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 range of web browsers available nowdays, little is known about the quality of these browsers' source code. Poorly written or designed software can cause inconvenience to users, involving aspects such as security and privacy. This study aims to perform a comparative analysis of the source code quality of open source Android browsers. The study is done by applying browser source code quality metrics that are divided into three classes: traditional, security and privacy. Metrics and 30 browsers are surveyed. Metric analysis consist of correlation and the cumulative distribution graph. The results show that classes behave similarly in some cases, but browser classes focused on privacy and security achieved better source code quality results.*

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

Resumo. *Enquanto existe uma ampla diversidade de navegadores Web disponíveis atualmente pouco se sabe sobre a qualidade do código-fonte destes navegadores. 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 de navegadores que são divididos em três classes: tradicional, segurança e privacidade. São levantadas métricas e 30 navegadores. As análises das métricas consistem em correlação e do gráfico de distribuição acumulada. Os resultados mostram que as classes apresentam comportamento semelhantes em alguns casos, mas as classes de navegadores focados em privacidade e segurança obtiveram melhores resultados de qualidade de código-fonte.*

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 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.

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. A principal contribuição deste estudo é apresentar uma análise comparativa da qualidade do código-fonte de diferentes classes de navegadores através da correlação e distribuição acumulada.

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. A seção 5 apresenta os resultados deste estudo. Por último, a seção 6 apresenta a conclusão e trabalhos futuros.

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.

Para classificação dos navegadores, foram levantados cerca de 30 navegadores de cada classe, sendo classificados de acordo com a sua descrição no *Google Play Store*. Para classifica-los, a descrição fornecida pelo desenvolvedor foi analisada e classificada de acordo com as definições das classes apresentadas anteriormente. Posteriormente, foram selecionados os 10 navegadores de cada classe que obtiveram maior números de *downloads* no *Google Play Store*.

2.2. Qualidade de Software

A qualidade no contexto de produção e manutenção de *software* é importante [Pressman e Maxim 2016]. Tendo em vista a importância do 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.

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 ou qual o módulo com maior número de linhas de código. Além de permitirem 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*).

Métricas de coesão permitem medir a diversidade de assuntos que uma classe implementa [Meirelles 2013]. Altos valores de coesão indicam se o foco de uma classe está em um único aspecto do sistema [Meirelles 2013]. Enquanto uma baixa coesão indica que a classe trata de diferentes aspectos. Tendo em vista essa definição, uma classe deve ser coesa. Exemplos de métricas de coesão são ausência de coesão de métodos (LCOM, do inglês *Lack of Cohesion in Methods*) e complexidade estrutural (SC, do inglês *Structural Complexity*).

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 se discutem 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 valida 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 são utilizadas no presente estudo.

Júnior (2015) apresenta um estudo cujo 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. A principal contribuição desse estudo é o

levantamento das métricas a serem utilizadas em código-fonte de *softwares* para dispositivos *Android*.

Amara e Rabai (2017) apresentam um estudo cujo objetivo foi propor uma análise completa dos processos de medição de confiabilidade de software. São apresentadas tendências de medição *software* atuais, métricas de *software*, onde foi 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. Nesse estudo destaca-se entre as contribuições, a apresentação das tendências de medição e as métricas apresentadas.

A predição de módulos propensos a falhas atrai 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. Em relação a contribuição pode-se destacar o resultado deste estudo, onde percebe-se que ambos os modelos se apresentam de forma valida em cada etapa do processo de desenvolvimento de *software*.

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, para tanto, foram realizadas perguntas aos usuários e medição de qualidade, através de aplicação das métricas. O estudo mostra que há casos em que métricas de qualidade são capazes de capturar a melhoria da qualidade melhor do que a percebida pelos desenvolvedores. Um exemplo disto é apresentada quando o desenvolvedor afirma que "melhorou a coesão da classe C", mas não foi constatada a melhoria de qualidade através da aplicação da métrica. Tendo em vista o cenário, onde a métrica capturou uma melhoria na qualidade melhor do que a percepção do desenvolvedor, pode-se dizer que sua principal contribuição é ressaltar a importância e eficiência da aplicação das métricas.

