



**INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA  
DO TRIÂNGULO MINEIRO - Campus Ituiutaba**

**CURSO SUPERIOR DE ANÁLISE E DESENVOLVIMENTO DE  
SISTEMAS**

**LEONARDO VILARINHO CORREIA DE SOUZA**

**Ferramenta para análise de métricas de qualidade de  
*software* em códigos *PHP***

**ITUIUTABA, MG**

**2017**

**LEONARDO VILARINHO CORREIA DE SOUZA**

**Ferramenta para análise de métricas de qualidade de  
software em códigos *PHP***

Trabalho de Conclusão de Curso  
apresentado ao Instituto Federal  
de Educação, Ciência e Tecnologia  
do Triângulo Mineiro, Campus  
Ituiutaba, como requisito parcial  
para conclusão do Curso de  
Análise e Desenvolvimento de  
Sistemas.

Orientador: Prof. \_\_\_\_\_

Ailton Luiz Dias Siqueira Junior

Membro: Prof. \_\_\_\_\_

Giselle Corrêa de Souza

Membro: Prof. \_\_\_\_\_

Reane Franco Goulart

**ITUIUTABA, MG**

**2017**

## SUMÁRIO

1.	Introdução	4
2.	Sistema desenvolvido	5
2.1	Dados de entrada	6
2.2	Dados calculados	6
2.3	Testes	7
3.	Resultados	8
4.	Conclusão	10
5.	Referências	11

# **Ferramenta para análise de métricas de qualidade de software em códigos *PHP***

**Leonardo Vilarinho Correia de Souza<sup>1</sup> ; Ailton Luiz Dias Siqueira Junior<sup>2</sup>**

<sup>1</sup>Estudante de Análise e Desenvolvimento de Sistemas, IFTM, Campus Ituiutaba, leonardo-i@outlook.com

<sup>2</sup>Professor do IFTM, Campus Ituiutaba, MG, ailton@iftm.edu.br

**Resumo:** O desenvolvimento de sistemas computacionais exige um constante planejamento e monitoração de resultados, pois sem eles o ciclo de vida do software tende a ser reduzido. Para que se obtenha sucesso nessas constantes, é fundamental que se tenha a garantia da qualidade do produto que está sendo desenvolvido. Nesse sentido, esse artigo demonstra uma ferramenta nomeada *Insphptor*, que realiza a análise do código fonte e do repositório *Git* de qualquer projeto feito com a linguagem *PHP* orientada à objetos. A ferramenta permite a visualização gráfica através de ferramentas estatísticas das principais métricas de código, permitindo identificar os seus principais problemas de qualidade. O sistema foi aplicado em um estudo de caso com códigos *PHP* de código aberto demonstrando a aplicabilidade do sistema.

**Palavras-chave:** Qualidade. Evolução. Métricas de software. *PHP*. Visualização da informação.

## **A tool for analyzing software quality metrics in *PHP* codes**

**Abstract:** The development of computational systems requires constant planning and monitoring of results, since without them the software life cycle tends to be reduced. In order to be successful in these constants, it is fundamental that you have the guarantee of the quality of the product being developed. In this sense, this article demonstrates a tool named *Insphptor*, which performs the analysis of the source code and the *Git* repository of any project made with the object-oriented *PHP* language. The tool allows graphical visualization through statistical tools of the main code metrics, allowing to identify its main quality problems. The system was applied in a case study with open source *PHP* codes demonstrating the applicability of the system.

**Keywords:** Quality. Evolution. Software Metrics. *PHP*. Display information.

## 1. INTRODUÇÃO

Ser expansível e modificável como um quebra-cabeça são requisitos essenciais para qualquer sistema orientado à objetos (ANICHE, 2015). Esses requisitos ditam um ciclo de vida longo para qualquer produto, porém sem a garantia de qualidade do que está sendo feito a vida útil do produto será reduzida (WALTERS; MCCALL, 1979).

Para atingir os requisitos citados, e consequentemente a garantia da qualidade, duas famosas abordagens foram estudadas e relevadas para esse artigo:

- ⌚ A abordagem de **Código Limpo**, que se refere a um código com três pilares básicos: a simplicidade, a flexibilidade e a expressividade (THOMAZ ALMEIDA; MACHINI DE MIRANDA, 2010).
- ⌚ A abordagem **SOLID** (*Single responsibility, Open-closed, Liskov substitution, Interface segregation and Dependency inversion*), que une cinco princípios universais para manter qualquer código simples, flexível e expressivo (ANICHE, 2015).

