projetos de software. Muitos engenheiros de software medem as características do mesmo, a fim de obter uma visão clara de uma série de aspectos, tais como, o atendimento dos produtos de software aos requisitos especificados, desempenho do processo de desenvolvimento, esforço/custo despendido, entre outros (VASCONCELOS et al, 2006, p. 105).

Segundo Meirelles (2013, p. 14), "métricas de software podem ser classificadas quanto ao âmbito da sua aplicação, quanto ao critério utilizado na sua definição e quanto ao método de obtenção da medida".

2.3. Ferramentas de medição de métricas

As ferramentas de medição de métricas, realizam o cálculo das métricas de software, desta forma, é possível obter uma medição do software de acordo com os parâmetros estabelecidos pela ferramenta em questão. As ferramentas conseguem automatizar tanto a aquisição quanto a apresentação dos valores das métricas de forma precisa e consistente (Tomas; Escalona; Mejias, 2013).

Atualmente tem-se um grande número de opções de ferramentas disponíveis no mercado, como SonarQube, Embold, Coverity Scan e Understand, capazes de extrair e exibir automaticamente medidas de projetos de software. Cada ferramenta possui sua lista de análise de métricas e cada uma das métricas é analisada de acordo com a sua definição disponibilizada pelo próprio fornecedor, algumas métricas podem ter como parâmetro de cálculo fontes externas ao fornecedor.

3. Metodologia

Para alcançar os objetivos propostos neste trabalho foram utilizados seis passos em sequência, como mostra figura a seguir:

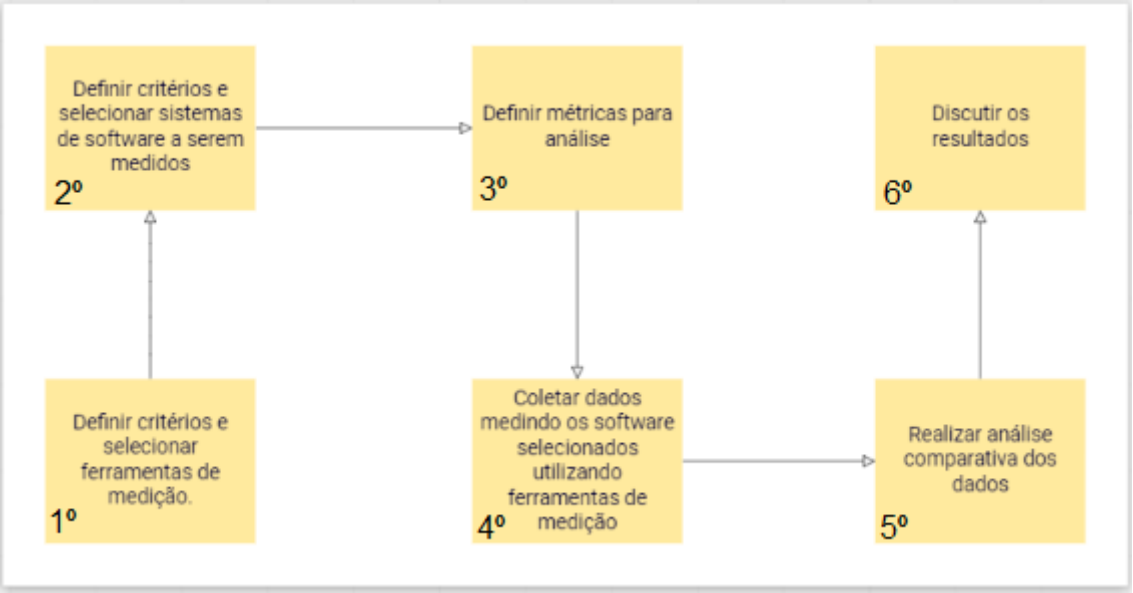

Figura 1. Passos da metodologia

Fonte: Elaborado pelo autor

Definir critérios e selecionar ferramentas de medição: Para selecionar as ferramentas de análise, define-se um protocolo, a fim de se obter as ferramentas necessárias, tendo como base linguagem de programação, tipo de análise, disponibilização da ferramenta no mercado, ter versão gratuita ou de teste e realizar medição de métricas. A saída desse passo é uma relação contendo os critérios definidos e um conjunto das ferramentas selecionadas de acordo com os critérios definidos

Definir critérios e selecionar projetos de software a serem medidos: Para selecionar os projetos de software, define-se um protocolo, a fim de se obter os projetos necessários, tendo como base a linguagem de programação, ter código aberto e o tamanho dos projetos. A saída desse passo é uma relação contendo os critérios definidos e um conjunto de projetos de software selecionados que cumprem os critérios definidos.

Definir métricas para análise: Deve-se fazer um relacionamento entre as métricas que as ferramentas contemplam na sua análise e, com isso, tem-se as métricas que as ferramentas possuem em comum. A saída desse passo é uma lista de métricas que todas as ferramentas selecionadas possuem.

Coletar dados medindo os softwares selecionados utilizando ferramentas de medição: Neste passo deve-se realizar uma análise utilizando as ferramentas selecionadas nos projetos de softwares escolhidos. A saída desse passo é um relatório da análise de cada ferramenta para cada software.

Realizar análise comparativa dos dados: Tendo como base os critérios de cada métrica utilizada, realizar uma análise comparativa dos relatórios obtidos, verificando os cálculos realizados de cada ferramenta. A saída desse passo são gráficos com as comparações das ferramentas para cada métrica.

Discutir os resultados: Utilizando dos dados, tabelas e gráficos obtidos anteriormente, busca-se validar os resultados encontrados e realizar uma análise tendo como foco a diferença e os critérios de cálculo entre as ferramentas. A saída desse passo é a resposta para o objetivo proposto no início do trabalho.

4. Desenvolvimento

Cada passo descrito na seção metodologia foi realizado com sucesso no desenvolvimento do trabalho, os resultados obtidos foram descritos de forma detalhada nesta seção.

4.1. Definir critérios e selecionar ferramentas de medição

Para selecionar as ferramentas de análise, foi seguido um protocolo, definido mais a frente, a fim de se obter as ferramentas necessárias para o desenvolvimento do trabalho, desta forma, foram definidos critérios para a busca das ferramentas. A fonte para obter as ferramentas foi exclusivamente o Google, utilizando as seguintes palavras chave nas buscas: *software metrics tools* e *software analysis tools*.

Os critérios de seleção para as ferramentas encontradas foram os seguintes:

Ser gratuita ou ter versão de teste: Para a ferramenta ser eleita deve ser totalmente gratuita ou, no seu pacote de obtenção, possuir um teste gratuito de no mínimo trinta dias, para que seja possível utilizar a ferramenta nos projetos.

Possuir uma versão disponível no mercado: A ferramenta deve ter uma versão em produção, essa restrição tem como objetivo utilizar apenas ferramentas que se encontram no mercado, pois existem ferramentas de código aberto em diversos repositórios, porém não são versões de mercado, tendo sua manutenção feita pela comunidade.