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 trabalho é do tipo quantitativa, 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 das métricas candidatas foi utilizado como referência o estudo realizado por Meirelles (2013), no qual é realizado 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	Categoria das Métricas	GQM - Questões respondidas
CBO - *Coupling Between Objects*	Mede quantas classes são utilizadas pela classe analisada. À medida que o CBO aumenta, é possível que a reutilização de uma classe diminua. Altos valores de CBO também dificultam modificações e o teste resultante dessas modificações. Em geral, os valores de CBO para cada classe deverão ser mantidos o mais baixo possível. É consistente com a diretriz geral de reduzir o acoplamento em software convencional.	Acoplamento	1, 2 e 5.
CF - *Coupling Factor*	Acoplamento é uma indicação das conexões entre elementos do projeto orientado a objeto.	Acoplamento	1 e 2.
LCOM - *Lack of Cohesion between Methods*	Número de métodos que acessam um ou mais dos mesmos atributos. Se o resultado do LCOM for alto, métodos podem ser acoplados uns aos outros via atributos. Isso aumenta a complexidade do projeto de classe.	Coesão	1 e 4.
NOC - *Number Of Children*	Número total de filhos de uma classe.	Estrutural	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. Conforme a RFC aumenta, o trabalho necessário para o teste também aumenta porque a sequência de testes cresce.	Estrutural	1 e 2.
WMC - *Weighted Methods per Class*	Soma ponderada de todos os métodos da classe. valores altos indicam que uma classe pode ter muita responsabilidade. Isso reduzirá a reutilização da classe e dificultaram a implementação e o teste.	Estrutural	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. Neste estudo foi realizado o levantamento das linhas de código das linguagens utilizadas pelos navegadores, onde são divididas em linguagens de *backend* e *frontend*. As linguagens de *backend* utilizadas foram *Groovy* e *Kotlin*. As relacionadas ao *frontend* são *XML* e *HTML*.	Tamanho	3.
LOC por pacote	Número de linhas médias por pacote.	Tamanho	3.
LOC por linguagem	Número de linhas por Linguagens.	Tamanho	3.
DIT - *Depth of Inheritance Tree*	Mede o número de ancestrais de uma classe. Quanto maior o valor desta métrica, maior o número de métodos e atributos herdados, aumentando também a sua complexidade. Quando os resultados atingem o valor até dois é considerado como um bom resultado, já quando os resultados atingem o valor entre dois e quatro é considerado como regular, a cima de quatro é considerado um valor ruim. [Pressman e Maxim 2016]	Estrutural	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. O ideal é que não haja atributos públicos nas classes, sendo o valor ideal para a métrica é um valor próximo à 1. [Filó 2014]	Estrutural	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. Valores aproximados a 0 significa que não foi utilizada a herança de atributos na especificação, e quando o valor do índice tende a 1, significa que a herança de atributos é muito utilizada.	Estrutural	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. Quanto maior o número de métodos ocultos, maior será a aproximação da métrica do valor 1, indicando alto encapsulamento do sistema [Filó 2014].	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. Valores próximos de 0 indicam que a herança está sendo pouco utilizada. Isso minimiza o reuso e a abstração provida pela herança em softwares orientados por objetos.	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. Valores próximos de 1 indicam alta utilização de polimorfismo e valores próximos de 0 indicam baixa utilização desse recurso. Valore próximos a 1 são os mais indicados.	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 feita posteriormente uma classificação em navegadores focados em privacidade, segurança e tradicionais de acordo com suas características. A Tabela 2 apresenta os navegadores usados neste estudo. Foi realizado um levantamento dos navegadores de cada classe, onde foram selecionados os dez navegadores que obtiveram maior número de *downloads* de cada categoria no *Google Play Store*.

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/arunkumar9t2/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	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	Segurança	https://github.com/scoute-dich/browser.git
Ducky Browser	Segurança	https://github.com/duckduckgo/android

4.3. Ferramentas Utilizadas

Uma das ferramentas utilizadas foi o *Android Studio*, que auxilia na medição das métricas. Com o *Android Studio* e o *plugin MetricsReloaded* é possível a coleta das métricas. O *MetricsReloaded* é uma ferramenta que fornece métricas de código automatizadas para as plataformas de desenvolvimento baseadas em *IntelliJ IDEA* e *IntelliJ*. Essa ferramenta foi utilizada para coleta das métricas selecionadas. Para realizar a etapa de análise estatística foi utilizada a ferramenta *RStudio*, que é um ambiente de desenvolvimento integrado para R, uma linguagem de programação para gráficos e cálculos estatísticos.

4.4. Análise Estatística

As análises realizadas são baseadas em correlação, desvio padrão e distribuição acumulada. A correlação mede o grau da correlação entre duas variáveis, ou seja, visa verificar se existe uma relação entre as variáveis. Para calcular a correlação foi utilizado o coeficiente de correlação de Pearson que mede o grau da correlação linear entre duas variáveis quantitativas. A correlação de cada métrica foi calculada com as demais métricas, após os resultados coletados, os resultados negativos foram transformados em positivos, pois o intuito é verificar se existe uma correlação entre as métricas. Por último, foi calculada a média de todas as métricas com as demais para cada classe e para todas as classes. Com base nos resultados das médias, as métricas que obtiveram média superior a 0,40 na média de todas as classes, pois indica que esta métrica está alta correlacionada com as demais.

O desvio padrão é uma medida que expressa o grau de dispersão de um conjunto de dados, ou seja, é uma medida que indica o quanto o conjunto de dados é uniforme. O desvio padrão é calculado com base nos resultados das métricas de cada classe de navegadores. Com isto, o resultado apresentado é o desvio padrão das correlações das métricas de cada classe de navegadores.

A distribuição acumulada apresenta a probabilidade de um valor de uma variável x assumir determinados valores. Ou seja, descreve como probabilidades são associadas aos valores ou aos intervalos de valores de uma variável aleatória. Os resultados obtidos com a distribuição acumulada servirão como base para realizar as análises comparativas das classes de navegadores deste estudo.