Com os padrões fornecidos por essas abordagens, foram descobertos aspectos comuns em projetos orientados à objetos, o que possibilitou analisar a evolução do código de um *software*. A partir disso surgiram as métricas de código, elas são métodos que analisam uma entrada e retornam índices que expressa a qualidade decorrente de um aspecto presente no dado analisado (SCHNEIDEWIND, 1997).

Dentre tais aspectos estão a complexidade, falta de coesão e acoplamento, que permitem a medição de diferentes tipos de métricas, como: de tamanho, complexidade, manutenibilidade, número de classes, número de métodos, acoplamento e outras (MEIRELLES, 2013).

No mercado atual, várias ferramentas têm o objetivo de calcular essas métricas, mas são complexas para configurar e mostram um resultado de difícil compreensão da equipe de desenvolvimento (MURPHY, 2005). Algumas tentam fugir desse problema, trazendo visualizações próximas do mundo real, como representação de cidades e redes sociais (BALOGH; BESZÉDES, 2013; SOKOL; ANICHE; GEROSA, 2013; WETTEL, 2009).

Todavia, no ambiente de desenvolvimento *PHP*, poucas ferramentas possuem o intuito de realizar essa medição. Com uma análise nesse mercado, foi tomado como objetivo a implementação de um sistema completo com uma simples configuração e uso, que dá a possibilidade de visualizar rapidamente e detalhadamente não só as métricas do código fonte, mas também da equipe de desenvolvimento.

## **2. SISTEMA DESENVOLVIDO**

Todo o trabalho foi desenvolvido usando a linguagem de programação *PHP*, fazendo uso de ferramentas modernas como o *Composer*, para o gerenciamento de dependências e *autoload*, e o *PHPUnit*, para construção e execução de testes automatizados e visualização da cobertura de testes.

A Figura 1 apresenta o fluxo de tarefas que o sistema desenvolvido segue para entregar o resultado das métricas para o desenvolvedor:



**Figura 1:** Fluxograma de trabalho do *Insphptor*.

Ao executar a análise de um código fonte, todos os arquivos são lidos e analisados tendo cada métrica calculada e armazenada em um objeto específico para ele. Após isso, as métricas da equipe de desenvolvimento são calculadas e só então um resumo do resultado é mostrado. Caso tenha sido habilitadas as funções para exportar o resultado e visualiza-lo no navegador, as mesmas serão feitas antes do término.

## 2.1 DADOS DE ENTRADA

Visando o uso simples do sistema, foi disponibilizado o comando *insphptor init* com intuito de criar automaticamente o arquivo *.yaml* necessário para a execução do sistema. Ao executar o comando uma série de perguntas serão feitas através do console, por exemplo: “Qual o nome do seu projeto?”.

No término da execução, será criado um arquivo nomeado *insphptor.yaml* no diretório atual, com a estrutura mostrada na Figura 2:

```

1  name: Insphptor Project
2  export: json
3  git: auto
4  rank: 5
5  hide:
6    - interface
7    - file
8  only:
9    - source
10 views:
11    overview: insphptor-overview

```

**Figura 2:** Arquivo de configuração do *Insphptor*

Dentre as opções configuradas são ressaltadas as opções: *git*, que indica se o projeto usa ou não o controlador de versões; *hide*, lista quais tipos de arquivos devem ser ocultos do resultado; *only*, determina quais diretórios do projeto serão analisados; *view*, mantém registrados os sistemas de visualização de resultados.

## 2.2 DADOS CALCULADOS

Como intuito principal do sistema dita, métricas são calculadas para identificar os aspectos negativos e positivos do *software* analisado. Neste trabalho foram selecionadas métricas que abrangem aspectos distintos, fazendo uso do controlador de versão *Git* para exploração de métricas sociais (SOKOL; ANICHE; GEROSA, 2013).

