Avaliação das ferramentas de análise estática de códigos PHP utilizando *templates* de *websites* eGov baseados em CMS

Diego Comis[1], **Samuel Forrati**[1], **Maicon Bernardino**[1], **Elder de Macedo Rodrigues**[1], **João Pablo da Silva**[1]

[1]Universidade Federal do Pampa (UNIPAMPA) - *Campus* Alegrete
Av. Tiarajú, 810, Ibirapuitã – Alegrete, RS – Brasil - Alegrete - RS

{diegocomis.aluno, samuelforrati.aluno}@unipampa.edu.br

{elderrodrigues, joaosilva}@unipampa.edu.br

{bernardino}@acm.org

Abstract. *Static code analysis is a software verification technique, where a tool identifies defects prior to the execution of code snippets. This work proposes to analyze and compare some of these tools for PHP. To achieve this goal, five tools were selected among the 70 found, which passed criteria such as: having open source, providing documentation and covering code metrics. The tools were classified according to efficiency and effectiveness and their evaluation took place through tests with three projects in PHP for websites of the Federal Government of Brazil (eGOV), based on the CMS's Drupal, Joomla and WordPress.*

Resumo. *A análise estática de código é uma técnica da verificação de software, onde uma ferramenta identifica defeitos anteriores à execução de trechos de código. Este trabalho propõe analisar e comparar algumas dessas ferramentas para o PHP. Para atingir esse objetivo, foram selecionadas cinco ferramentas entre as 70 encontradas, que passaram por critérios como: ter código aberto, disponibilizar documentação e abranger métricas de código. As ferramentas foram classificadas conforme eficiência e eficácia e sua avaliação ocorreu por meio de testes com três projetos em PHP para sites do Governo Federal do Brasil (eGOV), baseados nos CMS's Drupal, Joomla e WordPress.*

1. Introdução

O mercado de desenvolvimento de *software* tem evoluído aceleradamente e para o sentido de produtos cada vez mais complexos, no menor tempo possível e com custos menores. Nesse sentido, a engenharia de software busca ser o meio para produzir softwares economicamente viáveis e confiáveis [Boehm 1976]. Cada vez mais, utilizamos e somos dependentes dos softwares, aos quais utilizamos direta ou indiretamente todos os dias [Rios 2013], seja para controle de tráfego aéreo, pagamentos de contas, sistemas de defesa, entre outros [Boehm 1976]. Nesse sentido, os *softwares* devem passar por rigorosos critérios de testes com a finalidade de mitigar e evitar erros, pois simples falhas podem afetar a confiabilidade, funcionalidade, usabilidade e segurança, causando, *e.g.* aumento do custo de desenvolvimento, prejuízos econômicos, roubo de dados, dentre outros transtornos [Rios 2013].

Os testes manuais existentes para assegurar a correção de erros em *software* são muito exaustivos e dispendiosos, e como afirma Dijkstra [Dijkstra 1979], testes de *software* não provam ausência de erros, ou seja, a impossibilidade de poder ocorrer falhas em circunstâncias ainda não consideradas. Entretanto, o comportamento indefinido existe em diversas linguagens, no nosso caso, na linguagem PHP, são encontrados erros como fuga de memória, acesso a variáveis não inicializadas, entre outras, e quanto mais cedo um erro for identificado, menores serão as perdas, e assim consolidando cada vez mais o projeto ou *software* em desenvolvimento.

Dessa forma, o processo de validação de *software* é uma das etapas importantes na qual encontramos os princípios de Verificação e Validação (V&V). Conforme [Boehm 1976], V&V são muito semelhantes, porém com uma sucinta diferença entre elas, *i.e.* a verificação tende a mostrar o que o projeto atende quanto a especificação, enquanto na validação nos mostra se o produto de software realiza exatamente aquilo que o usuário final espera que ele faça. Assim, o principal objetivo do processo de V&V é estabelecer confiança de que o projeto de *software* esteja totalmente adequado ao seu real propósito.