5. Resultados

Nesta seção, são apresentados os resultados obtidos. Os resultados obtidos pelas correlações são apresentados na Tabela 1. A fim de melhor visualizar os resultados das métricas, recorreu-se aos gráficos de linhas para ilustrar a distribuição acumulada.

5.1. Correlação

Para calcular a correlação, foi coletada a correlação das métricas de cada navegador. Após isto foi calculada a média das métricas para cada classe de navegadores e a média das correlações de todas as classes. Esta medida foi utilizada para excluir as métricas com alta correlação com as demais, para isto foi estipulado como valor de exclusão as métricas no qual o valor for maior que 0,4 na média de todas as classes.

A Tabela 3, apresenta os resultados da média e o desvio padrão das correlações das métricas, onde as métricas marcadas com a cor vermelha indicam que não são utilizados por possuir alta correlação com as demais métricas, e com isso não sendo utilizadas nas análises. Em contrapartida as métricas marcadas na cor azul, indica que estas foram utilizadas nas análises.

Tabela 3. Média e Desvio Padrão das Métricas por Classes de Navegadores

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(Java)	0,62	0,22	0,49	0,26	0,56	0,37	0,50	0,31
L(Kotlin)	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	0,46	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

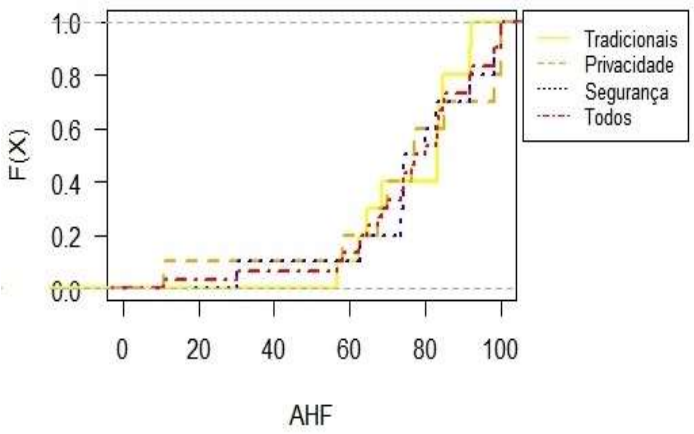
5.2. Distribuição Acumulada

Nesta subseção, são apresentados os gráficos gerados da distribuição acumulada das métricas coletadas. Os gráficos são apresentados separados pelas categorias de métricas, que são métricas estruturais, coesão, acoplamento e tamanho.

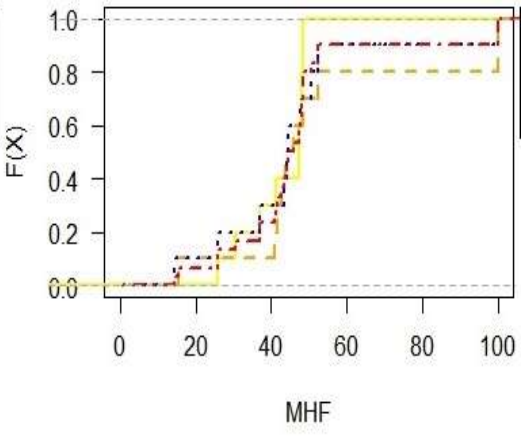
5.2.1. Distribuição Acumulada das Métricas Estruturais

A Figura 1 apresenta a distribuição acumulada das métricas estruturais das classes de navegadores. As Figuras 1(a). 1(b), 1(c) e 1(d) apresentam a distribuição das métricas AHF, MHF, MIF e PF respectivamente. Nestas métricas, as classes apresentam resultados semelhantes, onde estão concentrados em sua maior parte nos valores mais altos. Analisando as demais métricas estruturais, percebe-se que os resultados das classes estão diferentes. A Figura 1(e) apresenta os resultados relacionados a métrica RFC. A classe de navegadores tradicional possui 80% dos seus resultados concentrados nos valores entre 18 e 50. A classe dos navegadores focados em privacidade são os que apresentaram uma concentração dos resultados nos valores mais baixos, e consequente os melhores resultados. Cerca de 90% dos resultados estão concentrados nos valores entre 0 e 15. A classe de navegadores tradicionais necessita de melhorias nesta métrica, já a classe de privacidade apresentou os melhores resultados.