Na lista a seguir é detalhada cada métrica tomada para o sistema, o algoritmo de cada uma pode ser visto no Anexo 1:

- ⌚ *Complexidade ciclomática*: responsável por calcular a quantidade de caminhos que o *software* pode tomar em um determinado arquivo (MCCABE, 1976). Dada pela Equação 1:

(1)

onde: NR – Número de estruturas de repetição; NC – Número de estruturas condicionais; e NDB – Número de desvios abruptos, por exemplo: goto, break.



- ⌚ *Acoplamento eferente*: número de classes usadas dentro de uma classe, desde que foram declaradas fora do contexto/pacote (MARTIN; MICAHA, 2006). Dada no *PHP* pela Equação 2:

onde: NU – número de comandos *use*; NR – número de comandos *require* e *require\_once*; e NI – número de comandos *include* e *include\_once*.

























- ⌚ *Acoplamento aferente*: se define como o inverso da anterior, sendo o número de classes fora do contexto que dependem de determinada classe (MARTIN; MICAHA, 2006). No sistema atual, cada classe armazena suas dependências em uma lista, então o cálculo pode ser dado se a classe analisada está presente nas dependências de N classes, logo N é seu acoplamento aferente.
- ⌚ *Tamanho de código*: é a somatória dos números de instruções/tokens presentes na classe atual. No *PHP*, dado pela função *token\_get\_all*.
- ⌚ *Falta de coesão*: identifica a quantidade de partes em que uma classe pode se dividir (ANICHE, 2015). Ignorando *getters* e *setters*, calcula se os métodos de uma classe manipulam atributos distintos entre si, possibilitando a extração de métodos.
- ⌚ *Bugs*: a somatória de *commits* com palavras referentes a problemas no código na sua descrição, como: *bug*, *falha*, *fix*, *fail* e *broken*.
- ⌚ *Instabilidade*: a somatória de *commits* em que cada classe é alterada, concluindo que um arquivo que se modifica muito é considerado instável e de baixa qualidade.

- 🕒 **Inserções:** a somatória de linhas de código inseridas por um desenvolvedor em todo o histórico do projeto.
- 🕒 **Remoções:** a somatória de linhas de código removidas por um desenvolvedor em todo o histórico do projeto.

## 2.3 VALIDAÇÃO DO SISTEMA

Durante todo o desenvolvimento do sistema foram feitos testes manuais para visualizar e validar o resultado gerado pelo *Insphtor*. Esses testes consistiram em criar classes de exemplo e realizar a contagem manual das métricas desses arquivos, e então executar o sistema nesse código, comparando os resultados. Obteve-se uma notória conclusão da qualidade das heurísticas usadas no sistema, dado que os testes sempre estavam perto da real evolução do código analisado.

Em seguida, estes testes foram automatizados com cobertura do código a partir do *framework PHPUnit*, tendo como resultado um sistema com uma cobertura satisfatória mostrada na Figura 3:

	Code Coverage								
	Lines			Functions and Methods			Classes and Traits		
Total		95.25%	381 / 400		78.68%	48 / 61		75.86%	22 / 29
Analyzer		100.00%	43 / 43		100.00%	10 / 10		100.00%	2 / 2
Components		100.00%	99 / 99		100.00%	6 / 6		100.00%	6 / 6
Helpers		100.00%	11 / 11		100.00%	1 / 1		100.00%	1 / 1
Metrics		97.78%	132 / 135		76.92%	10 / 13		77.78%	7 / 9
Patterns		100.00%	14 / 14		100.00%	8 / 8		100.00%	3 / 3
Program		84.81%	67 / 79		42.86%	6 / 14		0.00%	0 / 4
Storage		78.95%	15 / 19		77.78%	7 / 9		75.00%	3 / 4
globals.php		n/a	0 / 0		n/a	0 / 0		n/a	0 / 0

**Figura 3:** Resultados da cobertura de testes

### 3. RESULTADOS

Foram coletados da plataforma *GitHub* diversos repositórios de largo uso para o desenvolvimento com *PHP*, dentre eles: *Laravel*, *Symfony*, *Yii*, *PHPUnit*, *Phing*, *Slim*, e outros. O *microframework Slim* foi selecionado para ser o estudo de caso apresentado neste artigo, pois o seu histórico de evolução e ganho de popularidade ficou bastante evidente com a análise feita pelo *Insphptor*.

