Análise Comparativa de Qualidade do Código-Fonte de Diferentes Classes de Navegadores Web para Sistemas Android

Felipe Augusto Silva Marques[1], Lesandro Ponciano[1]

Bacharelado em Engenharia de Software
[1] Instituto de Ciências Exatas e Informática – PUC Minas
Ed. Fernanda. Rua Cláudio Manoel, 1.162, Funcionários, Belo Horizonte – MG – Brasil

{felipe.marques@sga.pucminas.br, lesandrop@pucminas.br}

Abstract. *While there is a wide variety of Web browsers available today that can be used by users, little is known about the quality of the source code of these browsers. The quality of software source code is important because poorly written or designed software can bring inconvenience to users, including security and privacy concerns. This study aims to conduct a comparative analysis of the quality of the source code of browsers for Android that have open source license. The study is conducted by applying source code quality metrics. The motivation of this study is due to the increase in the number of Internet access through mobile devices, in addition to the great diversity of browsers available for Android and the importance of analyzing the quality of the source code of these systems.*

key-words: Quality, Metrics, Source Code, Android

Resumo. *Enquanto existe uma ampla diversidade de navegadores Web disponíveis atualmente e que podem ser usados pelos usuários, pouco se sabe sobre a qualidade do código fonte destes navegadores. A qualidade do código-fonte de um software é importante, pois um software mal escrito ou projetado pode trazer transtornos aos usuários, envolvendo aspectos como segurança e privacidade. Este estudo tem como objetivo realizar uma análise comparativa da qualidade do código-fonte de navegadores para Android que possuem licença open source. O estudo é feito através da aplicação de métricas de qualidade de código-fonte. A motivação deste estudo deve-se ao crescimento no número de acesso à Internet através de dispositivo móveis, além da grande diversidade de navegadores disponíveis para Android e a importância de se analisar a qualidade do código-fonte destes sistemas.*

Palavras: chave: Qualidade, Métricas, Código-Fonte, Android

1. Introdução

Estima-se que o número de usuários de Internet por meio de dispositivos móveis será de 395.400 pessoas em 2020 na América Latina [Equipe Dub Soluções 2017]. Com o crescimento da utilização de dispositivos móveis cresceu juntamente a diversidade de navegadores *Web*. Navegadores são *softwares* que apresentam páginas *Web* estáticas e dinâmicas [Tanenbaum 2011]. Há navegadores com diferentes características, porque cada um deles atende uma demanda de mercado específica, como segurança e privacidade. Uma parte destes navegadores são desenvolvidos em código aberto. Exemplos destes navegadores são Mozilla Firefox, Mozilla Firefox Focus e *Tor Browsers*.

Enquanto existe uma ampla diversidade de navegadores que podem ser usados pelos usuários, pouco se sabe sobre a qualidade do código-fonte destes navegadores. **A falta de informações sobre a qualidade do código-fonte desse tipo de *software* é, portanto, o problema tratado neste estudo.** A qualidade de um sistema é definida como o grau em que o

sistema satisfaz os requisitos de suas várias partes interessadas e, portanto, fornece valor a essas partes interessadas ou ao cliente [International Organization for Standardization 2011]. A qualidade de código-fonte de um *software* tem efeito na qualidade de uso que ele oferece. Para que se possa avaliar a qualidade do código-fonte de navegadores, este estudo utiliza métricas que visam quantificar atributos internos do *software* e a partir disso, avaliar a sua qualidade. Tais métricas também possibilitam realizar uma análise comparativa das diferentes classes de navegadores.

A importância deste trabalho deve-se ao fato da qualidade do código-fonte do *software* estar diretamente ligada à qualidade de uso, além do seu custo de desenvolvimento e manutenção. A qualidade do código afeta diretamente a confiabilidade a que o usuário está exposto e a possibilidade de manutenção do *software*, levando-o a ser estendido sem se degradar. O crescimento da diversidade de navegadores que possuem a licença de *software* livre permite verificar a qualidade de código-fonte destes *softwares* e estabelecer critérios de comparação entre eles. O surgimento de diferentes classes de navegadores, como os focados em segurança e os focados em privacidade, mostra a necessidade de uma análise da sua qualidade interna. Considerando este cenário, torna-se necessário medir e comparar a qualidade de cada classe de navegador.

Nesse contexto, o presente estudo tem como objetivo principal realizar uma análise comparativa da qualidade do código-fonte de diferentes classes de navegadores desenvolvidos para a plataforma de dispositivos *mobile Android*. Para alcançar o objetivo principal, almeja-se atingir os seguintes objetivos específicos i) definir uma abordagem de objetivo, perguntas e métricas (GQM, do inglês *Goal Question Metric*) de avaliação de código-fonte para navegadores; ii) caracterizar dados de navegadores através de aplicações da abordagem; iii) estabelecer critérios que permitam compreender qual classe de navegador apresenta uma melhor qualidade no código-fonte. Ao final deste estudo espera-se que com a análise dos resultados seja possível verificar a qualidade do código-fonte dos navegadores e ter critérios objetivos de comparação entre eles. Além de apresentar uma análise comparativa das classes de navegadores.

O restante do texto está organizado como segue. A Seção 2 consiste na fundamentação teórica, onde apresenta-se detalhadamente conceitos e teorias que fundamentam o estudo. Na seção 3 são apresentados trabalhos relacionados ao tema abordado neste estudo. Em seguida, a seção 4 destaca os materiais e métodos.

2. Fundamentação Teórica

Esta seção apresenta conceitos e teorias que fundamentam este trabalho. Dentre os tópicos a serem abordados estão i) Navegadores *Web*; ii) Qualidade de *software*; e iii) Métricas e GQM.

2.1. Navegadores Web

Desde o início da Internet os navegadores apresentam um papel importante. Os navegadores *Web* são *softwares* responsáveis por apresentar páginas *Web* [Tanenbaum 2011]. O navegador busca a página solicitada em servidores de Sistema de Nomes de Domínios (DNS, do inglês *Domain Name System*), interpreta seu conteúdo e exibe a página, formatada de modo apropriado, na tela do computador. Com o crescimento da utilização da Internet e consequentemente da utilização de navegadores *Web* surgiram diversos navegadores. Os mesmos podem ser divididos em diferentes classes. Para este estudo os navegadores são classificados em três classes a serem analisados com base no foco de cada navegador.

Na primeira classe encontram-se os *navegadores tradicionais*. Os navegadores tradicionais apresentam funcionalidades tidas como base para os navegadores. São *softwares*

que possibilitam aos seus usuários interagirem com documentos escritos em linguagens como a Linguagem de Marcação de Hipertexto (HTML, do inglês *HyperText Markup Language*) [Tanenbaum 2011]. A segunda classe de navegador é a focada em *segurança*. São navegadores que se preocupam com que pessoas mal-intencionadas não leiam ou modifiquem mensagens trocadas através das redes de computadores ou, ainda, que o navegador seja usado para o *download* de programas maliciosos [Tanenbaum 2011]. Por último, a terceira classe de navegador inclui aqueles focados em *privacidade*. A privacidade é o direito das pessoas preservar suas informações pessoais, permitindo o controle da exposição e disponibilidade de informações acerca de si mesmo [Tanenbaum 2011]. Navegadores que se preocupam com privacidade oferecem, por exemplo, a possibilidade de navegação anônima.

2.2. Qualidade de Software

A qualidade no contexto de produção e manutenção de *software* é importante [Pressman e Maxim 2016]. No gerenciamento de qualidade de código-fonte, torna-se necessário quantificar a complexidade de se realizar alterações no código ou acréscimo de novas funcionalidades. Este monitoramento da qualidade do código-fonte pode ser realizado por meio de técnicas de revisão e inspeção de qualidade de código, que tem como objetivo melhorar a qualidade de *software* [Sommerville 2011].

O desenvolvimento de código aberto como uma abordagem de desenvolvimento de *software* baseia-se em publicar o código-fonte de um *software* e voluntários são convidados a participar do processo de desenvolvimento [Sommerville 2011]. Uma das vantagens de projetos de código aberto é o compartilhamento do código fonte, o que pode melhorar a qualidade [Meireles 2013]. Isso se deve ao maior número de desenvolvedores e usuários envolvidos com a revisão e validação do *software*. Em outras palavras, um número maior de desenvolvedores, com diferentes perspectivas e necessidades, é capaz de identificar melhorias e corrigir mais erros em menos tempo e, consequentemente, promover refatorações que, geralmente, levam à melhoria da qualidade do código.

Qualquer que seja a metodologia de desenvolvimento, monitorar a qualidade do software é fundamental. Em um processo de coleta das métricas é necessário seguir as seguintes etapas i) identificar as metas; ii) identificar o que se deseja aprender; iii) identificar suas submetas; iv) identificar as entidades e atributos relacionados as submetas; v) formalizar suas metas de medição; vi) identificar questões quantificáveis e os indicadores, visando atingir os objetivos; vii) identificar os elementos de dados que vão ser coletados para construir os identificadores; viii) identificar as medidas a serem usadas e tornar essas definições operacionais; ix) identificar as ações que você tomará para implementar as medidas; x) preparar um plano para implantar as medidas [Pressman e Maxim 2016].

2.3. Métricas de Software e GQM

A medição de *software* preocupa-se com a derivação de um valor numérico ou o perfil para um atributo de um componente de *software*, sistema ou processo [Sommerville 2011]. Há uma necessidade de medir e controlar a complexidade do *software* [Pressman e Maxim 2016]. E, se é difícil obter um valor único desta complexidade de um *software*, pode-se desenvolver um modelo de qualidade que agrega diferentes atributos internos do *software*. Com isso, as métricas auxiliam a quantificar a complexidade do *software*. Há diversas métricas de *software*. Elas podem ser categorizadas como: métricas de tamanho, métricas estruturais e métricas de acoplamento. Essas categorias são descritas nos parágrafos a seguir.

As *métricas de tamanho* são métricas que buscam estimar ou verificar o tamanho de um *software* [Pressman e Maxim 2016]. Apesar de nem sempre indicar a complexidade de um *software*, elas possibilitam verificar informações importantes como o percentual do código

escrito para interface, qual o módulo com maior número de linhas de código, verificar se o código está bem dividido em métodos, entre outras medições.