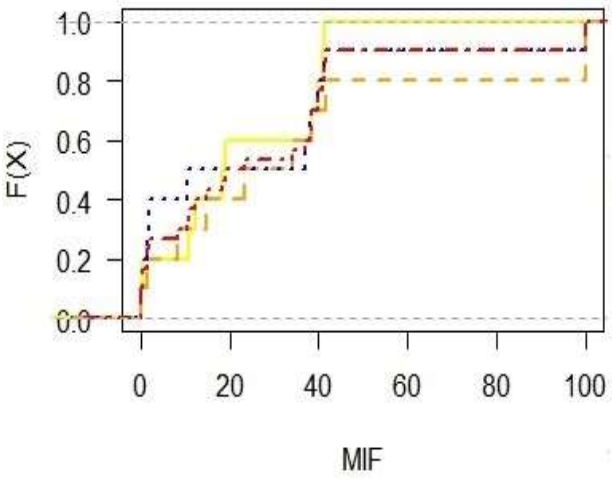
A Figura 1(f) apresenta os resultados da métrica WMC. Os valores da classe de navegadores tradicionais concentram os valores entre 10 e 40 em 80%, como na métrica RFC, os valores desta classe concentram-se nos maiores valores. Assim como a métrica RFC, a classe de privacidade apresentou os menores resultados, tendo 80% dos seus resultados concentrados em valores até 15. Com base nestes resultados, pode-se dizer que os navegadores tradicionais concentram valores altos para esta métrica, indicando que o código das classes é muito grande, necessitando distribuir o seu código em outras classes, dividindo a suas responsabilidades. A classe dos navegadores focados em privacidade apresentou os melhores valores para esta métrica.



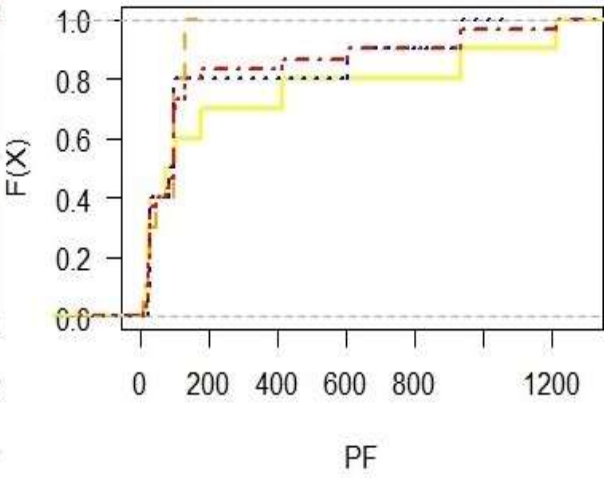
(a) Métrica AHF



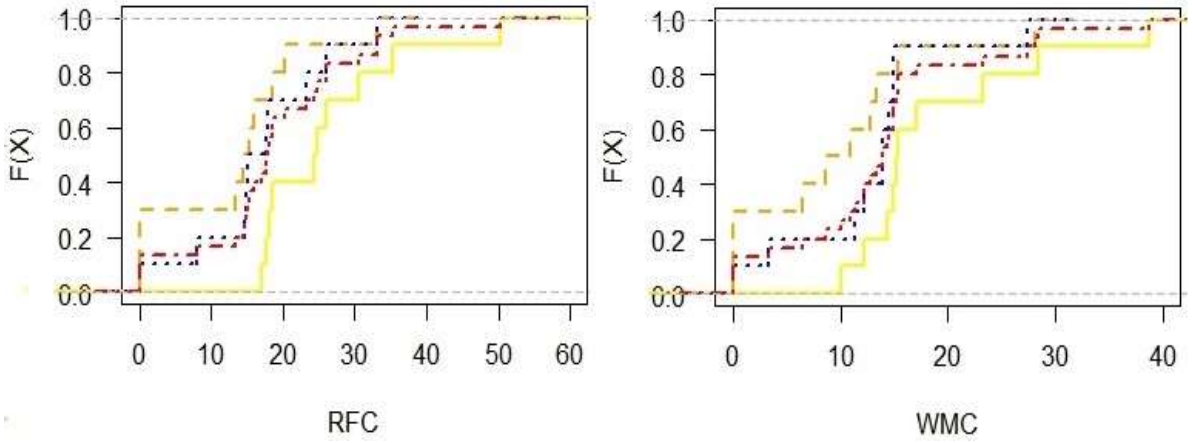
(b) Métrica MHF



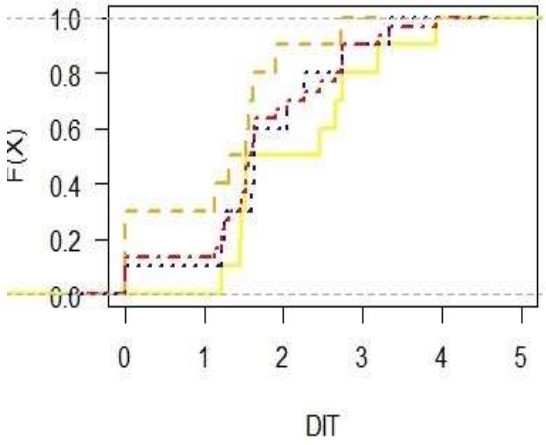
(c) Métrica MIF



(d) Métrica PF



(e) Métrica RFC (f) Métrica WMC



(g) Métrica DIT

Figura 1. Gráficos de Distribuição Acumulada das Métricas *Estruturais*

A Figura 1(g) apresenta as métricas *Depth of Inheritance Tree* (DIT). Para esta métrica a classe privacidade apresenta os melhores resultados, onde os resultados estão concentrados nos menores valores coletados. A classe tradicional possui valores concentrados nos maiores valores desta métrica. Isto indica que o código dos navegadores da classe tradicional é mais complexo que as demais classes.