Atualmente, três técnicas são essencialmente usadas para o processo de V&V, a revisão manual de código, a análise semiautomática, e a análise automática (composta pelas análises dinâmica ou estática). A Análise Estática de Código (AEC), a qual é apresentada nesse trabalho, permite a análise e identificação de erros sem a necessidade da execução do *software*. Essa técnica se baseia em examinar o código-fonte ou partes dele, podendo ser utilizada durante toda fase de desenvolvimento. O uso da análise estática nas fases iniciais de desenvolvimento do projeto ajuda na identificação de erros, reduzindo consideravelmente os custos finais do projeto [Corporation 2022] [Muske et al. 2018].

Motivado pelo esforço necessário, é interessante que a análise do processo de V&V seja o mais automatizada possível, isto é, sem a necessidade de muitas intervenções por parte dos programadores. Assim, a motivação para a realização desse trabalho, foi a análise e comparação das principais ferramentas encontradas para análise estática de código para a linguagem de programação PHP, realizando testes com modelos de projeto padrão (*templates*) para *sites* do Governo Federal do Brasil (eGOV), disponibilizados gratuitamente no *GitHub*, desenvolvidos para os CMS (CMS - Content Management System) ou Sistemas de Gerenciamento de Conteúdo denominados *Drupal*, *Joomla* e *WordPress*, visando encontrar defeitos no código.

Para a escolha dos projetos a serem testados (SUT - *System Under Test*), a motivação principal é testar grandes projetos utilizados por *sites* governamentais, disponibilizados nos principais CMS do mercado como *Drupal*, *Joomla* e *WordPress* [W3Techs 2022]. O objetivo deste artigo é realizar um estudo comparativo do mesmo projeto de *site* governamental (eGov), em diferentes plataformas CMS por meio de diferentes ferramentas de análise estática de código PHP.

O artigo está organizado conforme segue. Seção 2 descreve os principais conceitos abordados no escopo deste trabalho. Seção 3 apresenta o planejamento da avaliação realizada, descrevendo as ferramentas e os CMS selecionados. Seção 4 analisa e discute os resultados da avaliação das ferramentas com cada CMS. Seção 5 argumenta as ameaças a validade do estudo. Por fim, a Seção 6 conclui o estudo e indica trabalhos futuros.

2. Referencial Teórico

Esta seção apresenta os principais conceitos que envolvem os temas abordados, bem como os trabalhos relacionados ao presente trabalho proposto.

2.1. Análise Estática de Código (AEC)

Segundo [Louridas 2006], análise estática de código é a revisão do código-fonte por meio da busca por padrões conhecidos de erros, ou erros previsíveis em uma aplicação sem que seja necessário executá-la.

Pode-se citar alguns benefícios referente a utilização de ferramentas de AEC, tais como: encontrar erros e códigos de risco, fornecer um *feedback* aos programadores para ajudá-los a reconhecer onde eles foram precisos e onde eles foram desatentos, fornecer a um líder de projeto uma oportunidade para estudar o código, o projeto e a equipe sob uma perspectiva diferente, e retirar certas classes de defeitos, o que possibilita que a equipe se concentre mais nas deficiências do projeto [Terra and Bigonha 2008]. A verificação e validação utilizando essas ferramentas é feita tomando-se por base um conjunto de regras pré-estabelecidas, e podem ser agrupadas em três aspectos principais: regras de estilos de programação, boas práticas de desenvolvimento e *bugs* na execução do *software* [DevMedia 2022] [Hovemeyer and Pugh 2004].

A **Verificação por Estilo** varre o código na procura de elementos como identação, espaços e *tabs*, formato e/ou presença de comentários, quantidade de parâmetros, entre outros. A **Verificação por Boas Práticas** visa realizar uma varredura onde se aplicam uma gama de regras com o objetivo de verificar se as práticas corretas estão sendo usadas, evitar a duplicação de código, tamanho de classes, métodos e parâmetros, a criação desnecessária de variáveis locais, etc. Por fim, a **Verificação por *Bugs*** trata de varrer o sistema em busca de encontrar defeitos, tomando por base as tendências comuns dos desenvolvedores em atrair defeitos, muitos desses defeitos não visíveis aos compiladores [Louridas 2006].