As *métricas estruturais* têm como objetivo mensurar questões estruturais do código. Por exemplo, mensurar questões relacionadas às classes no caso da programação orientada a objeto [Meirelles 2013]. Dentre os elementos a serem mensurados por estas métricas estão número de atributos públicos, número de métodos públicos, média do número de parâmetros por método, profundidade da árvore de herança, número de filhos de uma classe, média da complexidade ciclomática por método e número de atributos de uma classe.

Métricas de acoplamento são medidas de como uma classe está ligada a outras classes no *software* [Meirelles 2013]. Altos valores de acoplamento indicam uma maior dificuldade para alterar uma classe do sistema, pois uma mudança em uma classe pode ter um impacto em todas as outras classes que são acopladas a ela [Meirelles 2013]. Em outras palavras, se o acoplamento é alto, o *software* tende a ser menos flexível, mais difícil de se adaptar e modificar e mais difícil de entender. Exemplos de métricas de acoplamento são acoplamento entre objetos (CBO, do inglês *Coupling Between Classes*), fator de acoplamento (COF, do inglês *Coupling Factor*) e conexões aferentes de uma classe (ACC, do inglês *Aferent Connections per Class*).

O *GQM* é uma abordagem que auxilia na seleção das métricas. GQM permite identificar métricas significativas para qualquer parte do processo de *software* [Pressman e Maxim 2016]. O GQM enfatiza a necessidade de (1) estabelecer um objetivo de medição explícita que é específico para a atividade do processo ou característica de produto que deve ser avaliada, (2) definir um conjunto de questões que devem ser respondidas para atingir o objetivo e (3) identificar métricas bem formuladas que ajudam a responder a essas questões [Pressman e Maxim 2016].

3. Trabalhos Relacionados

Nesta seção são discutidos trabalhos que realizam pesquisas semelhantes ou relacionadas ao tema abordado nesta proposta. Tratam-se, em particular, de estudos sobre métricas de *software* e qualidade de *software*.

Meirelles (2013) apresenta uma abordagem para a observação das métricas de código-fonte, estudando-as através de suas distribuições e associações. Também discutem-se as relações de causalidade e implicações práticas-gerenciais para monitoramento das mesmas. São avaliadas as distribuições e correlações dos valores das métricas de trinta e oito projetos de *software* livre. Dentre as principais contribuições desse estudo, pode-se destacar uma análise detalhada, em relação ao comportamento, valores e estudos de caso de quinze métricas de código-fonte. O estudo também propõe uma abordagem que visa diminuir as contradições das análises das métricas.

Com o crescimento da utilização de métodos ágeis, faz-se necessário definir uma forma eficaz de aplicação de métricas nestes métodos. Sato (2007) cita que a Programação Extrema (XP, do inglês *Extreme Programming*) propõe uma atividade para guiar a equipe em direção à melhoria, a atividade é conhecida como *tracking*. O papel do *tracker* é coletar métricas para auxiliar a equipe a entender o andamento do projeto. O estudo investiga o uso de métricas no acompanhamento de projetos utilizando métodos ágeis de desenvolvimento de *software*. Um estudo de caso da aplicação de XP em sete projetos válida algumas dessas métricas e avalia o nível de aderência às práticas propostas, com o objetivo de auxiliar o *tracker* de uma equipe ágil. Algumas das métricas consideradas nesse estudo também serão utilizadas no presente estudo.

Júnior (2015) apresenta um estudo cujo o objetivo é o monitoramento de métricas estáticas de código fonte na interface de programação de aplicações (API, do inglês *Application Programming Interface*) do sistema operacional *Android*. Também é apresentado um estudo da evolução de seus valores nas diferentes versões da API, realizando uma apresentação entre as semelhanças com aplicativos do sistema.

Amara e Rabai (2017) apresentam um estudo onde o objetivo foi propor uma análise completa dos processos de medição de confiabilidade de software. São apresentadas tendências de medição *software* atuais, métricas de *software*, sendo proposto um novo quadro de medição de confiabilidade com base em métricas de *software*. O estudo apresenta as etapas do processo básico e do processo proposto para medição de confiabilidade, além de suas respectivas vantagens e desvantagens. No processo proposto, são duas etapas principais, a primeira é a de aplicação e teste, que consiste na utilização de modelos de confiabilidade, métricas semânticas. A segunda fase é a de validação de confiabilidade que visa verificar se o objetivo da confiabilidade foi atingido.

A predição de módulos propensos a falhas atrai muito interesse, devido ao impacto significativo na garantia de qualidade de software. Um dos objetivos mais importantes de tais técnicas é prever os módulos onde as falhas tendem a se esconder e busca-se fazer essa previsão o mais cedo possível no ciclo de vida de desenvolvimento. Tendo em vista este cenário Jiang et al. (2008) realizam um estudo comparativo do desempenho entre os modelos preditivos que usam métricas de nível de *design*, métricas no nível do código, e aquelas que usam os dois. Analisa-se um conjunto de treze dados do programa de métricas da NASA que oferecem modelos de métricas de código e de *design*. Ambos tipos de modelos provam ser úteis, pois podem ser utilizados em diferentes fases do processo de desenvolvimento.

Pantiuchina, Lanza e Bavota (2018) apresentam um estudo que visa investigar empiricamente se as métricas de qualidade são capazes de capturar a melhoria da qualidade do código conforme a percepção dos desenvolvedores. Para estabelecer um comparativo de qualidade a partir da percepção dos desenvolvedores e a aplicação das métricas, foi realizado perguntas aos usuários e medição de qualidade, através de aplicação das métricas. O estudo mostra que há casos em que métricas de qualidade não são capazes de capturar a melhoria da qualidade conforme percebida pelos desenvolvedores. Um exemplo disto foi quando o desenvolvedor afirma que “melhorou a coesão da classe C”, mas não foi constatada a melhoria de qualidade através da aplicação da métrica.

Souza et al. (2017) apresentam um estudo que visa verificar a eficácia dos valores de referências das métricas para detecção de *bad smells*. No estudo são utilizados dezoito métricas e seus valores de referências para detecção de cinco *bad smells* em doze *softwares*. O resultado destas métricas é comparado com os resultados obtidos pelas ferramentas *JDeodorant* e *JSPiRIT*, usados para identificar *bad smells*. Com base nos resultados obtidos, pode-se dizer que as métricas foram significativamente eficazes no apoio à detecção de *bad smells*.

4. Metodologia

A pesquisa apresentada neste documento é do tipo quantitativa. Este estudo é quantitativo porque busca realizar uma análise comparativa da qualidade do código-fonte de diferentes classes de navegadores desenvolvidos para a plataforma de dispositivos *mobile Android*. A medição da qualidade se dá através da aplicação de métricas quantitativas de código-fonte.

4.1. Métricas

GQM é utilizado para auxiliar na seleção das métricas. Com a aplicação do GQM chegamos ao objetivo de analisar a qualidade de código fonte de diferentes classes de navegadores com a finalidade de realizar uma análise comparativa da qualidade destas classes com relação a qualidade do *software* do ponto de vista dos envolvidos com o desenvolvimento e manutenção destes *softwares* no contexto de *softwares* livres. Com base neste objetivo foram levantadas as seguintes questões:

1. Qual a classe de navegador apresenta uma melhor qualidade das classes e funções?
2. Qual classe de navegador apresenta uma melhor distribuição do código por pacotes?
3. Qual classe de navegador apresenta uma melhor coesão?
4. Qual classe de navegadores apresenta um melhor acoplamento?
5. Levando em consideração a preocupação com a segurança, os navegadores focados em segurança apresentam melhor índices de qualidade?

Com base no objetivo e questões definidos, pode-se selecionar as métricas candidatas, sendo que as mesmas são apresentadas na Tabela 1. As métricas apresentadas nesta tabela são candidatas a serem utilizadas na avaliação dos navegadores. Para seleção métricas candidatas foi utilizado como referência o estudo realizado por Meirelles (2013), onde este estudo realiza um levantamento de métricas a serem utilizadas para avaliação de qualidade de código-fonte em *softwares* livre.

Tabela 1. Lista de Métricas Candidatas

Métricas	**Descrição**	**Classes das Métricas**	**Perguntas do GQM a serem respondidas**
CBO - *Coupling Between Objects*	Mede quantas classes são utilizadas pela classe analisada;	Métrica de Acoplamento	1, 2 e 5;
CF - *Coupling Factor*	Acoplamento é uma indicação das conexões entre elementos do projeto orientado a objeto;	Métrica de Acoplamento	1 e 2;
LCOM - *Lack of Cohesion between Methods*	Número de métodos que acessam um ou mais dos mesmos atributos;	Métrica de Estruturais	1 e 4;
NOC - *Number Of Children*	Número total de filhos de uma classe;	Métrica Estruturais	1 e 2

RFC - Response For a Class	Número de métodos dentre todos os métodos que podem ser invocados em resposta a uma mensagem enviada por um objeto de uma classe;	Métricas Estruturais	1 e 2;
WMC - Weighted Methods per Class	Soma ponderada de todos os métodos da classe;	Métrica Estruturais	1 e 2;
LOC - Lines Of Code	Número de linha que não seja linha em branco ou comentário, independentemente do número de declarações por linha;	Métrica de Tamanho	3;
LOC por pacote - *Lines Of Code per package*	Número de linhas médias por pacote;	Métrica de Tamanho	3;
LOC por linguagem - *Lines Of Code per programming language*	Número de linhas por Linguagens;	Métrica de Tamanho	3;
DIT - *Depth of Inheritance Tree*	Mede o número de ancestrais de uma classe;	Métrica Estruturais	1 e 2;

AHF - *Attribute Hiding Factor*	Razão entre a soma de todos os atributos herdados de todas as classes do sistema em consideração ao número total de atributos das classes disponíveis;	Métrica Estruturais	1 e 2;
AIF - *Attribute Inheritance Factor*	Razão entre a soma dos atributos herdados em todas as classes do sistema e o número total de atributos disponíveis na classe;	Métrica Estruturais	1 e 2;
MHF - *Method Hiding Factor*	Razão entre a soma de todos os métodos invisíveis em todas as classes em relação ao número total de métodos definidos em um determinado sistema;	Métrica Estrutural	1 e 2;
MIF - *Method Inheritance Factor*	Razão entre a soma dos métodos herdados em todas as classes e o número total de métodos disponíveis em todas as classes;	Métrica Estrutural	1 e 2;
PF - *Polymorphism Factor*	Razão entre o número atual de possibilidades de polimorfismos de uma classe e o número máximo de possíveis polimorfismos distintos da referida classe;	Métrica Estrutural	1 e 2;

4.2. Navegadores

Para a seleção dos navegadores foi realizado um levantamento dos navegadores para *Android* que possuem uma licença *open source*, sendo feito posteriormente uma classificação em navegadores focados em privacidade, segurança e tradicionais de acordo com suas características. Após a classificação foi realizado um levantamento dos dez navegadores mais utilizados de acordo com sua classificação.