Possuir suporte para as linguagens C# e JAVA: Restrição imposta sobre a compatibilidade da ferramenta em relação às linguagens de programação, as linguagens escolhidas foram C# e JAVA, devido a familiaridade do autor com essas linguagens, podendo facilitar resolver problemas que pudessem ocorrer durante a análise e necessitasse de intervenção, e facilitar interpretar os resultados..

Utilizar análise estática: Há dois tipos de análise: análise estática, conhecida também como teste de caixa branca, e análise dinâmica, também conhecida como teste de caixa preta. A análise estática foi escolhida devido ao fato de não se precisar executar o projeto a ser analisado, ao contrário da análise dinâmica, cujo o projeto precisa ser executado para a obtenção dos dados.

Realizar medidas de softwares (métricas): Algumas ferramentas realizam apenas uma revisão de código no projeto, não fazendo cálculos de métricas. Para a ferramenta ser eleita ela deve realizar as medições de métricas e não apenas revisar o código automaticamente.

Ferramenta	Distribuição	Em produção	Suporta C# e JAVA	Análise estática	Realiza medição	Tempo para uso	Quantidade de Métricas
McCabe IQ	Teste	Sim	Sim	Sim	Sim	30 dias	42
CodeScene	Grat.	Sim	Sim	Sim	Não	Ilimitado	-
Coverity Scan	Grat.	Sim	Sim	Sim	Sim	Ilimitado	2
Deepsource	Teste	Sim	Não (C# em beta)	Sim	Sim	-	-
CodeSonar	Teste	Sim	Sim	Sim	Sim	30 dias	15
Embold	Teste	Sim	Sim	Sim	Sim	Ilimitado	19
Smartbear (Testcomplete)	Teste	Sim	Não (C# legado)	Sim	Sim	-	-
SonarQube	Grat.	Sim	Sim	Sim	Sim	Ilimitado	23
SciTools (Understand)	Teste	Sim	Sim	Sim	Sim	Ilimitado (estudante)	45
Squore	Pago	Sim	Sim	Sim	Sim	-	11
Parasoft (dotTest e JTest)	Teste	Sim	Sim	Sim	Sim	Necessita solicitação	26
PVS-Studio	Teste	Sim	Sim	Sim	Sim (utiliza o SonarQube)	-	-

Tabela 1. Lista inicial das ferramentas

Fonte: Elaborado pelo autor

A partir dessa lista inicial de ferramentas, foi feito um refinamento para escolher três delas, a princípio seriam escolhido de acordo com o maior número de métricas, sendo primeiramente escolhidas Understand, McCabe IQ e Parasoft (dotTest e JTest), porém, a solicitação do teste no site do McCabe IQ estava com problemas e não responderam o email solicitando o teste gratuito, após comunicação com a Parasoft foi constatado que a mesma não disponibiliza teste gratuito para o dotTest e o JTest. Desta forma foram selecionadas: Understand (v6.1.1096), SonarQube (v9.4.0) e Embold (v2). Vale ressaltar que as ferramentas que são pagas, sem teste gratuito, suporte limitado à alguma das duas linguagens escolhidas e/ou não realizam medição, não foram consideradas para a escolha.

4.2. Definir critérios e selecionar projetos de software a serem medidos

Para selecionar os projetos de software, foi seguido um protocolo, definido mais a frente, a fim de se obter os projetos necessários para o desenvolvimento do trabalho, desta forma foram definidos critérios para a busca dos mesmos. A fonte para obter os projetos foi exclusivamente o GitHub, utilizando os seguintes tópicos nas buscas: JAVA e CSHARP. Definiu-se três projetos para a linguagem JAVA e três projetos para a linguagem C#, totalizando seis projetos de softwares, sendo escolhidos arbitrariamente.

Os critérios de seleção para os projetos de softwares são os seguintes:

Código aberto: Para a seleção do projeto de software o código fonte do projeto deve ter seu repositório como público no GitHub.

Estar escrito nas linguagens C# ou JAVA: Todos os softwares devem estar escritos com pelo menos 95% do código fonte nessas linguagens, visto que as ferramentas possuem a mesma restrição.

Tamanhos do projeto de software: Os projetos, tanto em JAVA quanto em C#, devem possuir tamanhos diferentes, sendo definido três tamanhos: pequeno, médio e grande. Para o projeto pequeno foi definido um software com até 10.000 linhas de código, para o projeto médio foi definido um software entre 10.001 e 100.000 linhas de código e finalmente para o projeto grande foi definido um software com mais de 100.000 linhas de código. Sendo essas definições utilizadas para esse trabalho, visto que não se tem uma definição exata para os tamanhos de um software. Vale ressaltar que o cálculo das linhas de código, nesse primeiro momento, foi feito utilizando a API do GitHub.

Projeto de software	Branch	Tamanho
Phoenix	Master	Pequeno (aprox. 1.319 linhas de código)
Dropwizard	release-2.1.x	Médio (aprox. 76.061 linhas de código)
FastJson	Master	Grande (aprox. 263.476 linhas de código)

Tabela 2. Projetos em JAVA utilizados na medição das ferramentas

Fonte: Elaborado pelo autor

Projeto de software	Branch	Tamanho
DesignPatterns	Master	Pequeno (aprox. 5.314 linhas de código)
C# Algorithms	Master	Médio (aprox. 33.664 linhas de código)
Jellyfin	Master	Grande (aprox. 273.569 linhas de código)

Tabela 3. Projetos em C# utilizados na medição das ferramentas

Fonte: Elaborado pelo autor

Os softwares utilizados no trabalho são descritos abaixo:

Phoenix: Este projeto visa fornecer um *pull* simples e personalizável para atualizar a implementação.

Dropwizard: O Dropwizard reúne bibliotecas estáveis do ecossistema Java em um pacote simples e leve para construir serviços web RESTful prontos para produção.

FastJson: Fastjson é uma biblioteca Java que pode ser usada para converter objetos Java em sua representação JSON. Ele também pode ser usado para converter uma string JSON em um objeto Java equivalente.

DesignPatterns: O DesignPatterns reúne diversas soluções para problemas recorrentes e apresenta orientações sobre como lidar com determinados problemas. Incluindo implementações de alguns padrões de projeto em C#.

C# Algorithms: Um projeto de biblioteca de classes plug-and-play de estruturas de dados e algoritmos padrão, escrito em C#. Ele contém mais de 75 estruturas de dados e algoritmos, projetados como componentes isolados orientados a objetos.

Jellyfin: Jellyfin é um sistema de mídia de software livre que controla o gerenciamento e *streaming* de mídia. Podendo transmitir para diversos tipos de dispositivos a partir de um servidor próprio.