5.2.2. Distribuição Acumulada da Métrica de Coesão

Na Figura 2 é apresentada a métrica de coesão *Lack of Cohesion in Methods* (LCOM). Pode-se observar que a classe dos navegadores tradicionais quando atinge a distribuição de noventa por cento, os valores ficam distribuídos de 2 a 4, indicando que os valores estão concentrados nos valores mais altos. Considerando a linha da classe de navegadores focados em segurança, percebe-se que os valores estão concentrados nos valores inferiores. Com isto, pode-se considerar que a classe tradicional apresenta classes mais complexas. A classe de segurança apresenta valores mais baixos e com isso menos complexas.

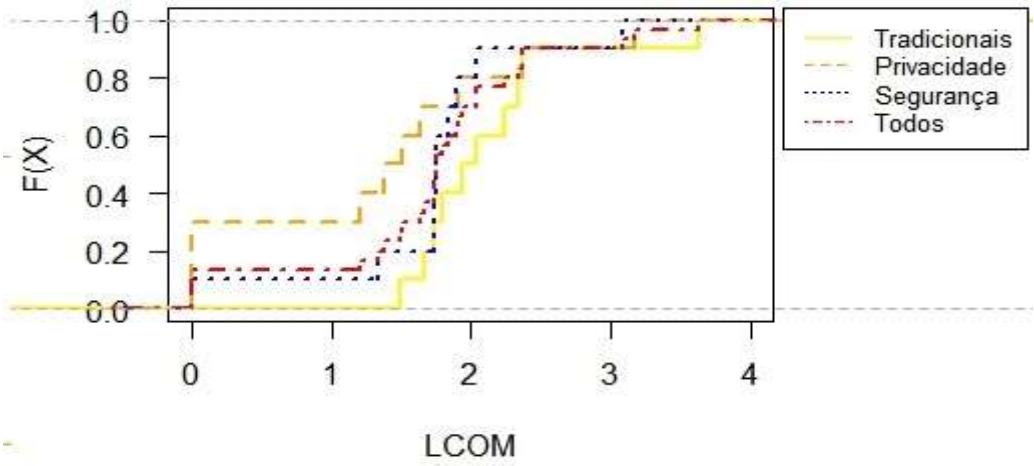


Figura 2. Gráficos de Distribuição Acumulada das Métricas de *Lack of Cohesion in Methods* (LCOM)

5.2.3. Distribuição Acumulada das Métricas de Acoplamento

A Figura 3 apresenta a distribuição da métrica de acoplamento. Após o processo de seleção das métricas através da aplicação da correlação, foi selecionado apenas uma métrica, sendo a métrica CBO. Conforme apresentado nesta figura, a classe de navegadores tradicionais é a que apresenta os maiores resultados, onde 90% seus valores concentram-se entre 4 e 14. Ou seja, os seus valores são os que estão concentrados nos maiores valores. Em contrapartida a classe dos navegadores focados em privacidade apresentaram os menores valores, estando concentrados 90% dos valores entre 0 e 8. Com isto, pode-se dizer que os navegadores focados em privacidade apresentaram os melhores resultados.

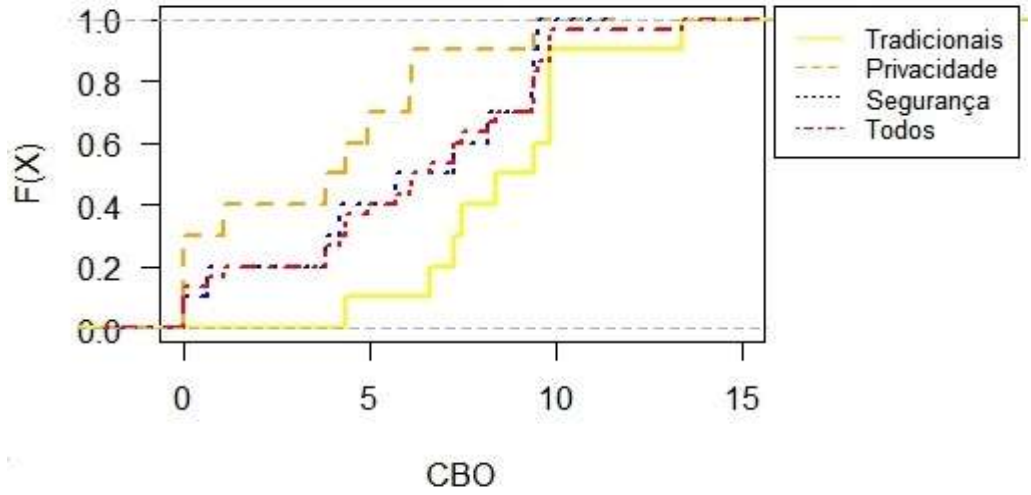


Figura 3: Gráficos de Distribuição Acumulada das Métricas de *Coupling Between Objects* (CBO)