A Tabela 2 apresenta os navegadores candidatos a serem utilizados neste estudo, juntamente do repositório do código-fonte de cada navegador. Foi realizado um levantamento dos navegadores de cada classe, onde foram selecionados os dez navegadores que obtiveram maior número de *downloads* de cada categoria. Durante a seleção dos navegadores foi realizado um levantamento dos navegadores na plataforma *Google Play*, plataforma responsável por disponibilizar *softwares*, onde após este levantamento foram selecionados os dez navegadores com maior número de *downloads*.

Tabela 2. Lista de Navegadores Candidatos

Navegador	Classes	Repositório do Código Fonte
F|L|OSS Browser	Tradicional	https://github.com/scoute-dich/browser.git
Firefox	Tradicional	https://hg.mozilla.org/mozilla-central
Lightning Browser	Tradicional	https://github.com/anthonycr/Lightning-Browser.git
Midori Web Browser	Tradicional	https://github.com/midori-browser/midori-android.git
Zirco	Tradicional	https://github.com/darvin/zirco-browser.git
Chromium	Tradicional	https://github.com/chromium/chromium.git
Kiwi Browser	Tradicional	https://github.com/kiwibrowser/android.git
Lucid Browser	Tradicional	https://github.com/powerpoint45/Lucid-Browser.git
Pale Moon	Tradicional	https://github.com/MoonchildProductions/Pale-Moon.git
JumpGo Browser	Tradicional	https://github.com/JTechMe/JumpGo.git
Keepass2Android	Privacidade	https://github.com/PhilippC/keepass2android.git
Lynket Browser	Privacidade	https://github.com/arunkuma

		r9t2/lynket-browser.git
Opera com VPN gratuita	Privacidade	https://operasoftware.github.io/upstreamtools/
Privacy Browser	Privacidade	https://git.stoutner.com/?p=PrivacyBrowser.git;a=summary
Tor-Browser	Privacidade	https://github.com/n8fr8/tor-android.git
IceCatMobile	Privacidade	https://f-droid.org/en/packages/org.gnu.icecat/
Waterfox	Privacidade	https://github.com/MrAlex94/Waterfox.git
Firefox Focus	Privacidade	https://github.com/mozilla-mobile/focus-android
Yuzu Browser	Privacidade	https://github.com/hazuki0x0/YuzuBrowser.git
Cliqz	Privacidade	https://github.com/cliqz-oss/browser-android.git
Fennec F-Droid	Segurança	https://github.com/f-droid/fdroidclient.git
Ungoogled Chromium	Segurança	https://github.com/Eloston/ungoogled-chromium.git
Firefox Nightly	Segurança	https://hg.mozilla.org/mozilla-central/
Iridium Browser	Segurança	https://github.com/iridium-browser/iridium-browser-dev.git
Kiwi Browser - Fast & Quiet	Segurança	https://github.com/kiwibrowser/android.git
Orfox Browser	Segurança	https://github.com/guardianproject/Orfox.git
Brave	Segurança	https://github.com/brave/browser-android-tabs.git
UFO Web Browser	Segurança	https://github.com/anthonycr/Lightning-Browser.git
Smart Browser - Free, Fast,	Segurança	https://github.com/scoute-

Secure Private Browser		dich/browser.git
Ducky Browser - Safe Browsing	Segurança	https://github.com/duckduckgo/android

4.3. Ferramentas Utilizadas

Uma das ferramentas utilizadas é o *Android Studio*, onde esta ferramenta é Ambiente de Desenvolvimento Integrado (IDE, do inglês *Integrated Development Environment*) para desenvolver para a plataforma *Android*, onde auxilia na medição das métricas. Com esta IDE e o *plugin MetricsReloaded* é possível a coleta das métricas utilizadas neste estudo. Por último, o *MetricsReloaded* é ferramenta que fornece métricas de código automatizadas para as plataformas de desenvolvimento baseadas em *IntelliJ IDEA* e *IntelliJ*. Esta ferramenta é utilizada para coleta das métricas selecionadas. Para realizar a etapa de análise estatística foi utilizado a ferramenta *RStudio*, onde este é um software livre de ambiente de desenvolvimento integrado para R, uma linguagem de programação para gráficos e cálculos estatísticos.

4.4. Etapas

Esta seção apresenta as etapas que foram executadas visando atingir o objetivo definido na introdução deste estudo. As etapas são listadas a seguir:

i. **Coleta dos códigos nos repositórios**: é realizada a coleta do código-fonte dos navegadores selecionados para realização deste estudo. Os códigos fontes são coletados e armazenados para apuração dos valores das métricas.

ii. **Preparação do ambiente**: realiza a instalação e configuração *Android Studio* e do *plugin* MetricsReloaded. Os códigos obtidos na etapa anterior são importados para o *Android Studio* e realizado testes iniciais.

iii. **Processamento do código para obtenção dos valores das métricas**: são realizados os cálculos das métricas através das ferramentas *Android Studio* e o *plugin MetricsReloaded*, sendo estas ferramentas apresentadas na etapa anterior. Nesta etapa gera-se um arquivo *Excel* com os valores das métricas por navegador.

iv. **Análise estatística**: os dados são analisados através da geração dos gráficos, onde será possível realizar a análise comparativa dos navegadores. Com isso será possível verificar qual das classes analisadas possui uma melhor qualidade de código fonte. Para realização da análise estatísticas será feito um levantamento de técnicas estatísticas a serem utilizadas, posteriormente será feita uma análise separada de cada classe de navegadores, buscando identificar diferentes resultados dentro de uma mesma classe de navegadores, após isto será realizado um comparativo entre as classes.

v. **Escrita do documento**: Após a análise dos dados, será realizado a documentação da análise e conclusão do estudo no texto a ser gerado neste estudo. O texto será atualizado, com o objetivo reforçar os resultados, as discussões e conclusões.

5. Resultados

Esta seção apresenta os resultados obtidos após a execução deste estudo. A partir dos resultados gerados a partir da execução desta pesquisa, busca-se verificar a qualidade do código fonte em diferentes classes de navegadores, através de análise estatística aplicadas em métricas de qualidade de código. Para verificar a qualidade do código dos navegadores, este estudo busca responder questões levantadas na subseção 4.1. As subseções apresentadas a seguir apresentam os resultados através dos gráficos gerados.

5.1. Estatística preliminares

5.1.1. Distribuição Cumulativas

Nesta subseção são apresentados os gráficos gerados da distribuição cumulativa das métricas coletada, sendo os gráficos apresentadas pelos conjuntos de métricas descritas na subseção 2.3 e agrupados de dois em dois gráficos, visando melhorar a apresentação e analise dos gráficos.

5.1.1.1. Distribuição Cumulativa das Métricas de Acoplamento

O primeiro conjunto de gráfico apresenta os gráficos gerados sobre as métricas *Attribute Hiding Factor* (AHF) e *Attribute Inheritance Factor* (AIF) pertencentes à categoria de acoplamentos.

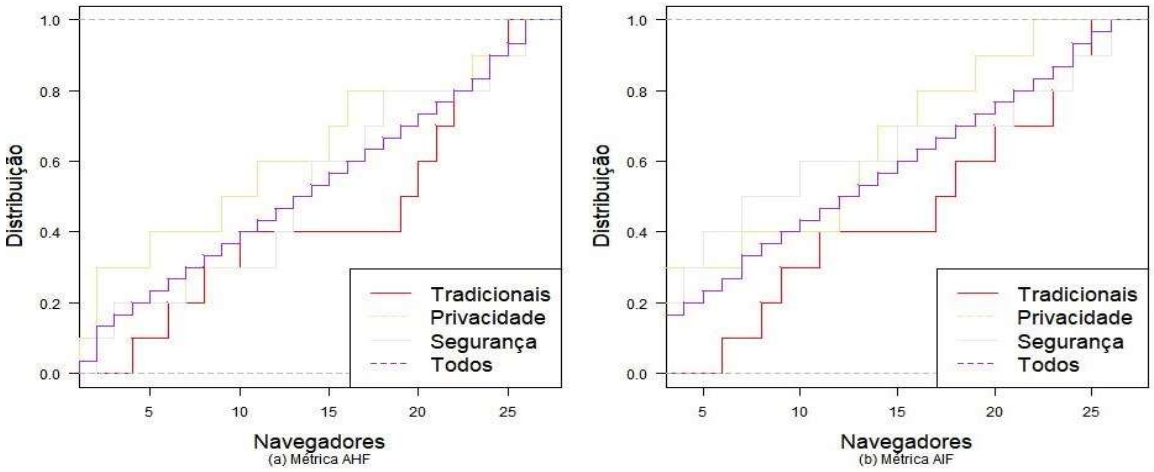

Figura 1: Gráficos de Distribuição Cumulativa das Métricas *Attribute Hiding Factor* (AHF) e *Attribute Inheritance Factor* (AIF)

Neste primeiro conjunto de gráficos pode-se verificar uma distribuição semelhantes nas duas métricas por parte das classes de navegadores. Onde os navegadores da classe tradicional apresentam uma distribuição mais baixa e os focados em segurança apresentam os maiores valores seguidos pelos navegadores focados em privacidade.

A seguir são apresentados os gráficos das métricas de *Coupling Factor* (CF) e *Method Hiding Factor* (MHF).