4.3. Definir métricas para análise

A partir das ferramentas definidas, foi feito um mapeamento em relação às métricas que cada uma calcula, utilizando a documentação disponibilizada pelos fornecedores, desta forma obtendo as métricas em comum entre as três ferramentas escolhidas, as métricas que serão utilizadas na análise são:

Ferramenta	Métricas			
Embold	Lines of Code	Lines Of Code Comments	Cyclomatic Complexity	Comment Ratio
SonarQube	Lines of Code	Number of Comments Lines	Cyclomatic Complexity	Comment Lines Density
Understand	Lines of Code	Lines with Comments	Cyclomatic Complexity	Comment to Code Ratio

Tabela 4. Métricas em comum das ferramentas

Fonte: Elaborado pelo autor

Após o mapeamento das métricas em comum, foram identificadas as quatro métricas exibidas na tabela acima, a seguir uma descrição das métricas identificadas seguindo a nomenclatura do Embold:

Lines of Code **(LOC)**: É a quantidade de linhas contidas no programa.

Lines of Comments **(LOCM)**: É a quantidade de linhas de comentários.

Cyclomatic Complexity **(CC)**: É a medida que indica a complexidade do programa em relação aos caminhos linearmente independentes.

Comment Ratio **(CR)**: É o percentual entre as linhas comentadas e o total de linhas de código no programa.

4.4. Coletar dados medindo os softwares selecionados utilizando ferramentas de medição

Com as ferramentas escolhidas, os projetos selecionados e as métricas a serem medidas foi feito o scanner em cada ferramenta de medição. Foi executado o scanner em cada um dos projetos de softwares, separados pela linguagem de programação.

Cada ferramenta possui suas características, sendo o Embold e o SonarQube utilizados via nuvem, o Embold é nativamente utilizado via aplicação web onde é feito um login utilizando o GitHub e os repositórios escolhidos são escaneados. O SonarQube possui o SonarCloud que é um serviço em nuvem que executa a análise do SonarQube no repositório do GitHub, similar ao realizado pelo Embold, porém para os projetos C# foi necessário a criação de um arquivo de construção da aplicação, exemplificados pelo próprio SonarQube, vale ressaltar que é necessário realizar um *fork* do projeto para a sua conta do GitHub. Por outro lado o Understand é um aplicativo desktop e sua análise é feita utilizando o repositório local, sendo necessário baixar o projeto para a própria máquina.

Os dados de medição obtidos estão disponíveis nas tabelas logo abaixo, as tabelas indicam os resultados da medição de cada ferramenta nos projeto de software escolhido, separados de acordo com a linguagem de programação.

Embold			
Métricas	Phoenix	Dropwizard	FastJson
LOC	582	33.145	55.763
LOCM	24	7.241	5.639
CC	22	1.313	7.556
CR	0,145	0,5489	0,7019

Tabela 5. Resultado da análise dos softwares em JAVA pelo Embold

Fonte: Elaborado pelo autor

Understand			
Métricas	Phoenix	Dropwizard	FastJson
LOC	1.066	64.564	187.999
LOCM	44	9.146	15.759
CC	159	7.728	31.248
CR	0,0245	0,2108	0,1021

Tabela 6. Resultado da análise dos softwares em JAVA pelo Understand

Fonte: Elaborado pelo autor

SonarQube			
Métricas	Phoenix	Dropwizard	FastJson
LOC	1.069	30.962	44.286
LOCM	25	5.659	2.950
CC	165	3.949	11.192
CR	0,023	0,155	0,062

Tabela 7. Resultado da análise dos softwares em JAVA pelo SonarQube

Fonte: Elaborado pelo autor

Embold			
Métricas	Design Patterns	C# Algorithms	Jellyfin
LOC	3.508	28.334	222.247
LOCM	10	6.160	37.006
CC	237	2.000	13.101
CR	0,003	0,3469	0,5664

Tabela 8. Resultado da análise dos softwares em C# pelo Embold

Fonte: Elaborado pelo autor

Understand			
Métricas	Design Patterns	C# Algorithms	Jellyfin
LOC	3.166	18.420	158.553
LOCM	10	6.263	37.494
CC	481	3.872	28.665
CR	0,0029	0,3353	0,5398

Tabela 9. Resultado da análise dos softwares em C# pelo Understand

Fonte: Elaborado pelo autor

SonarQube			
Métricas	Design Patterns	C# Algorithms	Jellyfin
LOC	2.820	13.987	143.086
LOCM	6	5.185	36.858
CC	438	3.826	29.936
CR	0,002	0,27	0,205

Tabela 10. Resultado da análise dos softwares em C# pelo SonarQube

Fonte: Elaborado pelo autor

4.5. Realizar análise comparativa dos dados

Após realizar a coleta de dados foi possível perceber que quase todas as medições foram diferentes entre as ferramentas, algumas foram próximas outras divergiram muito numericamente. Com isso em mente foi realizada uma análise comparativa dos dados obtidos.

A fim de se obter uma significância na diferença dos resultados foi feito uma análise sobre o valor normalizado. A normalização dos valores foi feita dividindo os valores de cada métrica pelo maior valor obtido da mesma métrica entre as três ferramentas, separando-as pela linguagem de programação, exemplificando: o maior valor obtido para LOC no projeto Phoenix (JAVA), considerando todas as ferramentas de medição, foi de 1.069, sendo assim todos os valores de LOC do projeto Phoenix foram divididos por 1.069, normalizando os dados obtidos da métrica.

4.5.1 Análise comparativa da métrica *Lines of Code* (LOC)

A métrica LOC consta nas três ferramentas de medição, segue abaixo a definição dessa métrica em cada uma das ferramentas.

Embold: Segundo a documentação do Embold, a métrica LOC é o número total de linhas do arquivo. As linhas de comentários e as linhas em branco também são contadas.

Understand: Segundo a documentação do Understand a métrica LOC é o número de linhas contendo código-fonte, incluindo regiões inativas (as regiões de código inativas são regiões controladas por diretivas de compilação ou configuração, cujo trecho de código pode mudar de acordo com esses controladores).

SonarQube: Segundo a documentação do SonarQube a métrica LOC é o número de linhas físicas que contêm pelo menos um caractere que não é um espaço em branco, nem uma tabulação, nem parte de um comentário.