2.2. Métricas de Análise Estática de Código

As ferramentas de análise estática conseguem fazer um levantamento e obter resultados sobre o ponto de vista de diferentes métricas, medindo dessa forma a qualidade do projeto de *software*, veja as principais [Metrics 2022]:

Medidas de complexidade Halstead: por meio dos operadores distintos e o número total de operadores, podemos calcular o vocabulário do programa, a duração, volume, dificuldade, esforço, número de *bugs* entregues e o tempo necessário para programar;

Número de Complexidade ciclomática e contagem de métodos ponderados: A complexidade ciclomática é uma medida da complexidade da estrutura de controle de uma função ou procedimento e pode ser calculada de duas formas, número de cada ponto de decisão (CCN); Os algoritmos possíveis são: Complexidade ciclomática, linhas de código e a contagem de método não ponderado (WMC);

Manutenibilidade: Índice de manutenção é uma métrica de *software* que mede a manutenção (fácil de suportar e alterar) o código-fonte. O índice de manutenção é

calculado como uma fórmula fatorada que consiste em linhas de código, complexidade ciclomática e volume de *Halstead*;

Acoplamento: Conseguimos medir o acoplamento aferente, na qual o número de classes em outros pacotes que dependem de classes dentro do pacote é um indicador da responsabilidade do pacote, e o acoplamento eferente, no qual o número de classes em outros pacotes das quais as classes de um pacote dependem é um indicador da dependência do pacote de externalidades;

Ranking de página: O algoritmo criado por Larry Page e Sergey Brin é usado para classificar páginas da *web* de acordo com sua importância, PageRank é aplicado ao caso de relacionamentos entre pacotes e classes; **Comprimento:** Pode ser medido por meio da contagem de linhas (*Lines of Code* - LOC), contagem de linhas sem comentários (CLOC) e contagem de linhas sem linhas vazias (LLOC);

Falta de coesão de métodos: As métricas de coesão medem quão bem os métodos de uma classe estão relacionados entre si. Uma classe coesa executa uma função enquanto uma classe não coesa executa duas ou mais funções não relacionadas. Uma classe não coesa pode precisar ser reestruturada em duas ou mais classes menores.

2.3. Sistema de Gerenciamento de Conteúdos

Sistema de Gerenciamento de Conteúdos (CMS - *Content Management System*) é um *software* que facilita a criação, controle e edição de conteúdo HTML (um *site*) de forma simplificada e não requer muitos conhecimentos técnicos em programação para postar um conteúdo [Mafereka and Winberg 2017]. Assim, o usuário comum possui autonomia de gerir os conteúdos do *site* e os programadores não precisam lidar com muitas linhas de código para colocá-lo no ar, podendo dedicar esse tempo a estilização do *site*.

2.4. Trabalhos Relacionados

Com a finalidade de obter uma melhor visão e entendimento sobre o tema deste trabalho, nesta seção apresentamos os principais trabalhos relacionados a nossa abordagem.

[Souza et al. 2021] propõem analisar ferramentas de inspeção de código para linguagem de programação PHP disponíveis gratuitamente no *GitHub Marketplace*. Para atingir este objetivo, o sistema GLPI foi escolhido para ser inspecionado, além disso, 4 ferramentas de inspeção de código foram selecionadas, de 28 disponíveis. Para classificar os resultados obtidos, a Enumeração de Fraqueza Comum (CWE - *Common Weakness Enumeration*) foi usada, a CWE é uma lista de fraquezas de *software* e hardware desenvolvida por várias empresas renomadas, como Microsoft, Apple e IBM. Como um resultado do trabalho de inspeção, foram encontradas mais de dez mil falhas divididas em 34 CWEs diferentes e a partir desses se analisou o *feedback* individual de cada ferramenta, pois cada uma delas tinha vantagens e desvantagens únicas.