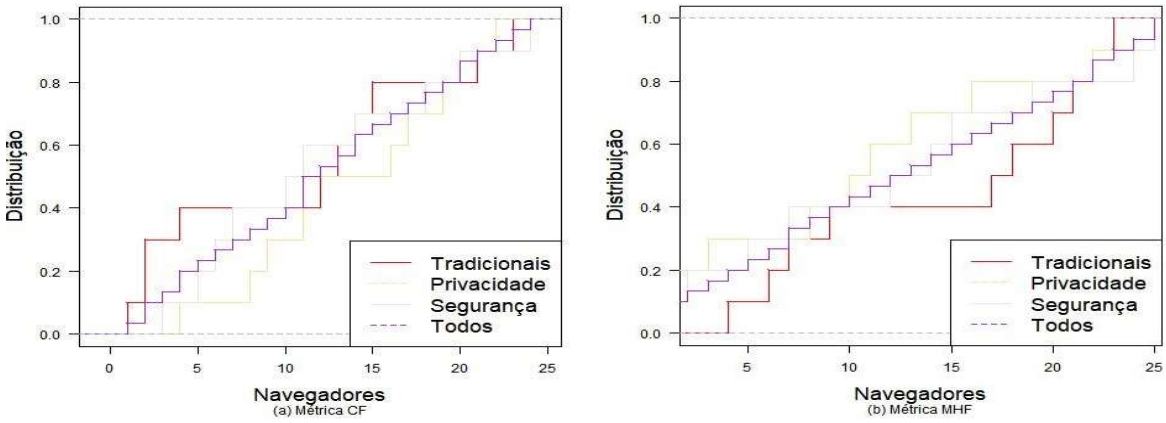

Figura 2: Gráficos de Distribuição Cumulativa das Métricas Coupling Factor (CF) e Method Hiding Factor (MHF).

O primeiro gráfico apresentado apresenta um comportamento distinto dos apresentados nos gráficos anteriores. Neste gráfico houve uma inversão das distribuições das classes, onde o que apresenta uma maior distribuição é a classe de navegadores tradicionais. Os que apresentaram os menores valores foram os focados em segurança, seguido pela classe de navegadores focados em privacidade. No gráfico de distribuição das métricas *Method Hiding Factor* (MHF) o comportamento apresenta-se aos dos gráficos apresentados na Figura 1.

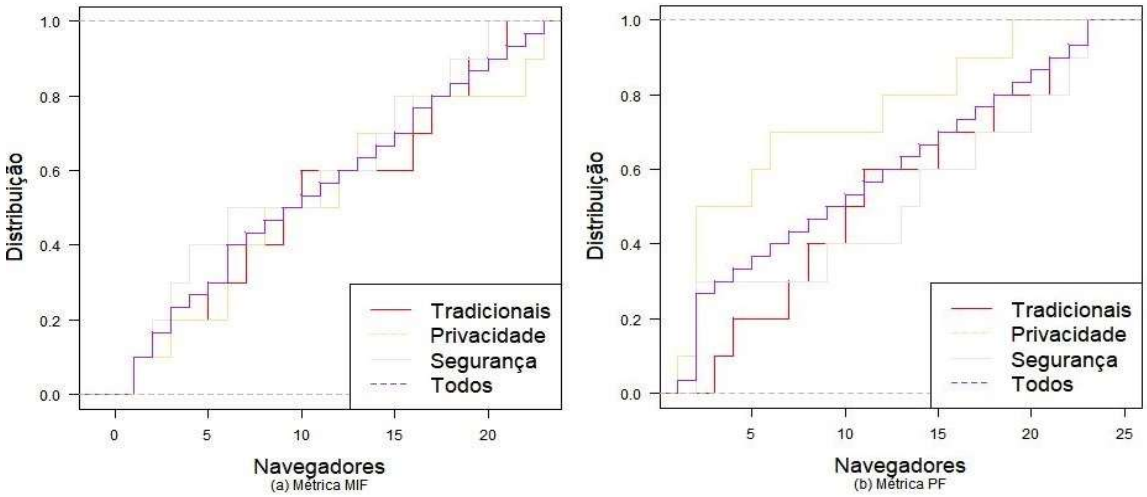

Figura 3: Gráficos de Distribuição Cumulativa das Métricas Method Inheritance Factor (MIF) e Polymorphism Factor (PF)

O primeiro gráfico apresentado na Figura 3 tem os valores das distribuições mais próximas, mas a classe de segurança apresenta uma maior distribuição, em contrapartida os valores das classes privacidade e tradicionais apresentam as menores distribuição.

5.1.1.2. Distribuição Cumulativa das Métricas de Tamanho

A seguir são apresentados os gráficos da distribuição das métricas de tamanho com os resultados das classes de navegadores separados em conjuntos de gráficos agrupados por dois gráficos por métricas. O primeiro gráfico apresenta as métricas de tamanho das linguagens de programação focados no *backend* do código fonte dos navegadores

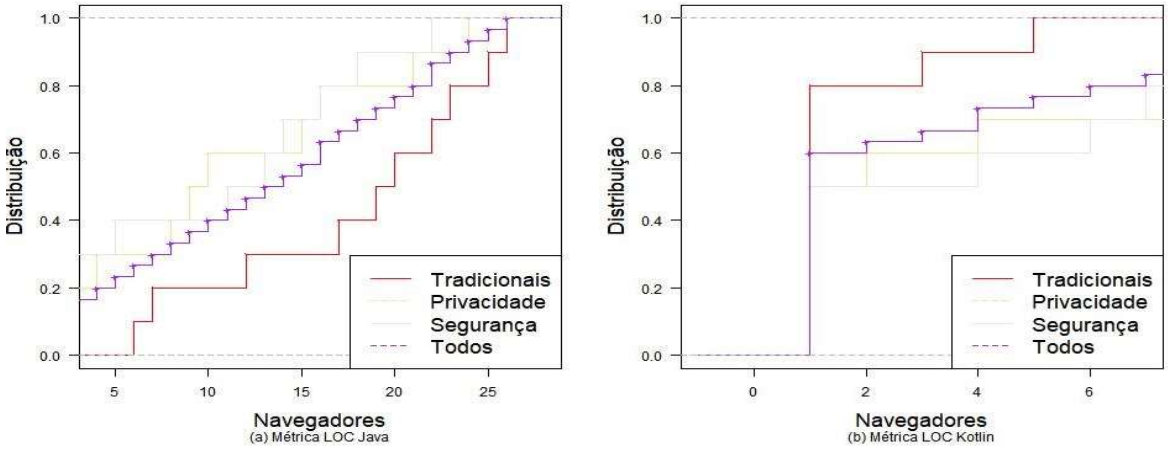

Figura 4: Gráficos de Distribuição Cumulativa das Métricas de Tamanho das Linguagens de *Backend*

Os gráficos das métricas das métricas de tamanho das linguagens de *backend* motra que na linguagem de programação Java os navegadores focados em segurança e privacidade apresentam as maiores distribuição. Já a classe de navegadores tradicional, possui um valor inferior, com resultados mais baixos dos que as demais classes. No gráfico das métricas de tamanho da linguagem de programação *Kotlin* mostra que os navegadores da classe tradicional apresentam maiores distribuição. As classes de segurança e privacidade apresentam menores distribuições.

A seguir são apresentados os gráficos das linguagens de *frontend* utilizadas no desenvolvimento dos navegadores.

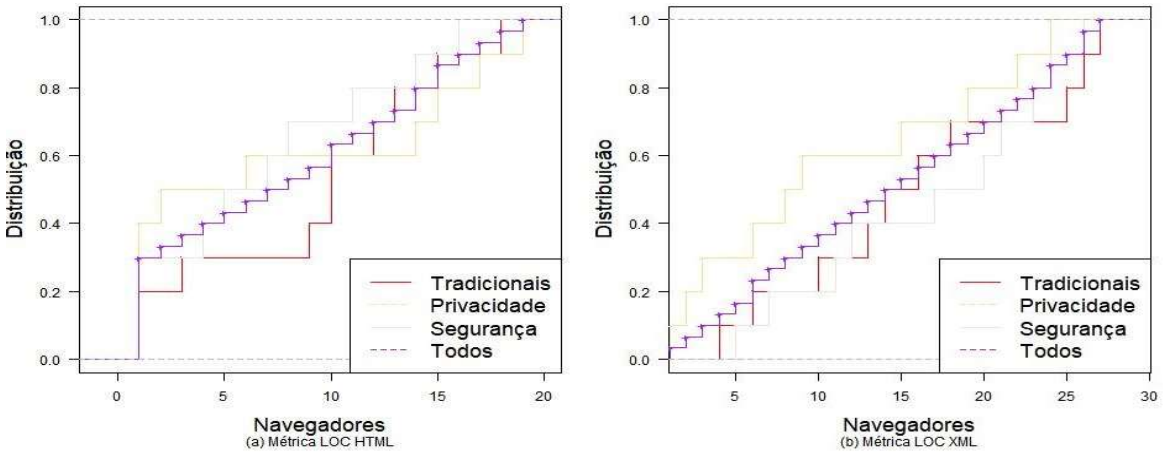

Figura 5: Gráficos de Distribuição Cumulativa das Métricas de Tamanho das Linguagens de *Frontend*

No primeiro gráfico apresentado na Figura 5 apresenta as distribuições da métrica de tamanho da linguagem de programação *HTML*. Neste gráfico, as classes de segurança e privacidade apresentam valores acima da distribuição de todas classes. Os navegadores da classe tradicionais apresentam valores abaixo das distribuições de todas classes na maior parte do gráfico.

Por último, será apresentado a seguir os gráficos referentes às métricas de tamanho e média de linhas de códigos por pacote.

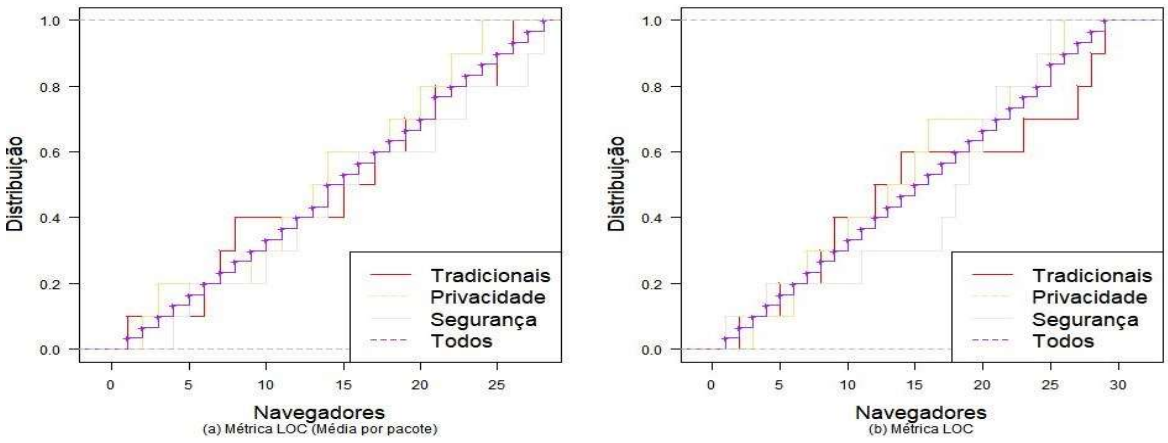

Figura 6: Gráficos de Distribuição Cumulativa das Métricas de Tamanho das Linguagens de *Frontend*

Neste último conjunto de gráficos de métricas focadas em tamanho, pode-se observar que os resultados das classes são muitos semelhantes. A única classe que apresentou valores diferentes das demais classes foi a de segurança no gráfico de métricas de *Lines Of Codes* (LOC), onde a distribuição apresentou menores valores.