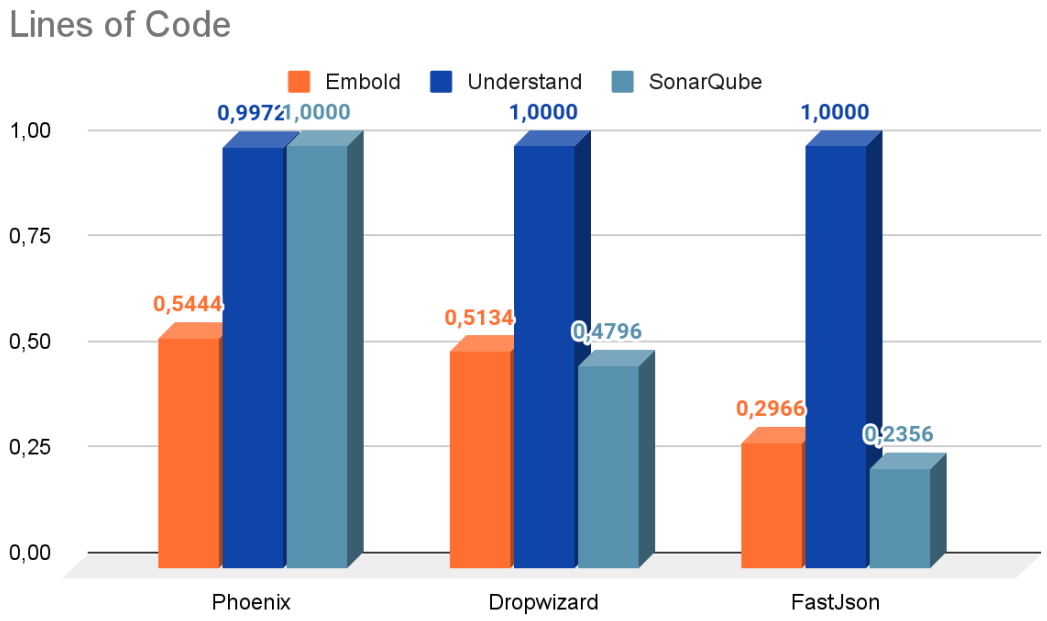

Figura 2. Gráfico da análise comparativa da métrica *Lines of Code* para os projetos em JAVA

Fonte: Elaborado pelo autor

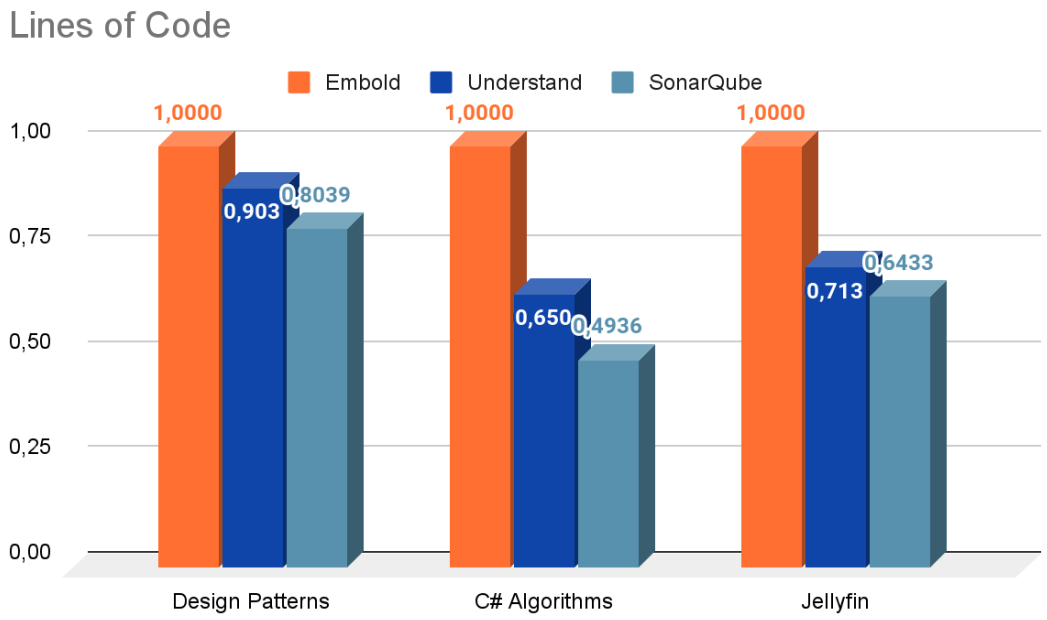

Figura 3. Gráfico da análise comparativa da métrica *Lines of Code* para os projetos em C#

Fonte: Elaborado pelo autor

As ferramentas Embold, Understand e SonarQube apresentaram diferenças nas medições nos projetos em JAVA, tendo somente uma grande aproximação entre o

Understand e o SonarQube no projeto Phoenix, correspondendo a uma diferença de 0,279% entre elas. Nos demais projetos em JAVA, a ferramenta Understand registrou mais linhas de código do que as demais, tendo diferenças significativas como mostra o gráfico da figura 2.

Nos projetos em C# as ferramentas Understand e SonarQube se comportaram de forma semelhante, não tendo uma diferença crítica entre as duas, a maior diferença foi no projeto C# Algorithms com 15,64%. Por outro lado, o Embold apresentou as maiores medidas entre as outras duas ferramentas, sendo a maior diferença de 35,67% e a menor de 9,7%, sendo a maior diferença no projeto C# Algorithms, em comparação com o SonarQube, e a menor diferença no projeto Design Patterns, em comparação com a Understand.

Para a métrica LOC é possível observar que as definições das métricas são distintas, o que ocasionou em diferenças nas medições das ferramentas. Não tendo uma equivalência entre as ferramentas, pois nenhuma das ferramentas obteve valores iguais e ambas não possuem os mesmos critérios de contagem.

4.5.2 Análise comparativa da métrica *Lines of Comments* (LOCM)

A métrica LOCM consta nas três ferramentas de medição, segue abaixo a definição dessa métrica em cada uma das ferramentas.

Embold: Segundo a documentação do Embold, *Lines of Comments* é o número de linhas de comentário em um componente.

Understand: Segundo a documentação do Understand, LOCM é o número de linhas contendo comentários, incluindo regiões inativas (as regiões de código inativas são regiões controladas por diretivas de compilação ou configuração, cujo trecho de código pode mudar de acordo com esses controladores).

SonarQube: Segundo a documentação do SonarQube, LOCM é o número de linhas contendo comentários ou código comentado. Linhas de comentário não significativas (linhas de comentário vazias, linhas de comentário contendo apenas caracteres especiais, etc.) não aumentam o número de linhas de comentário.

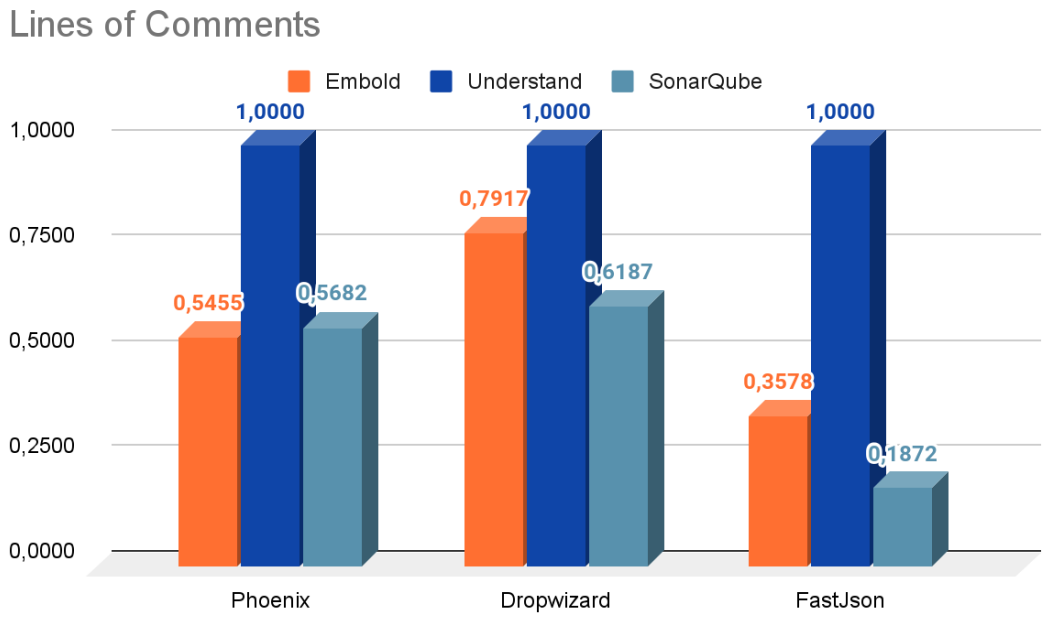

Figura 4. Gráfico da análise comparativa da métrica *Lines of Comments* para os projetos em JAVA