[Fatima et al. 2018] propõem uma análise estática de código eficiente para verificação do esquema da codificação do *software*, cuja característica significativa é que qualquer tipo de *bug* ou vulnerabilidade no código seja detectada sem a necessidade de execução do código. A principal preocupação dos autores é em detectar a construção do código complexo e os defeitos potenciais em um sistema, utilizando dezesseis diferentes ferramentas para análise de código C e C++ seguindo um conjunto de regras e padrões

definidos. O artigo inclui revisão sobre as ferramentas, comportamento em relação aos problemas injetados, problemas de entrada, funções e *loops* de problemas, problemas de variáveis e ponteiros e alguns problemas de códigos básicos. As ferramentas foram comparadas de acordo com 28 parâmetros e o resultado foi obtido por meio de pontuação.

O trabalho de [Lima et al. 2021] tem como principal objetivo a comparação das ferramentas de análise estática *SonarQube* e *PMD*, com a finalidade de analisar a eficácia dos mesmos quanto à identificação de defeitos em código de *software*. Para isso, foram criados (códigos) mutantes a partir de projetos *open-source*.

Após a leitura dos trabalhos, conseguimos identificar algumas diferenças em relação ao nosso estudo proposto, e apesar de já existirem alguns trabalhos de análise estática em diversas linguagens, não encontramos com muita facilidade trabalhos científicos que façam uma comparação de ferramentas de análise estática na linguagem PHP. Dessa forma, como já mencionado, entendemos que o PHP por ser uma linguagem muito utilizada para *web*, tanto na indústria como na academia, frisamos a importância em identificar a qualidade dessas ferramentas no contexto da utilização dessa tecnologia.

3. Escolha das Ferramentas para a Análise Estática

Nesta seção é explanado sobre a escolha dos CMS para este estudo, além dos templates de *sites* governamentais aplicados aos CMS selecionados e, por fim, serão abordados os critérios para escolha das ferramentas de análise estática de código PHP.

3.1. Escolha dos CMS

A escolha dos CMS *Drupal*, *Joomla* e *WordPress* se deu seguindo uma instrução normativa, publicada pela Secretaria de Comunicação Social da Presidência República (SECOM) em janeiro de 2014, nela continham alguns direcionamentos de padronização dos *sites* das instituições/órgãos do Governo Federal. O documento sugeria a adoção ou utilização de gerenciadores de conteúdo de código aberto que já continham o *template* oficial do governo (GovBr) criado por sua comunidade. Segundo [SECOM 2014] diversos ministérios vinculados ao planalto utilizavam o CMS *Joomla*, dentre eles o Ministério da Educação (MEC), Marinha e Exército Brasileiro, entre outros, e em *WordPress* o Ministério da Cultura e o da Previdência Social. Esses CMS mencionados são utilizados em larga escala pelas 42 Instituições de Ensino Superior (IES) ligadas ao MEC, a rede federal de ensino, as quais incluem também os Institutos Federais. Complementarmente, *Drupal*, *Joomla* e *WordPress* são os CMS *open source* mais utilizados pelo mercado, como pode ser visto na Tabela 1 [W3Techs 2022].

3.2. Ferramentas para o estudo de Análise Estática

Para a escolha das Ferramentas de Análise Estática (*Static Analysis Tool*), os autores investigaram ferramentas em trabalhos que fazem comparações de ferramentas de AEC. Adicionalmente, foi utilizado um repositório *GitHub* com uma lista de 71 ferramentas[1] de garantia de qualidade para PHP [Zalas 2022]. Dessa forma, para escolher as ferramentas de AEC participantes do estudo, foram definidos os seguintes critérios de seleção: **Critério 1:** permitir a realização da análise estática de código em linguagem de programação PHP; **Critério 2:** ser *open source*, ou seja, ter licença gratuita e documentação disponível