5.1.1.3. Distribuição Cumulativa das Métricas Estruturais

A seguir são apresentados os dois primeiros gráficos da distribuição das métricas estruturais. O primeiro gráfico é referente a métrica *Coupling Between Classes* (CBO), seguido pelo gráfico da métrica *Depth of Inheritance Tree* (DIT).

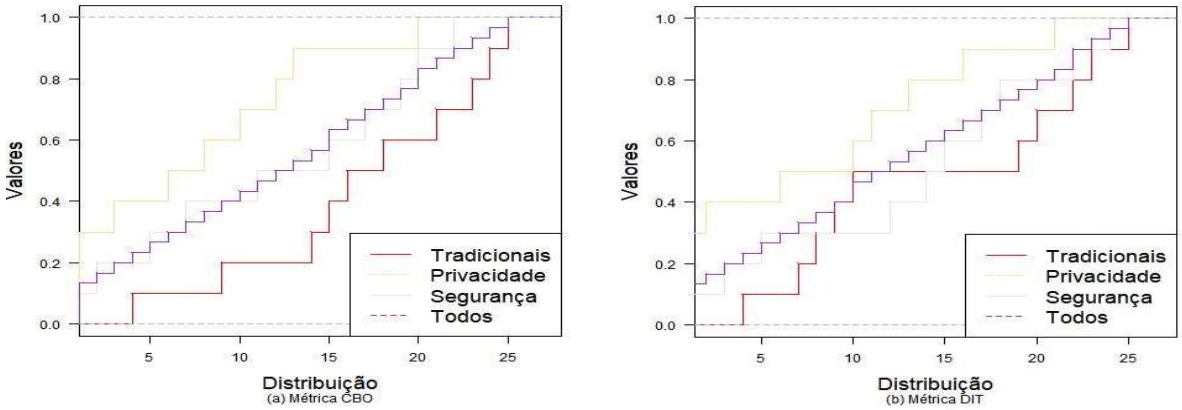

Figura 7: Gráficos de Distribuição Cumulativa das Métricas CBO e DIT

No primeiro gráfico pode-se verificar um maior distanciamento das classes tradicionais e de segurança em relação a distribuição de todas as classes. A classe de segurança apresenta um valor maior do que quando comparado com a distribuição de todas as classes. Em oposto, a classe tradicional apresenta uma distribuição menor comparado a todas as classes e o resultados das outras classes. O gráfico da métrica de *Depth of Inheritance Tree* o comportamento é semelhante ao analisado anteriormente, porém os resultados da classe de segurança e tradicional estão mais próximos em relação aos resultados da distribuição feita para todas as classes.

A seguir são apresentados os gráficos de distribuição das métricas *Lack of Cohesion Between Methods* (LCOM) e *Number Of Children* (NOC).

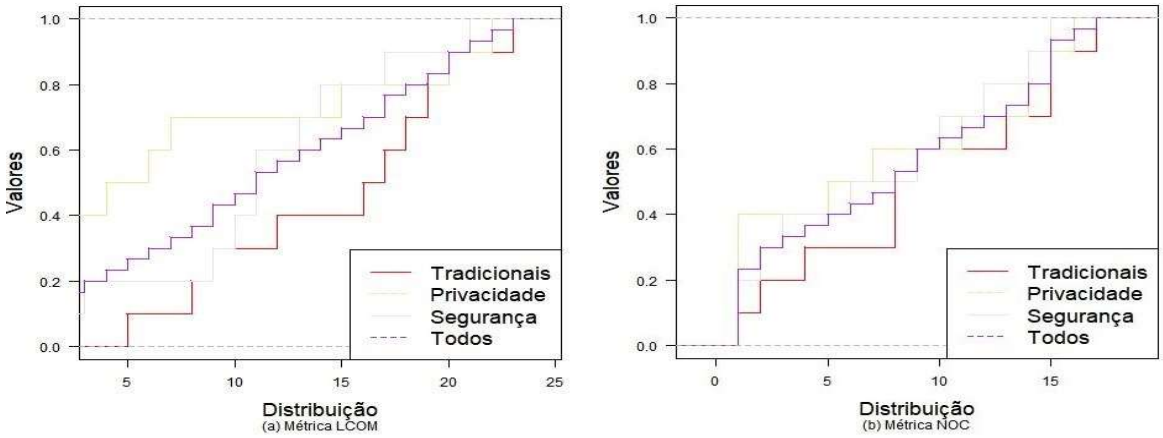

Figura 8: Gráficos de Distribuição Cumulativa das Métricas LCOM e NOC

A distribuição da métrica LCOM mostra uma maior distribuição desta métrica nas classes de segurança. Já a métrica tradicional apresenta uma baixa distribuição da métrica LCOM. Este comportamento se repete no gráfico de distribuição da métrica NOC.

Percebe-se que em termos estruturais, a classe de segurança apresenta uma maior distribuição e os navegadores tradicionais a menor distribuição. O que indica que há uma melhor distribuição destas métricas nos navegadores tradicionais.

5.1.2. Correlações e Desvio Padrão

Esta subseção apresenta os resultados das relações entre as métricas, onde visa constatar as métricas que possuem uma relação entre elas. Após estabelecer a relação entre as métricas, será apresentado o desvio padrão das métricas.

5.1.2.1. Correlação

Para apresentar a correlação das métricas foi calculado as médias das correlações de todas as métricas agrupadas por classes de navegadores. Além das médias é apresentado o desvio padrão destas métricas.

Métricas	**Tradicional**		**Privacidade**		**Segurança**		**Todas**	
	Média	**Desvio Padrão**	**Média**	**Desvio Padrão**	**Média**	**Desvio Padrão**	**Média**	**Desvio Padrão**
LOC (Média por pacote)	0,39	0,28	0,33	0,25	0,54	0,36	0,38	0,27
L(Groovy)	0,34	0,23	0,28	0,26	0,30	0,24	0,25	0,22
L(HTML)	0,39	0,26	0,42	0,26	0,57	0,35	0,45	0,27
L(J)	0,62	0,22	0,49	0,26	0,56	0,37	0,50	0,31
L(KT)	0,24	0,23	0,40	0,27	0,22	0,24	0,26	0,22
L(XML)	0,47	0,25	0,34	0,30	0,56	0,36	0,41	0,30
LOC	0,58	0,24	0,38	0,30	0,58	0,35	0,47	0,31
AHF (%)	0,27	0,27	0,38	0,25	0,32	0,23	0,21	0,26
AIF (%)	0,59	0,28	0,49	0,22	0,54	0,30	0,43	0,27
CF (%)	0,65	0,25	0,49	0,26	0,55	0,37	0,48	0,32
MHF (%)	0,51	0,24	0,35	0,28	0,24	0,29	0,23	0,27
MIF (%)	0,60	0,27	0,32	0,23	0,42	0,21	0,32	0,23
PF (%)	0,62	0,28	0,36	0,22	0,55	0,28	0,44	0,24
CBO – Média	0,39	0,30	0,52	0,25	0,52	0,34	0,46	0,26
DIT – Média	0,56	0,31	0,38	0,26	0,35	0,26	0,35	0,22
LCOM – Média	0,39	0,31	0,43	0,32	0,32	0,28	0,35	0,28
NOC – Média	0,46	0,30	0,44	0,30	0,53	0,31	0,42	0,27
RFC – Média	0,58	0,29	0,41	0,30	0,28	0,30	0,28	0,31
WMC – Média	0,37	0,28	0,44	0,33	0,33	0,27	0,33	0,29

Tabela 1: Média e Desvio Padrão das Métricas por Classes de Navegadores

Considerando as métricas de tamanho, observa-se que em média as classes de navegadores de segurança apresentaram os maiores resultados nas médias e desvio padrão. As classes de privacidade e tradicionais apresentaram os menores resultados nas médias e desvio padrão das correlações. Nos resultados das métricas de acoplamento apresentaram os maiores valores nas classes tradicional e segurança nas médias e desvio padrão respectivamente. Em média os menores valores da média e desvio padrão apresentaram na classe de privacidade. As métricas estruturais em média tiveram os maiores valores para as médias e desvio padrão nas classes tradicionais. Os menores resultados apresentaram-se na classe de segurança.

5.2. Resultados

Para esta subseção foram calculados as médias de cada métricas para cada classe de navegadores, onde os resultados serão apresentados por categorias das métricas.

5.2.1. Métricas de Acoplamento

As métricas de acoplamento visam apresentar o quão uma classe está ligada a outra. Para isto, será apresentado a seguir dois gráficos gerados para as métricas AHF e AIF.