Fonte: Elaborado pelo autor

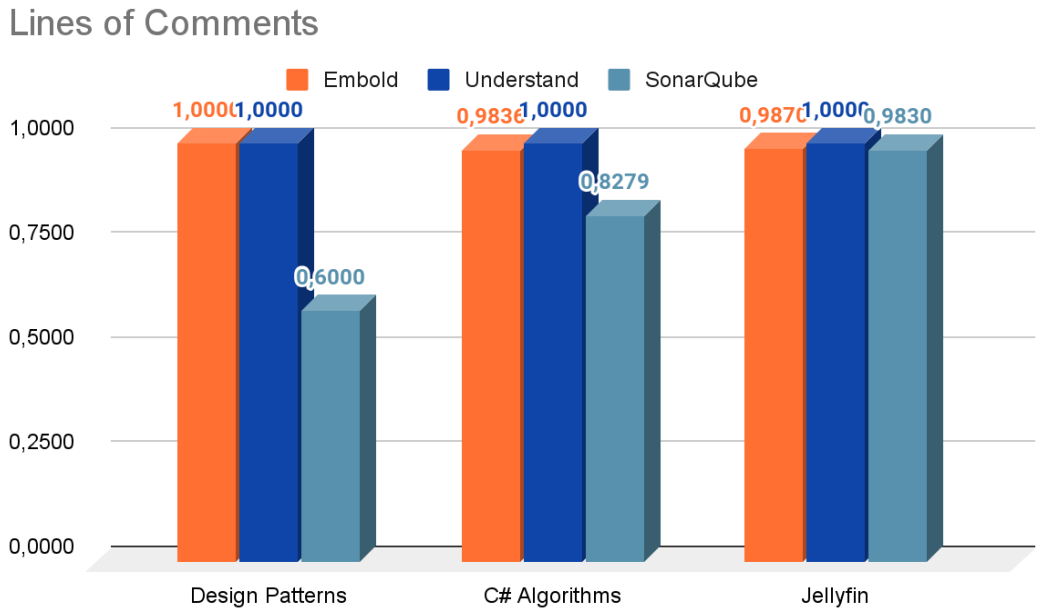

Figura 5. Gráfico da análise comparativa da métrica *Lines of Comments* para os projetos em C#

Fonte: Elaborado pelo autor

Nos projetos em JAVA teve-se uma grande diferença nas medições, onde a ferramenta Understand marcou os maiores valores. A maior diferença está no projeto de grande porte, o FastJson, cujo a diferença entre o Embold e o Understand foi de 64,22%

e a diferença entre o SonarQube e o Understand foi de 81,29%. A medição mais próxima do cenário JAVA foi no projeto de médio porte, o Dropwizard, onde a menor diferença foi de 20,83%, entre o Embold e o Understand.

As ferramentas tiveram um comportamento mais próximo nos projetos em C#. O Embold e o Understand obtiveram a mesma medição para o projeto de pequeno porte, o Design Patterns, e nos demais projetos as medições foram muito próximas tendo a maior diferença de 1,65% no projeto de médio porte (C# Algorithms). Por outro lado o SonarQube obteve diferenças entre as outras duas ferramentas, principalmente no projeto de pequeno porte, tendo uma diferença de 40%, contudo apesar de ser uma diferença grande na porcentagem, em valor absoluto foram apenas 4 linhas de diferença. No projeto de grande porte (Jellyfin), o SonarQube se comportou muito próximo das outras ferramentas, tendo uma diferença de 1,7% entre o Understand e 0,39% entre o Embold.

Para a métrica LOCM foi possível observar que as definições das métricas são distintas, o que ocasionou em diferenças nas medições das ferramentas. Nos projetos em JAVA todas as ferramentas obtiveram valores distintos com uma diferença significativa. Entretanto, as ferramentas Embold e Understand se comportaram de maneira muito próxima nos projetos em C#, obtendo medições iguais no projeto de pequeno porte e nos demais a diferença não foi significativa, sendo a maior diferença entre eles no projeto C# Algorithms, com uma variação de 1,64%.

4.5.3 Análise comparativa da métrica *Cyclomatic Complexity* (CC)

A métrica CC consta nas três ferramentas de medição, segue abaixo a definição dessa métrica em cada uma das ferramentas.

Embold: Segundo a documentação do Embold, a Complexidade Ciclomática (CC) é uma medida da complexidade do programa alcançada medindo o número de caminhos linearmente independentes através do código-fonte de um programa. O método de McCabe é usado para calcular a métrica.

Understand: Segundo a documentação do Understand, a complexidade ciclomática de qualquer programa estruturado com apenas um ponto de entrada e uma saída é igual ao número de pontos de decisão contidos nesse programa mais um. Utiliza o método McCabe para o cálculo.

SonarQube: Segundo a documentação do SonarQube, a Complexidade Ciclomática é calculada com base no número de caminhos através do código.

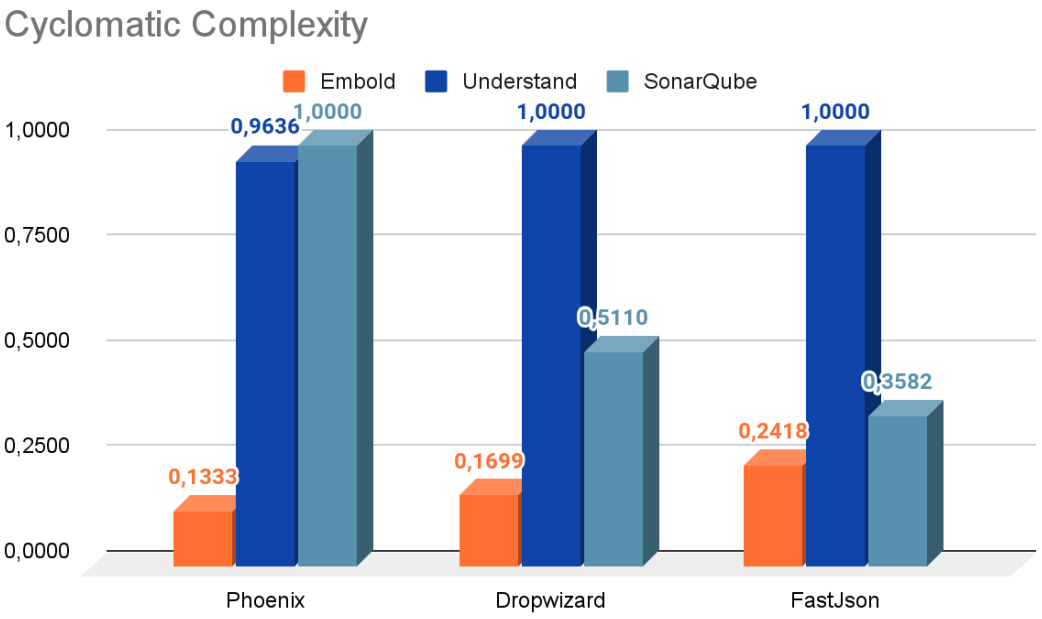

Figura 6. Gráfico da análise comparativa da métrica *Cyclomatic Complexity* para os projetos em JAVA