[1]Repositório *Quality Assurance Tools for PHP*: https://jakzal.github.io/toolbox/

Tabela 1. Top 10 CMS líderes do mercado global

#	CMS	Ano	Tipo	Quota de Mercado	Quota de Uso	Site - URL
1	*WordPress*	2003	O	64.10%	42.90%	`WordPress.com`
2	Shopify	2006	S	6.40%	4.30%	`shopify.com.br`
3	Wix	2006	S	3.40%	2.30%	`wix.com`
4	Squarespace	2004	S	3.00%	2.00%	`squarespace.com`
5	*Joomla*	2005	O	2.50%	1.60%	`launch.joomla.org`
6	*Drupal*	2001	O	1.90%	1.20%	`drupal.org`
7	Blogger	2008	O	1.30%	0.90%	`blogger.com`
8	Bitrix	1999	F	1.20%	0.80%	`bitrix24.com.br`
9	Magento	2012	S	0.90%	0.60%	`magento.com`
10	Webflow	2013	S	0.90%	0.60%	`webflow.com`

Legenda: Ano - Ano de Lançamento | O - *Open Source* | S - SaaS | F - *Free*

em algum repositório aberto de código-fonte, *e.g. GitHub*; **Critério 3:** possuir métricas de AEC relacionadas ao código-fonte, arquitetura, complexidade e estilo. Dentre as ferramentas identificadas, foram selecionadas as que atendiam os critérios estabelecidos. As ferramentas selecionadas para a análise estática de código PHP foram: ***PHP Depend, Phan, PHP Insights, PHP Metrics e PHP Mess Detector***.

4. Análise e Discussão dos Resultados

Nesta seção é abordado o formato da análise das ferramentas de análise estática de código PHP e os resultados obtidos nessas avaliações.

4.1. Metodologia da Análise das Ferramentas

Para se avaliar os resultados das ferramentas de análise estática, foram escolhidos 3 projetos PHP disponíveis para o Portal Padrão em CMS (eGOV), conforme consta na Seção 3.1 deste artigo. A versão do *Drupal* utilizada é a versão 7.x e está disponível no repositório Git da própria ferramenta, para o *Joomla* a versão utilizada é a "3.9.6", disponível no repositório do *GitHub*, contendo 82% de seu código-fonte em PHP, já a versão para o projeto *WordPress* utilizado foi a versão "4.9", também disponibilizado no repositório aberto do *GitHub* e contendo um total de 66,7% de seu código-fonte em PHP. Cada projeto possui ainda arquivos de imagens, folhas de estilo CSS e códigos-fontes em *JavaScript*.

Sendo assim, optamos por definir os seguintes passos para a obtenção dos resultados. Primeiramente, foi analisado um relatório geral extraído de cada ferramenta contendo as principais métricas da análise estática: complexidade, manutenibilidade, esforço, dificuldade, número de erros entregues, boas práticas, entre outros. Os relatórios foram analisados em conjunto por dois analistas, verificando e classificando de acordo com a nomenclatura ou descrição das métricas analisadas pelas ferramentas. Após a verificação dos resultados de cada ferramenta, foram tabulados para efeito de comparação e disponibilizados no repositório externo da Zenodo[2], criado exclusivamente para este trabalho. Devido a análise estática ser bem extensa, todos os resultados foram inseridos no repositório, contendo um arquivo PDF explicando a análise dos resultados através de gráficos, tabelas e imagens.