Após executar o comando *insphptor run*, com o arquivo de configuração já criado (Anexo 2), é possível visualizar o resultado geral da análise feita em 9 de Dezembro de 2017 no *branch 4.x* do *Slim* (VILARINHO, 2017a) na Figura 4:

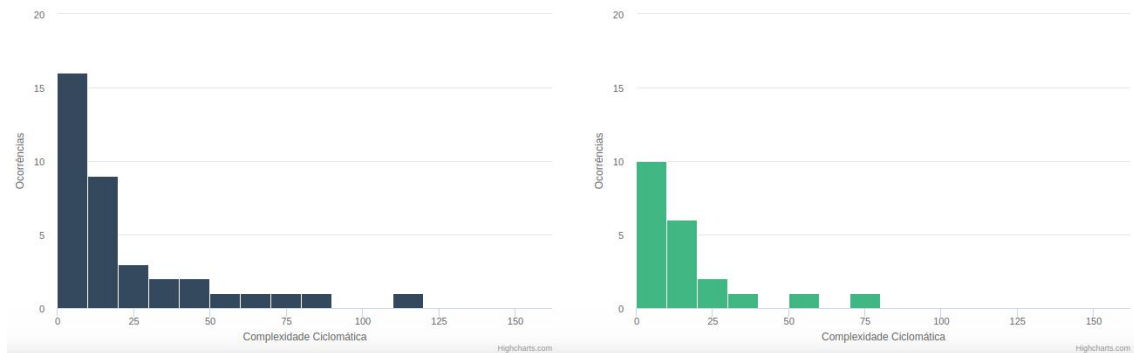
Type	Name	Weight
class	Slim\Handlers\Error	320.36
class	Slim\Handlers\PhpError	302.04
final class	Slim\CallableResolver	224.96
trait	Slim\MiddlewareAwareTrait	196.84

Developer	Commits	Inserts	Deletions
Josh Lockhart <joshlockhart@gmail.com>	792	16946	11581
Rob Allen <rob@akrabat.com>	475	11567	2120
Josh Lockhart <info@joshlockhart.com>	310	4026	3562
= <info@joshlockhart.com>	251	8444	6173

**Figura 4:** Resultado rápido da análise do *branch 4.x* do *framework Slim*

Nesse resultado, são destacados os piores arquivos encontrados, além de uma tabela com os principais colaboradores daquele projeto. Com isso, se torna simples o processo de revisão de código e checagem da garantia de qualidade, pois em poucos segundos se tem um resumo das partes que merecem atenção no projeto.

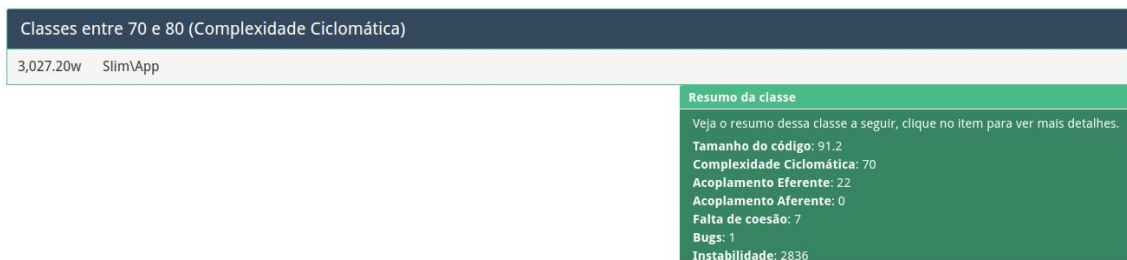
É possível apresentar esses resultados em um sistema de gráficos, o *Insphptor* disponibiliza por padrão um sistema nomeado *Insphptor Overview*, que tem como objetivo ler os arquivos exportados pelo comando *run:export* e apresentá-los em histogramas. A Figura 5 mostra a comparação de complexidade entre os códigos dos *branches 3.x* e *4.x* do *Slim*:



**Figura 5:** Comparação de histogramas de complexidade ente 3.x e 4.x

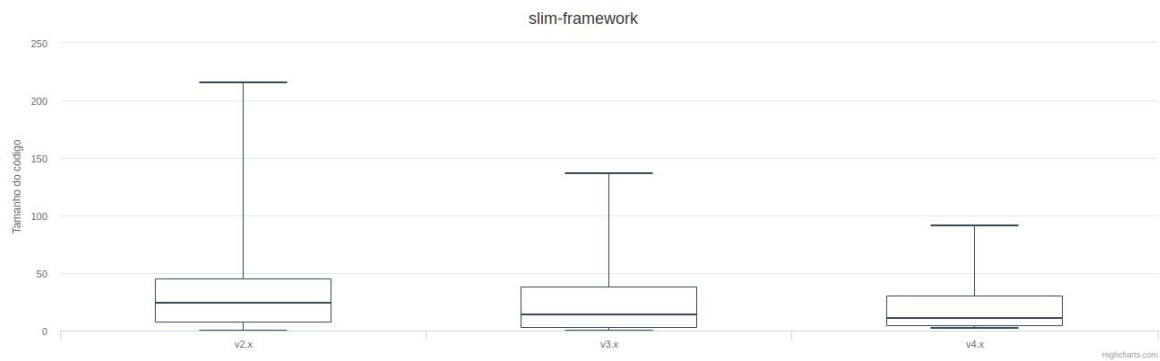
Se torna evidente nos gráficos da Figura 5 que houve uma grande redução da complexidade do código entre essas duas versões, apesar da quantidade de arquivos diminuir, nota-se que a maior complexidade ciclomática no branch 3.x encontra-se acima de 100, enquanto que no branch 4.x está em 70. Observa-se ainda uma diminuição nas ocorrências de complexidades ciclomáticas acima de 40.

Para uma análise mais aprofundada, é possível clicar em qualquer barra do histograma e analisar as classes que fazem parte do intervalo representado, como mostra a Figura 6. Nessa tela, são apresentados o valor de cada métrica da classe em destaque:



**Figura 6:** Resumo da classe *Slim\Handlers\Error* mostra as métricas dela

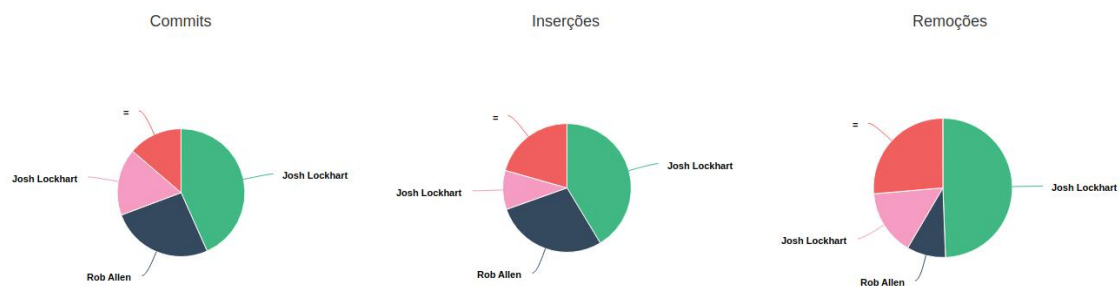
Caso seja necessária uma comparação direta entre os elementos o *Insphtor Overview* oferece um diagrama *boxplot* que compara, em um mesmo ambiente, as cinco últimas versões do projeto em avaliação. A Figura 7 apresenta o *boxplot* que compara o tamanho do código das várias versões do *Slim*:



**Figura 7:** *Boxplot* comparando o tamanho de código entre versões do *Slim*

No gráfico apresentado, é notória a redução do tamanho do código do projeto, mostrando que as mudanças feitas foram essenciais para atingir um código limpo.

Por fim, gráficos de pizza foram usados para realizar a comparação do trabalho realizado pelos colaboradores do projeto. A Figura 8 apresenta a equipe de desenvolvimento da versão 4.x do *Slim*:



**Figura 8:** Gráficos de pizza mostram a eficiência e trabalho dos colaboradores

Visualizando a Figura 8, toma-se a conclusão que *Josh Lockhart* é o colaborador mais ativo do sistema, talvez pelo tempo em que já trabalha no projeto. Essa identificação mostra que uma possível interação entre esse desenvolvedor e outro que não contribui tanto, pode aumentar a produtividade da equipe.

É possível constatar que o mesmo, aparece três vezes em um mesmo gráfico correspondente, isso se dá pelo uso de diversos *emails* distintos pelo mesmo desenvolvedor, o que impossibilita a união dos resultados.