Fonte: Elaborado pelo autor

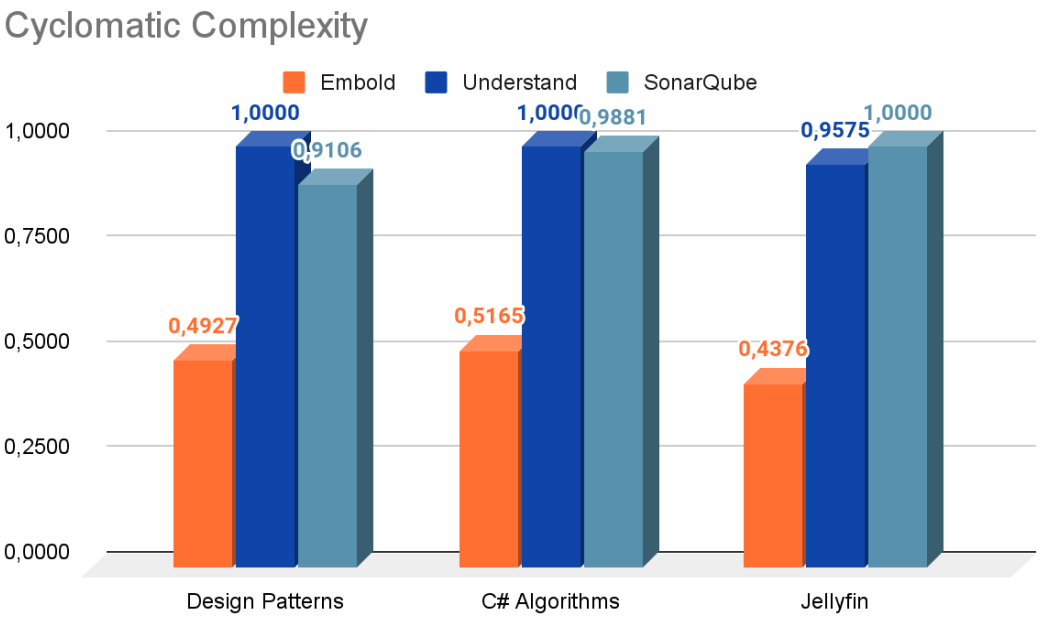

Figura 7. Gráfico da análise comparativa da métrica *Cyclomatic Complexity* para os projetos em C#

Fonte: Elaborado pelo autor

No cenário da linguagem JAVA a maior diferença registrada foi entre o Embold e o SonarQube no projeto de pequeno porte, o Phoenix, com uma diferença de 86,67%. A

menor diferença para o JAVA foi também no projeto de pequeno porte, com uma diferença de 3,64% entre o SonarQube e o Understand.

Para o C# o Embold foi o que obteve os menores valores de complexidade. O Understand e o SonarQube obtiveram valores próximos, principalmente no projeto de médio porte, o C# Algorithms, com uma variação de 1,65% entre eles. A maior variação no cenário do C# foi entre o Embold e o SonarQube no projeto de grande porte (Jellyfin) com uma diferença de 56,24%.

Para a métrica CC é possível observar que as definições das métricas são distintas, mesmo duas das ferramentas utilizando o método McCabe para o cálculo da métrica, o que ocasionou em diferenças nas medições das ferramentas. Não tendo uma equivalência entre as ferramentas, pois nenhuma das ferramentas obteve valores iguais e ambas não possuem os mesmos critérios de contagem para caminhos distintos.

4.5.4 Análise comparativa da métrica *Comment Ratio* (CR)

A métrica CR consta nas três ferramentas de medição, segue abaixo a definição dessa métrica em cada uma das ferramentas.

Embold: Segundo a documentação do Embold, *Comment Ratio* é simplesmente a razão entre a linha de comentários e o número total de linhas de código.

Understand: Segundo a documentação do Understand, *Comment Ratio* é a relação entre o número de linhas de comentário e o número de linhas de código. Podendo uma linha ser os dois, código e comentário.

SonarQube: Segundo a documentação do SonarQube, *Comment Ratio* é a razão entre as linhas de comentários e a soma entre as linhas de código e as linhas com comentários. Comment lines / (Lines of code + Comment lines) * 100.

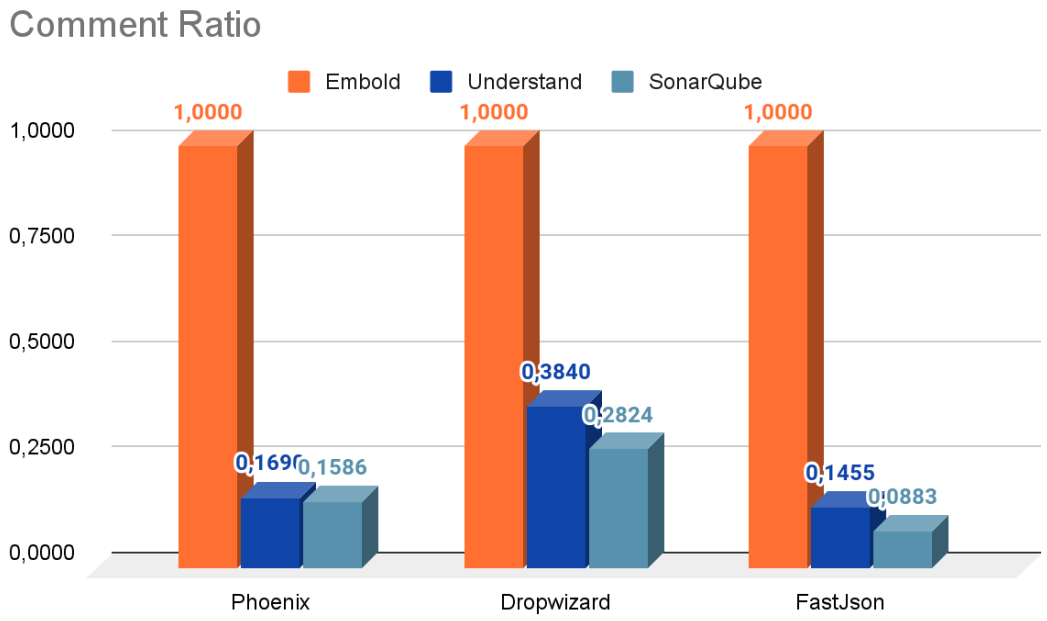

Figura 8. Gráfico da análise comparativa da métrica *Comment Ratio* para os projetos em JAVA