[2]Repositório *Zenodo*: https://doi.org/10.5281/zenodo.7073490

4.2. Comparação das Ferramentas de AEC

A primeira etapa foi realizar o levantamento de quais e quantas métricas cada ferramenta possui e é capaz de fazer a Análise Estática de Código (AEC), e enumeramos as ferramentas em ordem de melhor eficiência, quantidades de métricas, e que possuem serviços e componentes para a verificação e avaliação dos resultados por diferentes pontos de vista:

1. ***PHP Metrics*:** mais de 30 métricas em oito (8) grupos de análise, e possui funcionalidades que geram um painel *dashboard* completo com gráficos, comparações de acordo com os tipos de erros, podendo ser possível visualizar os erros e obter as indicações de como corrigir; 2. ***PHP Mess Detector*:** 17 métricas organizadas por nomes e relatórios extraídos em HTML, organizadas por classes e tipos de erros; 3. ***PHP Depend*:** 51 métricas próprias de código e funcionalidades para gerar gráficos e análises de código via JSON ou XML, entre outros; 4. ***Phan*:** 18 métricas próprias de código, porém possui poucas ferramentas, como relatórios para análise, sendo melhor executada para a correção em linha de código; 5. ***PHP Insights*:** quatro (4) grupos de análise de código, sendo realizada uma análise mais geral por arquitetura, estilo, código e complexidade, gerando um sumário com as análises médias e porcentagens de acordo com a pontuação ranqueada por meio da ferramenta. Exibe e executa os erros de cada arquivo em console da IDE de desenvolvimento.

Dentre as ferramentas analisadas, podemos verificar ainda as funcionalidades existentes ou disponíveis para a análise estática de código PHP em cada uma delas, como apresentado na Tabela 2.

Tabela 2. Comparação entre as funcionalidades das ferramentas analisadas

	PHP Metrics	PHPMD	PDepend	*Phan*	PHPInsights
Relatórios HTML	✓	✓	✗	✗	✗
Relatórios JSON	✓	✗	✗	✓	✓
Relatórios XML	✓	✓	✓	✓	✓
Dashboard	✓	✗	✗	✗	✗
Gráficos	✓	✗	✓	✗	✗
Métricas de Classes	✓	✓	✓	✓	✓
Métricas de Métodos	✓	✓	✓	✓	✓
Métricas de Complexidade de Código	✓	✓	✓	✓	✓
Integrações com IDEs	✓	✓	✓	✓	✓
Corrigir Erros Automaticamente	✓	✓	✓	✓	✓
Disponível para mais de uma versão PHP	✓	✓	✓	✓	✓

Na Tabela 3, é possível verificar a comparação entre métricas gerais de análise entre as cinco (5) ferramentas de análise estática de código PHP utilizadas neste estudo. São analisadas as Métricas de Arquitetura (classes, *interfaces*, *traits*, *globally*), Código (classes, *comments*, *globally*), Complexidade e Estilo. Os resultados possíveis são ✓ (Sim) ou ✗ (Não). Na Tabela 4, é possível verificar a comparação entre número de erros encontrados por ferramenta de análise estática entre as cinco (5) ferramentas de análise estática de código PHP utilizadas neste estudo. Os resultados possíveis são valores numéricos.

Já na comparação da Tabela 5, analisamos as principais métricas abordadas anteriormente nesse artigo em cada ferramenta de análise estática de código PHP selecionada.

Tabela 3. Comparação entre métricas gerais de análise

Métricas Gerais de Análise	*Phan*	PHPInsights	PHPMD	PHPMetrics	PDepend
Architecture (e.g. classes, interfaces, traits, globally)	✓	✓	✓	✓	✓
Code (e.g. classes, comments, globally)	✓	✓	✓	✓	✓
Complexity	✗	✓	✓	✓	✓
Style	✗	✓	✗	✗	✗

Tabela 4. Comparação entre número de erros encontrados por ferramenta de análise estática de código PHP

Projeto Analisado	*Phan*	PHPInsights	PHPMD	PHPMetrics	PDepend
CMS *Drupal*	7284	662	10	110	12,06
CMS *Joomla*	2.280	14	479	13	18,8
CMS *WordPress*	57.917	49	2062	164	37,06

Então, podemos chegar a constatação que grande parte das ferramentas atendem ou executam a maioria dessas métricas, salvo algumas exceções, como é evidenciado pelos dados apresentados nessa tabela com esse estudo.