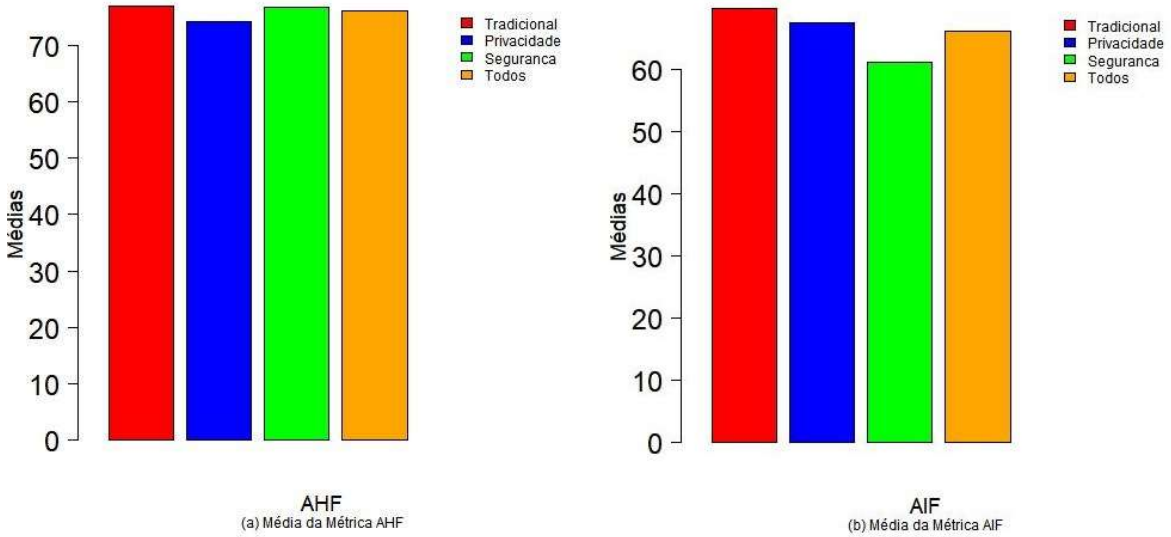

Figura 9: Gráficos das Médias das Métricas AHF e AIF

Na Figura 9 pode-se observar que os valores de cada classe estão próximos, além dos resultados apresentados estarem bem semelhantes. No gráfico que apresenta a métrica AHF, pode-se verificar que a maior média é o apresentado pela classe dos navegadores tradicionais. O menor valor obtido foi pela classe dos navegadores focados em privacidade. No gráfico responsável por apresentar os resultados da métrica AIF, percebe-se que a classe tradicional novamente apresenta o maior resultado. Considerando o menor valor, a classe de segurança foi o responsável por apresentar o menor valor.

Considerando que as métricas de acoplamento, os menores valores são os que apresentam melhor efeito de qualidade do código. Com isto, os piores resultados apresentados em ambas as métricas foi o da classe tradicional. Em relação os melhores valores, cada métrica apresentou uma classe com os melhores resultados. Na métrica AHF, a classe que apresentou o melhor resultado foi o de privacidade. Em relação à métrica AIF, o melhor resultado foi obtido pela classe de segurança.

Os próximos gráficos apresentados na Figura 10 são as médias obtidas das métricas CF e MHF.

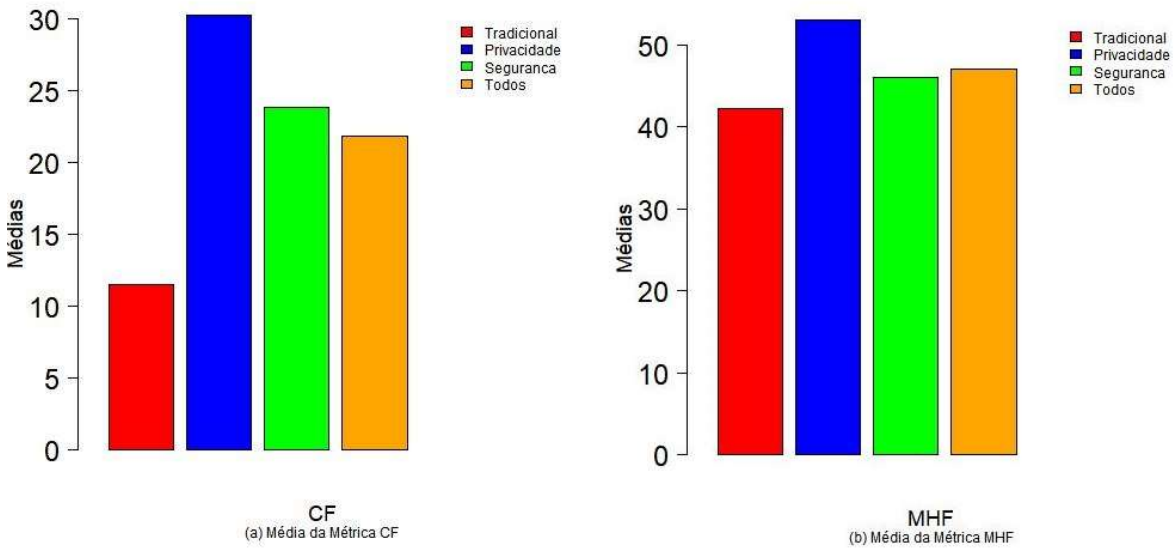

Figura 10: Gráficos das Médias das Métricas CF e MHF

Analisando os gráficos pode-se verificar que em ambas as métricas a classe dos navegadores focados em privacidade apresenta os maiores valores. Em relação aos menores valores, a classe tradicional apresentou os menores valores em ambas as métricas. Considerando os valores obtidos por estas métricas, pode-se considerar a classe tradicional obteve os melhores resultados. A classe dos navegadores focados em privacidade apresentou os piores resultados, sendo necessário realizar uma melhoria destes resultados.

Por último são apresentados os resultados das métricas MIF e PF. Estes gráficos são apresentados na Figura 11.

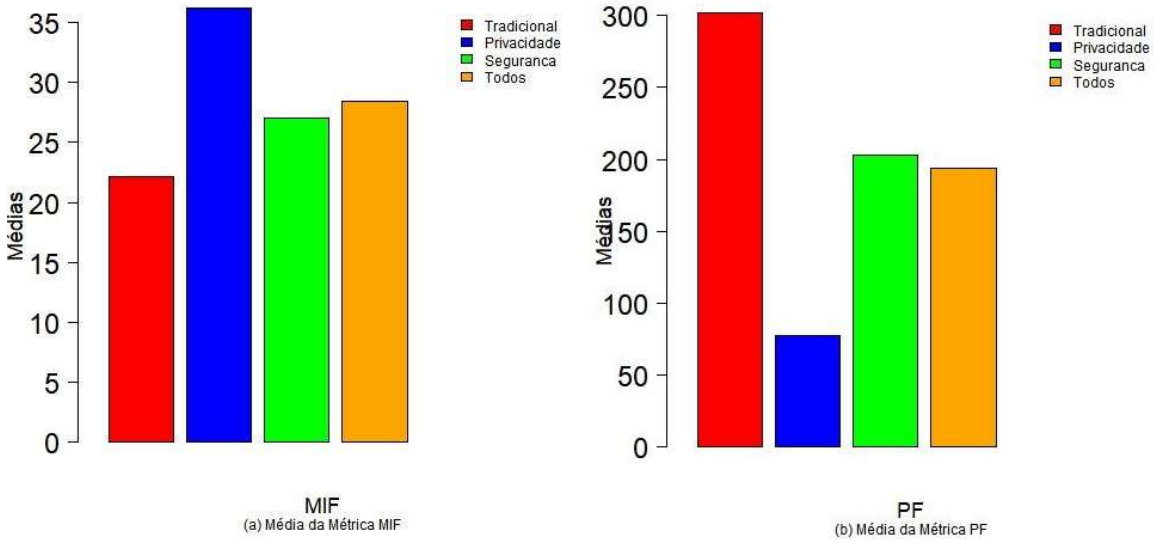

Figura 11: Gráficos das Médias das Métricas MIF e PF

No gráfico da Métrica MIF, o maior valor obtido foi da classe de privacidade, em contrapartida a classe tradicional obteve um menor resultado. Na métrica PF, diferente da métrica anterior, a classe tradicional apresentou o maior resultado. O resultado obtido pela classe de privacidade foi o que apresentou o menor valor. Nas métricas MIF e PF, as classes tradicional e privacidade apresentaram os melhores valores respectivamente. Em relação aos

valores que apresentaram nas métricas MIF e PF são as classes de privacidade e tradicional respectivamente.

5.2.2. Métricas de Tamanho

Estas métricas buscam apresentar a distribuição do código-fonte das classes analisadas neste estudo. Primeiramente são apresentados os gráficos das linguagens do *Backend* utilizados no código-fonte dos navegadores.

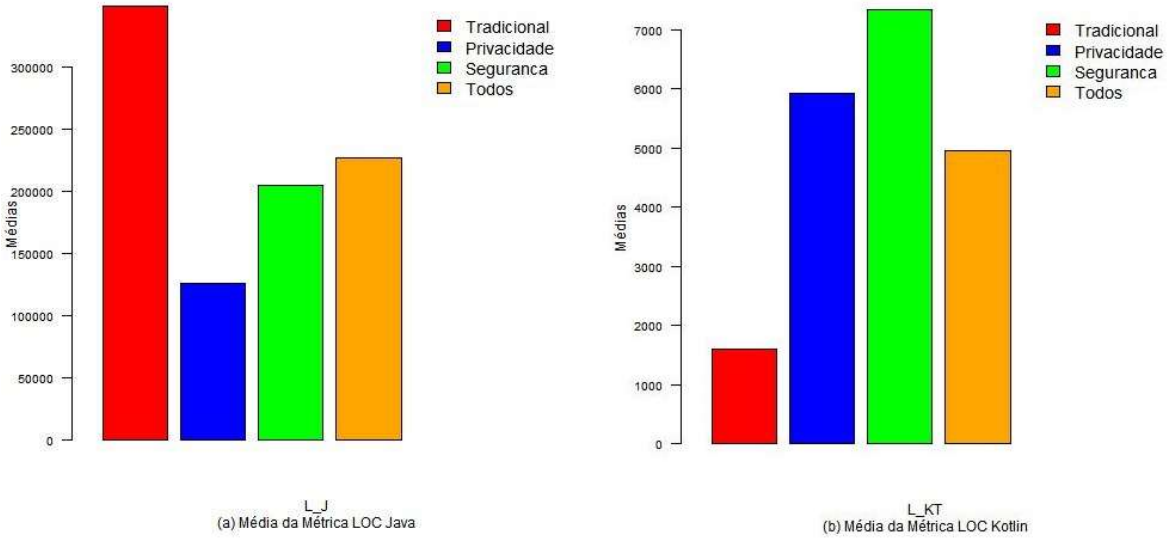

Figura 12: Gráficos das Médias das Linguagens de *Backend*

Com estes resultados pode-se notar que os códigos de *backend* dos navegadores da classe tradicional estão concentrados na linguagem *Java*. Enquanto isso, os navegadores focados em segurança e privacidade possuem seus códigos de *backend* conscentrados na linguagem de programação *Kotlin*. Apesar de atualmente o desenvolvimento *Android* estar crescendo, a linguagem de programação Java pode facilitar o entendimento do código fonte para quem está iniciando no desenvolvimento *Android*. Com isso, a classe tradicional pode ter uma vantagem para atrair novos desenvolvedores para o desenvolvimento de seus códigos.