Fonte: Elaborado pelo autor

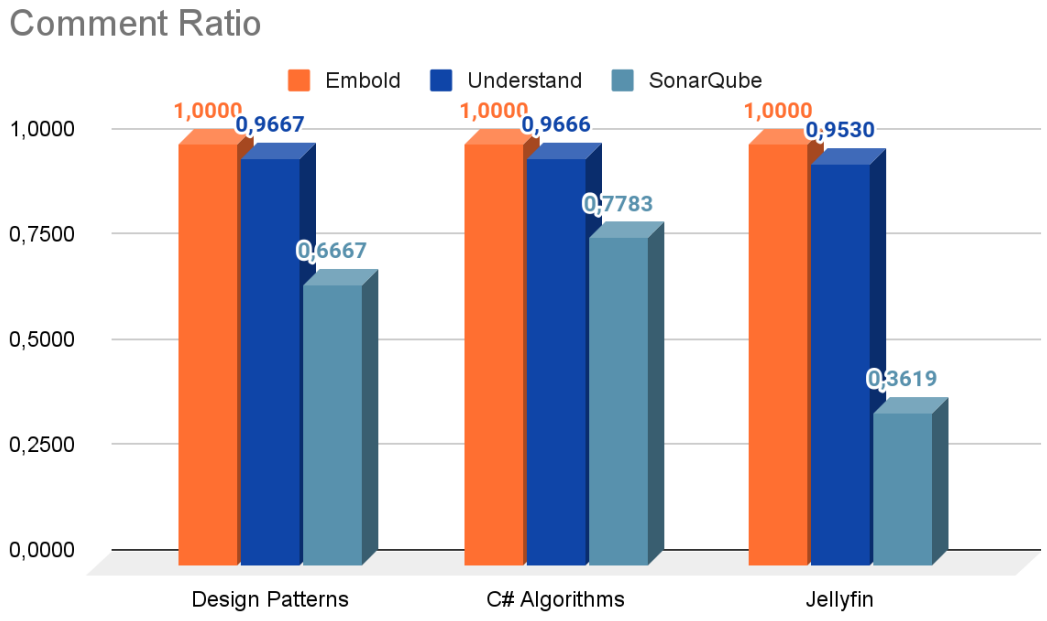

Figura 9. Gráfico da análise comparativa da métrica *Comment Ratio* para os projetos em C#

Fonte: Elaborado pelo autor

As ferramentas apresentaram medições distintas nos projetos JAVA, apenas no no projeto de pequeno porte (Phoenix), as ferramentas SonarQube e Understand tiveram

medições próximas com uma variação de 6,1%. Nos demais projetos teve-se uma diferença significativa entre as ferramentas, principalmente em relação ao Embold que registrou as maiores medições em todos os projetos JAVA, a maior variação foi registrada no projeto de grande porte (FastJson) entre o Embold e o SonarQube, com uma diferença de 91,17% entre elas.

Nos projetos em C# não houve medições iguais, tendo o Embold e o Understand medições próximas, sendo a menor delas com uma variação de 3,34% no projeto de pequeno porte (Design Patterns) e maior variação foi de 4,7% no projeto de grande porte (Jellyfin). Porém o SonarQube foi o que mais teve variações entre as demais ferramentas, a maior variação registrada foi no projeto de grande porte (Jellyfin) 63,81% em comparação com o Embold, e a menor variação foi de 18,82% no projeto de médio porte (C# Algorithms) em comparação com o Understand.

Para a métrica CR foi possível observar que as definições das métricas são semelhantes, porém o cálculo dessa métrica depende da definição da ferramenta sobre linhas de código o que ocasionou em diferenças nas medições das ferramentas. Para os projetos JAVA a diferença foi grande e significativa, entretanto as ferramentas Embold e Understand se comportaram de maneira muito próxima nos projetos em C#, obtendo medições próximas em todos os projetos C# sendo a diferença não significativa em termos percentuais.

5. Discussão dos resultados

Após o término do desenvolvimento, fica claro que as ferramentas, no geral, tiveram medições diferentes nas métricas em comum, isso se dá pelo fato de que algumas métricas possuem diferentes definições e formas de calcular.

Havia uma expectativa de que haveria mais métricas em comum, o que não ocorreu, ficando o estudo com a comparação de poucas métricas. Havia, também, a expectativa de ter comparado mais ferramentas, porém não foi possível usar duas delas (Parasoft e McCabe IQ) por não ter uma versão de teste gratuita liberada pela empresa.

As quatro métricas foram escolhidas por serem métricas que todas as ferramentas de medição possuíam, com isso, foi possível uma comparação direta nos resultados, porém ficou claro que a definição das métricas e formas de calcular diferem.

A métrica de LOC, uma das métricas mais básicas, teve uma variância grande entre as ferramentas, principalmente nos softwares em JAVA, isso é relacionado com como a ferramenta trata uma linha de código, a ferramenta Embold define uma linha de código como qualquer linha que se tenha no código, contabilizando comentários e linhas em branco, a ferramenta Understand entende como linha de código uma linha contendo código fonte e incluí regiões inativas do código, por outro lado o SonarQube entende linha de código como uma linha que contém pelo menos um caractere e não seja uma linha em branco e nem tabulação. Desta forma cada ferramenta tem uma característica específica para a contagem da métrica.

O mesmo acontece com a métrica *lines of comments*, cada ferramenta tem uma característica para contabilizar a métrica. O Embold considera como comentário qualquer linha que se inicie com o caractere especial de comentário, a mesma definição para o Understand, porém adicionando regiões inativas do código, já o SonarQube

contabiliza de forma diferente, linhas de comentário não significativas (linhas de comentário vazias, contendo apenas caracteres especiais, etc.) não aumentam o número de linhas de comentário. Isso acaba gerando diferenças nas medições das ferramentas.

Se tratando de características específicas para a contagem das métricas, um exemplo que mostra muito bem esse cenário é a métrica de complexidade ciclomática, onde duas das ferramentas, Embold e Understand, utilizam o método McCabe para calcular, porém foi observado que teve diferenças significativas entre essas duas ferramentas. Uma das possíveis causas é o fato de que a ferramenta Embold apenas considera caminhos linearmente independentes nos casos de IF-ELSE ou SWITCH e laços FOR, para o caso da Understand ele não contabiliza o SWITCH em si, mas contabiliza cada um dos CASE e além disso é considerado caminhos linearmente independentes: CATCH, ternário (“?”) e DO. Para o SonarQube, além desses gatilhos de caminhos independentes, citados anteriormente, tem-se outros como WHEN e THROW, variando de acordo com a linguagem, o que justifica medições maiores de complexidade em alguns projetos.