Tabela 5. Comparação entre as principais métricas das ferramenta selecionadas

Métricas	*Phan*	PHPInsights	PHPMD	PHPMetrics	PDepend
Medidas de complexidade *Halstead*	✓	✓	✓	✓	✓
Número de complexidade ciclomática e contagem de métodos ponderados	✓	✓	✓	✓	✓
Manutenibilidade	✗	✓	✓	✓	✓
Acoplamento	✓	✓	✓	✓	✓
Ranking de página	✗	✗	✓	✓	✓
Comprimento	✓	✓	✓	✓	✓
Falta de coesão de métodos	✓	✗	✓	✓	✓
Regras de código limpo	✓	✗	✓	✓	✓
Regras controversas	✓	✓	✓	✓	✓
Regras de projeto	✓	✓	✓	✓	✓
Regras de nomenclatura	✓	✗	✓	✓	✗
Regras de código não utilizadas	✓	✓	✓	✓	✓

5. Ameaças a Validade do Estudo

A validade desse estudo passa por algumas ponderações importantes, ou seja, as diferentes ferramentas seguem métricas próprias quanto a análise de código estático, mas isso não significa que não seja interessante utilizá-las, ou que possuem qualidade menor do que outras ferramentas. Cada ferramenta possui formas específicas de classificar determinados trechos de código, podendo nem sempre encontrar ou analisar os mesmos erros, categorizando-os de diversas maneiras, e para mitigar essa falta de padronização, procuramos utilizar as métricas comuns entre si para realizar a classificação e análise estática.

Devemos destacar ainda, que a seleção das ferramentas foi feita de forma manual e, tentando evitar o erro humano, conforme apresentados na Seção 3.2.

Primeiramente, fizemos uma pesquisa por trabalhos correlatos sobre o uso de ferramentas de AEC, chegando a uma lista de 71 ferramentas para garantia de qualidade PHP, organizadas em um projeto de [Zalas 2022], em que as reúne em um repositório público do *GitHub*. Dentre todas, procuramos selecionar aquelas que seguissem os critérios de seleção estabelecidas nesse estudo, conforme Seção 3.2. Devido a esses pontos abordados, ou por alguma classificação equivocada dos proprietários, pode-se ter deixado de incluir alguma ferramenta nesse estudo. Sobre os projetos analisados, utilizamos três projetos semelhantes referentes a base de código para padronização de *template* de *sites* do eGOV, para CMS *Drupal*, *Joomla* e *WordPress*, e a verificação de cada ferramenta foi executada manualmente, uma a uma, gerando assim diversos relatórios com resultados adicionados ao repositório *Zenodo*, podendo assim serem revalidados a qualquer tempo.

6. Considerações Finais

Nesse artigo, realizamos o estudo de cinco ferramentas de análise estática de código PHP, para realizar a verificação e comparação com base no código de três projetos existentes no *GitHub*, os quais são disponibilizados de forma gratuita com a finalidade de padronização dos *sites* governamentais brasileiros, sendo um em CMS *Drupal*, *Joomla* e outro em *WordPress*. Por meio das métricas de manutenibilidade, complexidade, comprimento de código, coesão de métodos, acoplamento, entre outros, conseguimos obter diversos resultados, bem como o vocabulário dos programas, duração, número de erros ou *bugs*, a dificuldade e o tempo necessário de execução para programas, dentre muitos outros.

Esta revisão será importante para pesquisadores, desenvolvedores e gestores da área, pois servirá de apoio na tomada de decisões sobre as diferentes ferramentas de análise estática de código PHP e seus diversos recursos. No entanto, podemos constatar que nenhuma ferramenta é completa para detectar todos os erros ou vulnerabilidades de código, algumas funcionam melhor na análise de pequenos trechos de código, como a ferramenta *Phan* e a *PHP Insights*. Por outro lado, as ferramentas que apresentaram melhores resultados foram *PHP Metrics*, *PHP Depend* e *PHP Mess Detector*, por possuir e se basearem em excelentes métricas, nos proporcionar diversas formas de análise e acesso aos seus relatórios. Nenhuma das ferramentas cobre todos os padrões, mas abrange a maioria deles que são necessários para a segurança crítica e sistemas de boa qualidade desenvolvidos. Ainda como resultados deste estudo, concluímos que dentre os projetos de CMS utilizados como objeto desse estudo, o que apresentou maiores erros foi o projeto baseado no CMS *WordPress*, resultados esses disponibilizados no repositório Zenodo[3].