A Figura 13 apresenta os resultados dos códigos de *frontend* da plataforma *Android*.

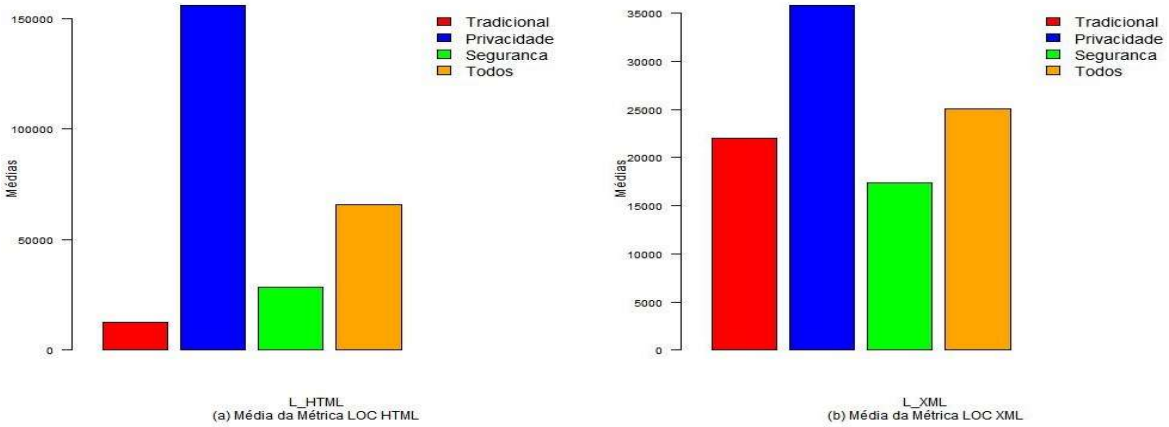

Figura 13: Gráficos das Médias das Linguagens de *Frontend*

A linguagem *XML* apresentou maiores resultados. A classe dos navegadores focados em privacidade estão melhor distribuídos em ambas as linguagens. As demais classes estão concentradas na linguagem XML.

O último grupo de métricas, apresentadas na categoria de métricas de tamanho são as métricas relacionadas a linhas de código.

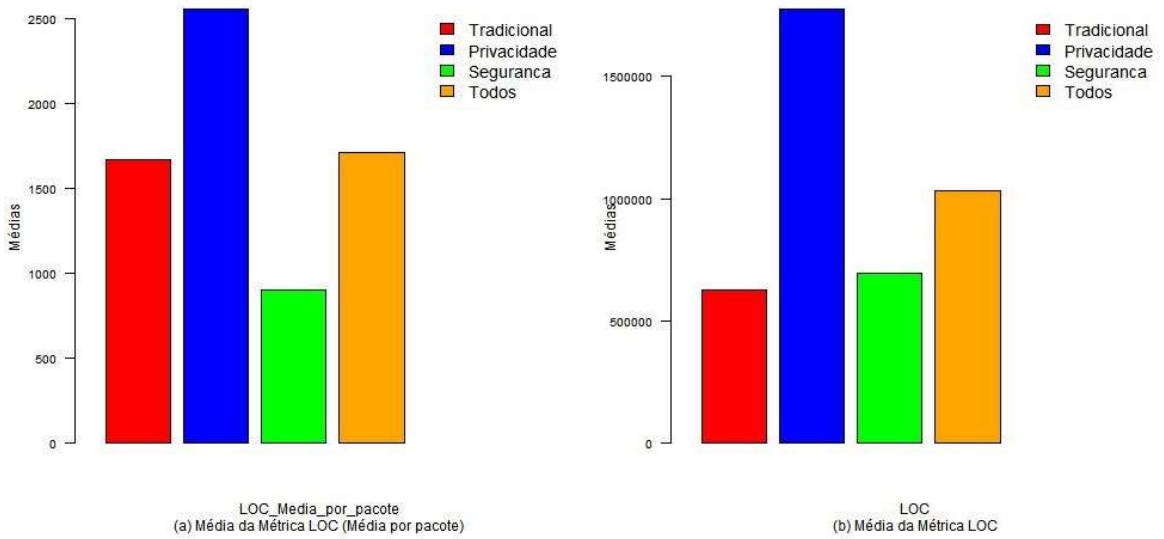

Figura 14: Gráficos das Médias de LOC

Em ambos os gráficos apresentam resultados semelhantes, onde percebe-se que todas as classes de navegadores estão bem distribuídas entre os pacotes. Em relação ao número de linha de código fonte, percebe-se que a classe dos navegadores focados em privacidade possui o maior valor.

5.3. Métricas Estruturais

Estas métricas têm como objetivo mensurar questões estruturais do código. A apresentação dos resultados é feita nas figuras que apresentam as métricas de dois em dois. O primeiro gráfico apresenta os resultados das métricas CBO e DIT.

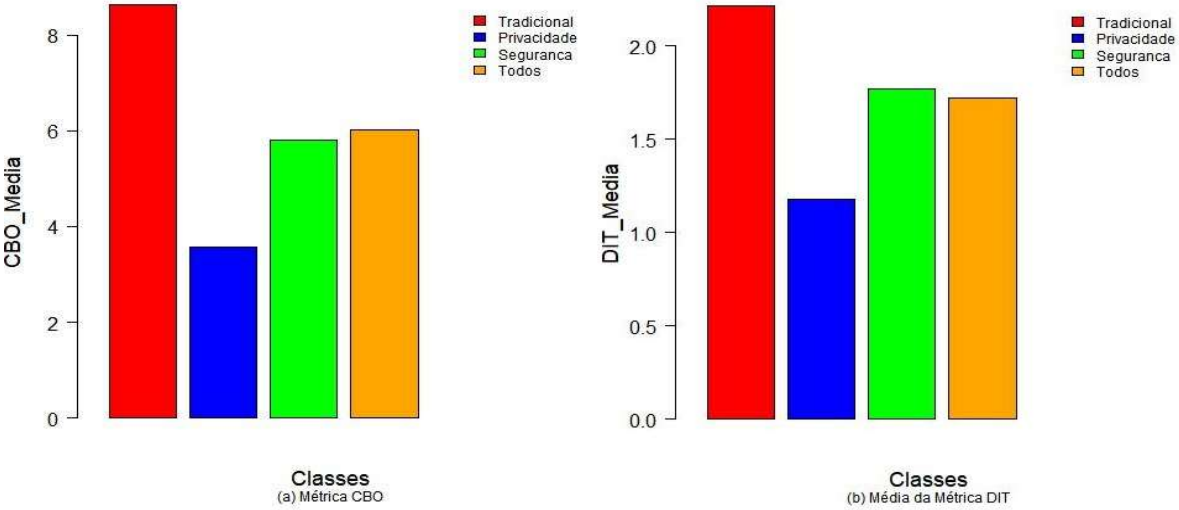

Figura 15: Gráficos das Médias CBO e DIT

Ambos os gráficos apresentaram valores semelhantes. A classe tradicional apresentou os maiores valores em ambas as métricas. A classe de privacidade apresentou os menores valores em ambas as métricas. Tendo em vista, os resultados de ambas métricas, observa-se que os navegadores tradicionais necessitam apresentar uma melhoria nestes pontos, já a classe de navegadores focados em privacidade, os resultados coletados se apresentaram melhor em ambas as métricas.

A seguir são apresentadas as métricas LCOM e NOC. Os resultados apresentam-se na Figura 14.

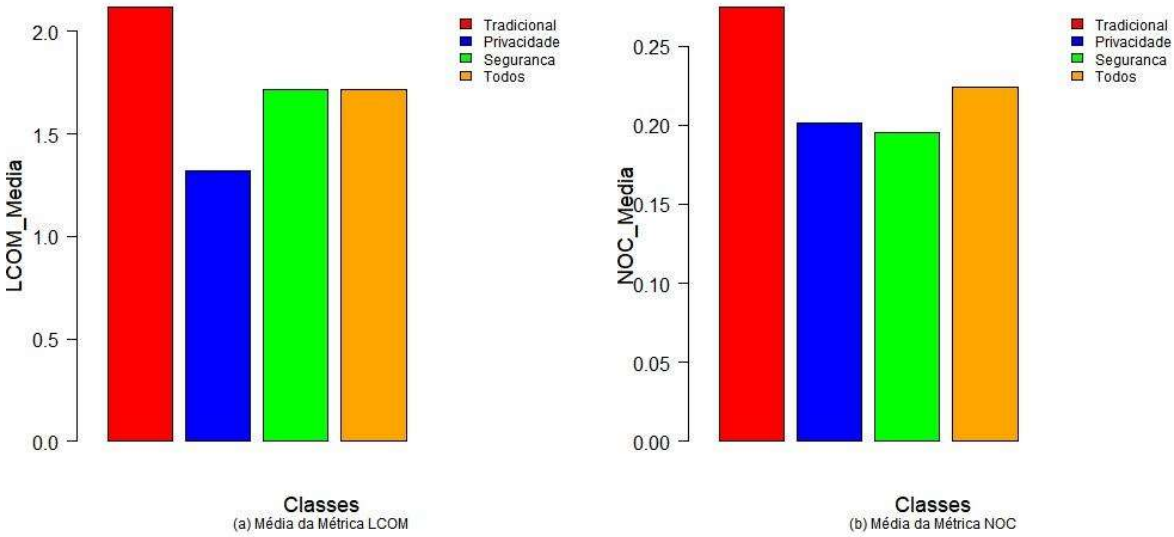

Figura 16: Gráficos das Médias LCOM e NOC

Nestes resultados a classe tradicional apresentou os maiores valores em ambas as métricas. Na métrica LCOM, a classe de navegadores focados em privacidade apresentou os menores resultados, já na métrica NOC a classe de segurança foi a que obteve os menores resultados. A LCOM é uma métrica que mede coesão de uma classe, com isso a classe tradicional apresenta um melhor resultado, e a classe de privacidade apresenta resultados que devem ser melhorados. Já na métrica NOC, o valor alto indica que é necessário um maior cuidado ao realizar edições em um código com esta métrica alta. Tendo em vista o impacto de um valor alto desta métrica, percebe-se que a classe tradicional apresenta o pior resultado, enquanto a classe de segurança apresentou os melhores resultados.