Outro fator que vale ressaltar é a questão de dependência de uma métrica em relação à outra, como ocorreu na métrica *Comment Ratio*. A definição de todas as ferramentas são muito parecidas, porém dependentes da métrica LOC, que possui definição diferente em cada uma das ferramentas, o que ocasionou em divergências significativas nas medições.

Um possível fator para a variação de valores em métricas com mesma definição e critério de contagem, seria a contabilização de arquivos de configuração e/ou bibliotecas utilizadas no código. Como foram utilizados projetos de software em duas linguagens, apesar de não poder comparar as medidas entre softwares de linguagens distintas, foi possível observar que, em algumas métricas, os projetos em C# tiveram uma maior similaridade entre os valores das métricas do que os projetos em JAVA. Essa variação pode ocorrer devido a como a ferramenta lida com arquivos de configuração de determinada linguagem ou as bibliotecas que estão incluídas no projeto. Existe essa possibilidade, contudo não foi avaliado neste trabalho de pesquisa.

6. Conclusão

Para a conclusão do presente trabalho, com os dados coletados, foi possível observar que em alguns projetos, determinadas ferramentas tiveram um comportamento semelhante com diferenças estatisticamente insignificativas, mostrando definições e critérios de cálculo alinhados entre as ferramentas. Porém, por outro lado, foi observado o contrário, variações grandes entre a medição das métricas, estatisticamente significativas.

Em decorrência da análise comparativa realizada neste trabalho, é relevante apontar algumas considerações.

Foi possível notar que as ferramentas de medição de software analisadas tiveram discordância nos valores medidos na coleta de dados, essa discordância ocorreu mesmo utilizando ferramentas de medição que utilizavam o mesmo tipo de análise (estática), com a mesma entrada (os mesmos softwares utilizados foram utilizados em cada uma das ferramentas) e a mesma métrica medida.

As ferramentas de medição de software possuem, em alguns casos, definições iguais porém diferentes critérios para o cálculo da métrica, como na métrica LOC que depende de como a ferramenta identifica uma linha de código, ou mesmo na LOCM, que tem a mesma ideia, porém no contexto de como a ferramenta identifica uma linha de comentário.

Outro cenário são métricas com definições e critérios para o cálculo da métrica iguais, contudo possuem características específicas por parte da ferramenta de medição, como foi o caso da métrica de complexidade ciclomática, onde cada ferramenta identificava um caminho linearmente independente de forma diferente.

Um último caso foi a dependência da definição de outras métricas para o cálculo, como observado na métrica de CR, onde a mesma dependia das definições da LOC e LOCM para o cálculo.

Esses fatores foram os que mais indicaram a diferenciação entre as medições dos projetos de software.

Desta forma, fica claro que a escolha da ferramenta de análise de software, em um projeto, é determinante para medição do software. Tendo que ser levado em conta como a ferramenta irá calcular determinada métrica e quais métricas a ferramenta possui em sua análise.

Por fim é possível concluir que, a partir das ferramentas de medição de software analisadas, não há uma equivalência entre as ferramentas e as diferenças das medidas são, no geral, estatisticamente significativas.

6.1. Trabalhos Futuros

Como trabalhos futuros sugere-se:

- Fazer uma análise comparativa de ferramentas de análise de débitos técnicos.
- Fazer uma análise do SonarQube, que também faz análise de débitos técnicos, para verificar se os débitos que a ferramenta indica de fato são débitos ou se alguns débitos técnicos não precisam ser corrigidos porque durante a compilação ou interpretação o otimizador de código já faz a correção, permitindo que o código tenha uma leitura mais amigável.
- Realizar uma comparação de uma determinada ferramenta de análise de software se comporta em diferentes linguagens de programação, de forma a verificar se características da linguagem afetam as medições das métricas.
- Analisar a qualidade dos testes de vulnerabilidade feitos pelo SonarQube em diferentes linguagens.
- Fazer uma análise comparativa de ferramentas de análise de vulnerabilidades.

Referências Bibliográficas

SOMMERVILLE, I.; ARAKAKI, R.; SHIN, S. Engenharia de software (8a. ed.). São Paulo: Pearson Educación, 2008.

TOMAS, P.; ESCALONA, M. J.; MEJIAS, M. Open source tools for measuring the Internal Quality of Java software products. A survey. Computer Standards & Interfaces, v. 36, n. 1, p. 244–255, nov. 2013.

DE PÁDUA, W. Engenharia de software : fundamentos, métodos e padrões. Rio De Janeiro (RJ): Ltc, 2009.

FENTON, N. E.; BIEMAN, J. Software metrics : a rigorous and practical approach. Boca Raton: Crc Press, 2015.

ROSENBERG, L. H.; SHEPPARD, S. B. Metrics in software process assessment, quality assurance and risk assessment. Proceedings of 1994 IEEE 2nd International Software Metrics Symposium, [s.d.].

HENDERSON-SELLERS, B. OO software process improvement with metrics. Proceedings Sixth International Software Metrics Symposium (Cat. No.PR00403), [s.d.].

DE VASCONCELOS, Alexandre Marcos Lins et al. Introdução à Engenharia de Software e à Qualidade de Software. Minas Gerais: Universidade Federal de Lavras, 2006.

MEIRELLES, Paulo Roberto Miranda. Monitoramento de métricas de código-fonte em projetos de software livre. 2013. Tese (Doutorado em Ciência da Computação) - Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2013.

Continuous Inspection | SonarQube. Disponível em: <https://www.sonarqube.org/>. Acesso em: 5 abr. 2022.

Embold | Better code today. Disponível em: <https://embold.io/>. Acesso em: 5 abr. 2022.

Coverity Scan - Static Analysis. Disponível em: <https://scan.coverity.com/>. Acesso em: 5 abr. 2022.

SciTools. Disponível em: <https://www.scitools.com/>. Acesso em: 5 abr. 2022.

Java | Oracle. Disponível em: <https://www.java.com/>. Acesso em: 5 abr. 2022.

BILLWAGNER. Documentação do C# – introdução, tutoriais, referência. Disponível em: <https://docs.microsoft.com/pt-br/dotnet/csharp/>. Acesso em: 5 abr. 2022.

Software Quality, Testing, and Security Analysis | McCabe - The Software Path Analysis Company. Disponível em: <http://www.mccabe.com/>. Acesso em: 9 abr. 2022.

Automated Testing to Deliver Superior Quality Software. Disponível em: <https://www.parasoft.com/>. Acesso em: 9 abr. 2022.

GITHUB. GitHub. Disponível em: <https://github.com/>.

Metrics – Embold Help Center. Disponível em: <https://docs.embold.io/metrics/>. Acesso em: 13 abr. 2022.

What Metrics Does Understand Have? | Understand. Disponível em: <https://support.scitools.com/support/solutions/articles/70000582223-what-metrics-does-understand-have->. Acesso em: 13 abr. 2022.

Metric Definitions | SonarQube Docs. Disponível em: <https://docs.sonarqube.org/latest/user-guide/metric-definitions/>.