Como direcionamento para trabalhos futuros, destacamos a necessidade de aprimorar o protocolo e método de pesquisa sistemático em engenharia de software experimental, além de criar projetos com erros que possam ser detectados pelas ferramentas e realizar novos testes comparativos. Por fim, agradecemos a FAPERGS pelo financiamento do projeto de número 22/2551-0000841-0.

Referências

Boehm, B. W. (1976). Software engineering. *IEEE Trans. Computers*, 25(12):1226–1241.

[3]Repositório *Zenodo*: https://doi.org/10.5281/zenodo.7073490

Corporation, I. (2022). Dynamic analysis vs. static analysis. Disponível em: https://software.intel.com/en-us/inspector-user-guide-linux-dynamic-analysis-vs-staticanalysis. Acesso em 29 Jul. 2022.

DevMedia (2022). Como adotar a análise estática de código. Disponível em: https://www.devmedia.com.br/como-adotar-a-analise-estatica-de-codigo/32727. Acesso em 13 Jul. 2022.

Dijkstra, E. (1979). Go to statement considered harmful. In *Classics in software engineering (incoll)*, pages 27–33. Yourdon Press, Upper Saddle River, NJ, USA.

Fatima, A., Bibi, S., and Hanif, R. (2018). Comparative study on static code analysis tools for c/c++. In *15th International Bhurban Conference on Applied Sciences and Technology (IBCAST'18)*, pages 465–469.

Hovemeyer, D. and Pugh, W. (2004). Finding bugs is easy. *SIGPLAN Not.*, 39(12):92–106.

Lima, Y., Fonseca, I., Chagas, J., Rodrigues, E., Bernardino, M., and Silva, J. (2021). Comparação de ferramentas de análise estática para detecção de defeitos de software usando mutantes. In *V ERES'21*, pages 159–168, Porto Alegre, RS, Brasil. SBC.

Louridas, P. (2006). Static code analysis. *IEEE Software*, 23(4):58–61.

Mafereka, M. and Winberg, S. (2017). Analysis and development of an online knowledge management support system for a community of practice: Comparing joomla, wordpress and drupal with regard to development of community of practice website. In *International Conference on Information System and Data Mining*, pages 6–10. ACM.

Metrics, P. (2022). Main metrics. Disponível em: https://phpmetrics.github.io/website/metrics/. Acesso em 20 Jul. 2022.

Muske, T., Talluri, R., and Serebrenik, A. (2018). Repositioning of static analysis alarms. In *27th ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 187–197. ACM.

Rios, Emerson; Moreira Filho, T. (2013). *Teste de software*. Alta Books, third edition.

SECOM (2014). Manual de diretrizes identidade padrão de comunicação digital do poder executivo federal, versão 3.4.

Souza, I., Campello, L., Rodrigues, E., Guedes, G., and Bernardino, M. (2021). *An Analysis of Automated Code Inspection Tools for PHP Available on Github Marketplace*, pages 10—-17. ACM, New York, USA.

Terra, R. and Bigonha, R. S. (2008). Ferramentas para análise estática de códigos java. *Trabalho de Conclusão de Curso de Ciência da Computação da UFMG*, page 63.

W3Techs (2022). Usage statistics of content management systems. Disponível em: https://w3techs.com/technologies/overview/content_management. Acesso em 16 Jul. 2022.

Zalas, J. (2022). Quality assurance tools for php | toolbox | phpqa. Disponível em: https://jakzal.github.io/toolbox. Acesso em 19 Jul. 2022.