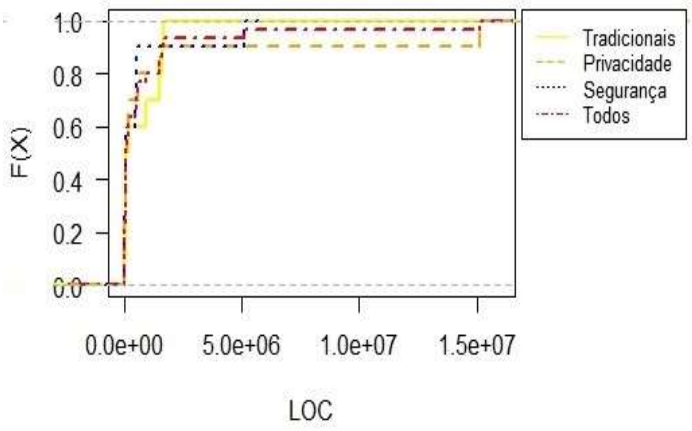
5.2.4. Distribuição Acumulada das Métricas de Tamanho

A Figura 4 apresenta os resultados das métricas de tamanho. O objetivo de apresentar estas métricas, deve-se ao fato de analisar a qualidade da distribuição dos códigos nos pacotes e quais classes possuem maior número de linhas de códigos em linguagens *backend* e *frontend*. Para isto, possuem métricas de tamanho total de linhas de código, média de linha de código, linhas de código de *frontend* e *backend*.

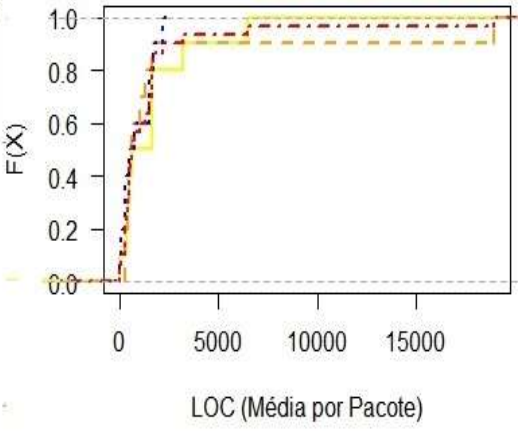
A Figura 4(a) apresenta a distribuição acumulada do total de linha de código. A figura mostra que não há diferenças significantes entre as classes. A Figura 4(b) apresenta a média de LOC por pacote, onde esta métrica visa verificar a distribuição das linhas de código pelos pacotes. Nesta figura pode-se observar que as classes possuem resultados similares, mas pode-se verificar que na classe de segurança cerca de 90% dos valores estão entre 2.000 e 6.000 linhas de código por pacote. A classe de navegadores tradicionais apresenta 80% dos valores entre 2.000 e 4.000 linhas de código por pacote. Com isto, observa-se que a classe de navegadores focados em segurança são melhores distribuídos nos pacotes.

A linguagem *Groovy* é orientada a objetos e foi desenvolvida para a plataforma Java como alternativa à linguagem de programação *Java*. *Groovy* possui características de *Python*, *Ruby* e *Smalltalk*. A Figura 4(c) apresenta a métrica de *LOC* da linguagem *Groovy*, onde a classe de navegador focado em segurança apresenta cerca de 60% dos resultados conentrados entre 0 e 1.000 linhas de código. A classe de privacidade apresenta os valores maiores concentrados na maior porcentagem da distribuição e consequentemente os maiores valores desta métrica. A Figura 4(e) apresenta os resultados da métrica de tamanho da linguagem de programação *Kotlin*. *Kotlin* é uma Linguagem de programação multiplataforma que compila para a Máquina virtual Java e que também pode ser traduzida para *JavaScript* e compilada para código nativo. Nesta figura é possível verificar que os navegadores da classe tradicional possuem 90% do código estão em até 15.000 linhas de códigos. A classe tradicional é a que possui maior número de linhas de códigos escritas em *Kotlin*. A classe de navegadores focados em segurança possui os menores valores de linha de código escrita em *Kotlin*.