A Figura 14 apresenta os resultados das métricas EFC e WMC.

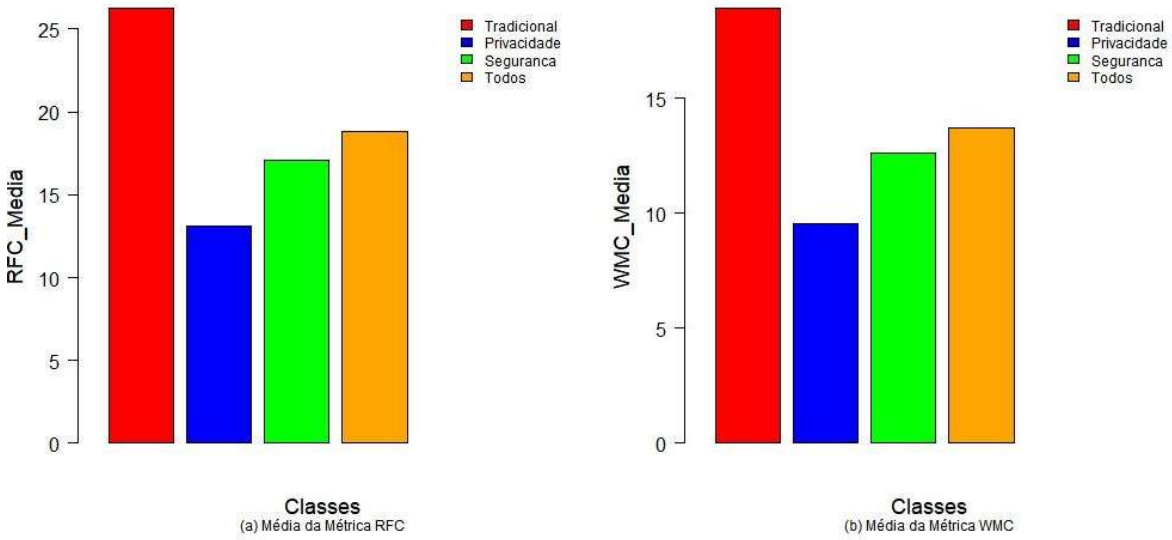

Figura 17: Gráficos das Médias RFC e WMC

Na métrica RFC, valores altos indicam que será necessária uma força de trabalhos nos testes maior, por isso quanto menor o resultado desta métrica melhor. Com base nestes resultados, pode-se verificar que a classe de privacidade apresenta o melhor resultado, em contra partida, a classe tradicional possui os maiores valores. Com relação a métrica WMC, altos valores indicam que as classes se tornam mais complexas e difíceis de serem reutilizadas. Tendo em vista este conceito, a classe tradicional apresenta o maior valor, indicando que este resultado deve ser melhorado. A classe dos navegadores focados em privacidade apresentaram o melhor resultado nesta métrica.

5.3. Analise

Nas duas principais categorias de métricas a classe tradicional apresentou resultados a serem melhorados. Já os navegadores focados em privacidade apresentaram resultados a serem melhorados nas métricas de acoplamento e os melhores índices nas métricas estruturais. Por último, os navegadores focados em segurança se apresentaram com resultados mais constantes em todas as categorias de métricas.

A principal motivação da realização deste estudo é a de verificar se os navegadores focados em segurança são os que apresentam melhores resultados na qualidade do código-fonte, pelo fato de estarem focados em segurança. Com base neste nos resultados obtidos pode-se verificar que estes navegadores apresentaram os resultados constantes, mas que precisam melhorar a qualidade do código fonte.

Referências

ALBESON, E. Frank. SEAN, Robi. ORTIZ, C. Erique. Android em Ação. 3. ed. Rio de Janeiro: Elsevier, 2012.

Android Studio (2019) “Android Studio” Disponível em: <https://developer.android.com/studio/?hl=pt-br> Acesso em: 24/04/2019

AMARA, Dalila. RABAI, Latifa Ben Arfa. 2017 “Towards a new framework of software reability measurement based on software metrics" Procedia Computer Science 2017. pp. 81-90.

AMRUTKAR, Chaitrali. Traynor, Patrick. Oorschot, Paul C. van (2010) "*An Empirical Evaluation of Security Indicators in Mobile Web Browsers*" in *IEEE Transactions on Mobile Computing*, vol. 14, no. 5, pp. 889-903, May 2015.

Ó Cinnéide, M., Hemati Moghadam, I., Harman, M. (2017) "*An experimental search-based approach to cohesion metric evaluation*" Empir *Software* Eng 22: 292–329.

DE FIGUEIREDO, João Pedro Pacheco. "Indicadores de Desempenho em Equipes de Desenvolvimento de *Software*". 2018. 103f. Mestrado Integrado em Engenharia Informática e Computação - Faculdade de Engenharia da Universidade do Porto, Porto, 2018.

Equipe Dub Soluções. (2017) "Estatísticas de uso de aplicativos no Brasil", Disponível em: <https://www.dubsolucoes.com/single-post/estatisticas-de-uso-de-aplicativos-no-Brasil> Acesso em: 24 fev. 2019.

International Organization for Standardization (2011). ISO/IEC 25010:2011, *Systems and software engineering -- Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models, International Organization for Standardization*.

JIANG, Yue. CUKI, Bojan. MENZIES, Tim. BARTLOW, Nick. 2008. "*Comparing Design and Code Metrics for Software Quality Prediction*" *In Proceedings of the Fourth International Workshop on Predictor Models in Software Engineering. New York, NY,* pp. 11-18

JÚNIOR, Marcos Ronaldo Pereira. "Estudo de métricas de código fonte no sistema *Android* e seus aplicativos" 82f. Trabalho de Conclusão de Curso - Graduação em Engenharia de *Software* - Universidade de Brasília, Brasília

KAF, Ali Al. ISMAIL, Talal Al. Baggili, Ibrahim. Marrington, Andrew. (2018) "*Portable web browser forensics: A forensic examination of the privacy benefits of portable web browsers*" 2012 International Conference on Computer S*ystems and Industrial Informatics*, Sharjah, 2012, pp. 1-6.

PANTIUCHINA, Jevgenija. LANZA, Michele. BAVOTA Gabriele. 2018 "*Improving Code: The (Mis) Perception of Quality Metrics*" *IEEE International Conference on Software Maintenance and Evolutioin*. Madrid. pp. 80-91.

PRESSMAN, Roger S. MAXIM, R. Bruce. Engenharia de *Software*: uma abordagem profissional. 8. ed. São Paulo: Pearson Makron Books, 2016.

ROSHAN, Shashi. KUMAR, S Vinay. KUMAR, Manish. (2017) "Performance evaluation of web browsers in iOS platform" *2017 Third International Conference on Research in Computational Intelligence and Communication Networks (ICRCICN)*, Kolkata, 2017, pp. 74-78.

MDN WEB DOC. (2018) "Firefox" Disponível em: <https://developer.mozilla.org/pt-BR/docs/Mozilla/Firefox> Acesso em: 20 abr. 2019

MEIRELLES, Paulo R. Miranda. "Monitoramento de métricas de código-fonte em projetos de *software* livre". 2013. 161f. Tese de Doutorado em Ciência da Computação Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2013

MENDIVELSO, Luis F. Garcés, Kelly Casallas, Rubby (2018) "*Metric-centered and technology-independent architectural views for software comprehension*" Journal of *Software* Engineering Research and Development, 2018

MOHAN, Michael. (2018) "*A survey of search-based refactoring for software maintenance*" Disponível em: <https://jserd.springeropen.com/articles/10.1186/s40411-018-0046-4> Acesso em: 24 fev. 2019

Mozilla Firefox Focus. (2017) "O que é o Firefox Focus?", Disponível em: <https://support.mozilla.org/pt-BR/kb/o-que-e-o-firefox-focus> Acesso em: 24 fev. 2019.

SATO, Danilo Toshiaki. "Uso Eficaz de Métricas em Métodos Ágeis de Desenvolvimento de *Software*". 2007.155f. Dissertação de Mestrado em Ciências - Instituto de Matemática e Estatística da Universidade de São Paulo, São Paulo, 2018.

SOMMERVILLE, Ian. Engenharia de *Software*. 9. ed. São Paulo: Pearson Addison Wesley, 2011.

Souza, Priscila P. and Sousa, Bruno L. and Ferreira, Kecia A. M. and Bigonha, Mariza A. S. 2017. "Applying Software Metric Thresholds for Detection of Bad Smells" In Proceedings of the 11th Brazilian Symposium on Software Components, Architectures, and Reuse (SBCARS '17). ACM, New York, NY, USA, Article 6, 1-10 pp.

STATCOUNTER. (2019) "Website analytics made easy." Disponível em: <https://statcounter.com/> Acesso em: 20 abr. 2019

STATCOUNTER STATISTICS. (2019) "Mobile & Tablet Browser Market Share Worldwide" Disponível em: <http://gs.statcounter.com/browser-market-share/mobile-tablet/worldwide/#yearly-2018-2018-bar> Acesso em: 20 mar. 2019

STERLING, C. Managing Software Debt: Building for Inevitable Change. 1. Ed. New Jersey: Addison-Wesley, 2010.

TANENBAUM, A. S. Redes de Computadores 5ª ed. São Paulo: Pearson, 2011.

Tor. (2017) "What is Tor Browser?", Disponível em: <https://www.torproject.org/projects/torbrowser.html.en> Acesso em: 24 fev. 2019.

TOURE, Fadel. (2018) "A metrics suite for JUnit test code: a multiple case study on open source *software*" Journal of Software Engineering Research and Development, 2018 2ª ed.

W3COUNTER "Browser & Platform Market Share" Disponível em: <https://www.w3counter.com/globalstats.php> Acesso em: 20 mar. 2019

YE, Peng. (2010) "Research on mobile browser's model and evaluation" 2010 IEEE 2nd Symposium on Web Society, Beijing, 2010, pp. 712-715.