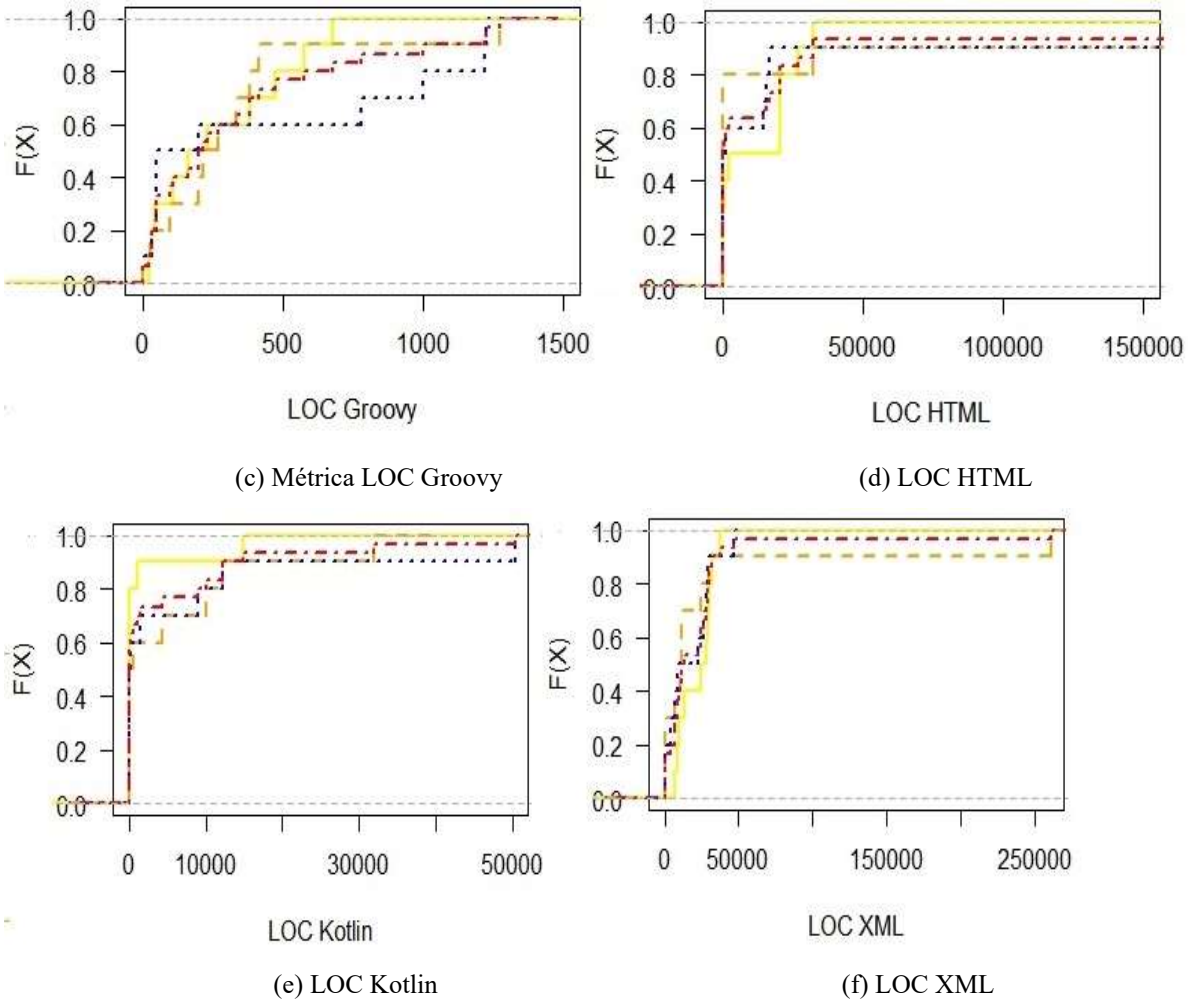
Na Figura 4(d) é apresentado os resultados da métrica de *LOC* da linguagem *HTML*, onde os resultados das classes possuem comportamentos semelhantes. Os resultados da métrica *LOC* referentes a linguagem *XML* são apresentados na Figura 4(f), onde os resultados das classes possuem valores similares. Os resultados das linguagens de *frontend* possuem valores similares em ambas as linguagens analisadas.



(a) Métrica LOC



(b) Métrica LOC Média por Pacote



(c) Métrica LOC Groovy (d) LOC HTML

(e) LOC Kotlin (f) LOC XML

Figura 4. Gráficos de Distribuição Acumulada das Métricas de Tamanho.

5.3. Discussão

Esta seção apresenta uma análise dos resultados de qualidade de código-fonte das métricas coletadas. Foram apresentadas 15 métricas, onde em 60% dos resultados das classes foram semelhantes. Considerando as métricas onde os resultados das classes não foram semelhantes, a classe de navegadores focados em privacidade apresentou os melhores resultados em 5 das 6 métricas analisadas. Destas 6 métricas a classe de navegadores tradicionais apresentou os piores resultados em todas.

Conforme mencionado a classe que apresentou os melhores resultados foi a classe de navegadores focados em privacidade. A primeira métrica na qual os navegadores focados em privacidade apresentam o melhor resultado foi a RFC. Essa métrica mede a complexidade de uma classe, que aumenta proporcionalmente ao valor do RFC. A próxima métrica, onde a classe de privacidade obteve os melhores valores foi na WMC. Nessa métrica obtém-se o valor para a complexidade dos métodos de um sistema, assim como uma medida de tamanho da classe. A métrica DIT foi outra na qual esta classe de navegadores apresentou o melhor resultado. Essa métrica mede o comprimento da árvore de herança da raiz até o nó folha de maior tamanho. Pode-se considerar que, quanto maior o valor de DIT, maior será a dificuldade de prever o comportamento das classes herdeiras e maior será complexidade do sistema. Por último a métrica CBO avalia o acoplamento de uma classe, onde uma classe A está acoplada a uma classe B quando a classe A utiliza métodos ou variáveis da classe B.

Já os navegadores focados em segurança apresentaram resultados regulares, apresentando o melhor resultado na métrica de coesão LCOM. Este resultado indica que os navegadores da classe de segurança possuem métodos fortemente coesos. Em contrapartida a classe de navegadores tradicionais apresentam resultados que devem ser melhorados. Em alguns casos como na métrica DIT, apesar da classe tradicional apresentar o pior resultado, os resultados não são classificados como péssimos resultados, tendo em vista os valores de referência apresentados na Tabela 1. Para realizar as melhorias nos navegadores tradicionais seria necessário à equipe de desenvolvimento verificar os valores de referências, os quais estão presentes na literatura.

Nesta pesquisa, foram levantados também alguns questionamentos, utilizando a abordagem GQM para definição das métricas que seriam usadas para medir a qualidade das classes dos navegadores que foram utilizados nesta pesquisa. A seguir é apresentada a Tabela 4, onde são listadas as respostas levantadas no GQM, visando apresentar os questionamentos referentes a qualidade do software.

Tabela 4. Questões às perguntas levantadas no GQM

Questões	Resposta	Explicação
Qual a classe de navegador apresenta uma melhor qualidade das classes e funções?	A classe de privacidade apresentou o melhor resultado, considerando as métricas de coesão e estruturais.	As métricas de coesão e estruturais, visam medir questões de qualidade e complexidade das classes e seus atributos. Cada uma das classes de navegadores apresentou valores melhores em 4 das 9 métricas avaliadas.
Qual classe de navegador apresenta uma melhor distribuição do código por pacotes?	A classe com melhor distribuição de linhas de código por pacote foi a de privacidade.	Esta classe apresentou os maiores valores, onde cerca de 90% está entre 2.000 e 6.000.
Qual classe de navegador apresenta uma melhor coesão?	A classe de segurança foi a classe que apresentou melhores resultados para a coesão.	A métrica responsável por medir a coesão foi a métrica LCOM, onde está métrica é o número de métodos que acessam um ou mais dos mesmos atributos. Altos valores para esta métrica, indica que os métodos podem ser acoplados uns aos outros via atributos. Isso aumenta a complexidade do projeto de classe. Com isto, a classe de navegadores focados em segurança apresentou os melhores resultados.
Qual classe de navegadores apresenta um melhor acoplamento?	A classe que obteve o melhor resultado foi a de navegadores focados em privacidade.	Para avaliar esta questão foi analisado a métrica CBO.
Levando em consideração a preocupação com a segurança, os navegadores focados em segurança apresentam melhor índices de qualidade?	Não	A classe de segurança apresentou bons resultados, mas pode-se dizer que a classe de navegadores focado em privacidade apresentaram resultados mais constantes e com isso, uma melhor qualidade. O principal ponto a ser melhorado pela classe de segurança, refere-se ao polimorfismo dos códigos da classe.

Tendo em vista a análise apresentada, pode-se dizer que as classes de privacidade e segurança foram as que apresentaram melhores resultados respectivamente. A classe de navegadores tradicionais apresentou resultados a serem melhorados na maior parte das métricas coletadas.

6. Conclusões e Trabalhos Futuros

Este estudo tratou da questão de pesquisa: Os navegadores focados em segurança apresentam os melhores índices de qualidade de código-fonte? Para se obter as respostas deste

questionamento, inicialmente, foi realizado o levantamento das métricas e navegadores de cada classe a serem utilizados neste estudo. Após isto foi realizado a obtenção dos códigos fonte dos navegadores, preparação do ambiente e coleta dos resultados das métricas. Por último, foi realizado a coleta das correlações das métricas, visando verificar quais métricas seriam utilizadas no estudo e geração dos gráficos de distribuição acumulada, onde pode-se verificar a distribuição dos resultados e obter-se uma conclusão sobre a qualidade de código-fonte dos navegadores.

Os resultados obtidos mostram que as classes de privacidade e segurança foram as que apresentaram melhores resultados respectivamente. A classe de navegadores tradicionais apresentou resultados a serem melhorados na maior parte das métricas coletadas. Com isso, pode-se verificar que a classe de navegadores focados em segurança não são os que apresentam os melhores valores de qualidade de software, ficando atrás dos navegadores focados em privacidade. Apesar da classe de segurança não ser o que apresenta os melhores resultados, foi possível observar que a mesma apresentou bons resultados.

Em relação aos trabalhos futuros, um dos possíveis estudos a serem feitos é realizar um estudo comparativo entre as classes de navegadores para dispositivos móveis e computadores, visando verificar se o comportamento é semelhante. Para auxiliar possíveis trabalhos futuros, os materiais utilizados para execução deste trabalho foram disponibilizados em repositório público na plataforma *GITHUB* < https://github.com/felipeaugustosm/TCC2>.

Referências

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.

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.

Filó, Tarcisio Guerra Savino. 2014. “Identificação de valores Referência para Métricas de *Software*" 223f. Dissertação de Mestrado em Ciência da Computação do Instituto de Ciências Exatas da Universidade Federal de Minas Gerais, Belo Horizonte, 2014

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.

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

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.

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, 2011.0