## 4. CONCLUSÃO

O *Insphtor* é capaz de calcular diversas métricas de software escrito na linguagem PHP e, através da aplicação de visualização de dados estatísticos permite analisar a evolução e identificar problemas de qualidade de software. Os resultados demonstraram essa capacidade através do estudo de caso com o framework Slim.

A ferramenta está disponível para o público através da plataforma *Packgist*. É possível realizar a instalação da mesma a partir do único comando: `composer global require leonardovilarinho/insphptor` e em seguida avaliar um projeto conforme descrito na seção 2.1 desse documento. Seu código também está disponível online (LEONARDO, 2017b).

## 5. REFERÊNCIAS

ANICHE, M. Orientação à Objetos e SOLID para Ninjas. 2015.

BALOGH, G.; BESZÉDES, Á. CodeMetropolis - A minecraft based collaboration tool for developers. **2013 1st IEEE Working Conference on Software Visualization - Proceedings of VISSOFT 2013**, 2013.

MARTIN, R. C.; MICAH, M. **Agile Principles, Patterns, and Practices in C#**. [s.l: s.n.].

MCCABE, T. J. A Complexity Measure. **IEEE Transactions on Software Engineering**, v. SE-2, n. 4, p. 308-320, 1976.

MEIRELLES, P. Monitoramento de métricas de código-fonte em projetos de software livre. 2013.

MURPHY, G. C. **Big data. Little Brain.**, 2005.

SCHNEIDEWIND, N. Software metrics model for quality control.

**Software Metrics Symposium**, p. 127-136, 1997.

SOKOL, F. Z.; ANICHE, M. F.; GEROSA, M. A. MetricMiner: uma ferramenta web de apoio à mineração de repositórios de software. **IEEE 13th International Working Conference on Source Code Analysis and Manipulation, SCAM 2013**, p. 142-146, 2013.

THOMAZ ALMEIDA, L.; MACHINI DE MIRANDA, J. Código Limpo e seu Mapeamento para Métricas de Código Fonte. 2010.

WALTERS, G. F.; MCCALL, J. A. Software Quality Metrics for Life-Cycle Cost-Reduction. **IEEE Transactions on Reliability**, v. R-28, n. 3, p. 212-220, 1979.

WETTEL, R. Visual exploration of large-scale evolving software. **2009 31st International Conference on Software Engineering - Companion Volume, ICSE 2009**, p. 391-394, 2009.

VILARINHO, L. Clone Slim Framework, 2017a, disponível em: <https://github.com/leonardovilarinho/slim>.

VILARINHO, L. Repositório do Insphptor no Github, 2017b, disponível em: <https://github.com/leonardovilarinho/insphptor>.

## ANEXO 1: ALGORITMOS DE MÉTRICAS

```
// Cálculo usado para o acoplamento aferente

$afferent = 0;

$classname = $class->namespace . '\\\\' . $class->name;

$classes = ClassesRepository::instance();

// Percorre as classes do sistema

foreach ($classes() as $current) {

    // Se a classe analisada estiver no array de dependencias da
    // classe percorrida, adiciona um ao acoplamento

    if (in_array($classname, $current->dependencies))

        $afferent ++;

}
```

---

```
// Cálculo para métrica de falta de coesão

$cohesion = count($class->methods);

$methodVariables = [];

$exclude = 0;

// percorre cada método da classe

foreach ($class->methods as $key => $method) {

    $valid = 0;

    $name = '';

    $prefix = substr($method['name'], 0, 3);

    $prefix2 = substr($method['name'], 0, 4);

    $prefix = strtolower($prefix);
```



```

        // Verifica se o método não é um getter ou setter
        if (!in_array($prefix, ['get', 'set']) and $prefix2 != 'push')
    {

        // Percorre os tokens do método a procura das palavras
        // reservadas $this e self, para identificar o uso de
        // atributos, no final, se há atributo detectado irá
        // colocar seu nome em methodVariables

        foreach ($method['content'] as $token) {

            if (in_array($token[0], [T_VARIABLE, T_STRING]))

                if (in_array($token[1], ['$this', 'self']))

                    $valid = 1;

            if ($valid > 0 and $valid < 4) {

                $name .= isset($token[1]) ? $token[1] : '';

                $valid ++;

                if ($valid == 4) {

                    $methodVariables[$key][] = $name;

                    $name = '';

                    $valid = 0;

                }

            }

        }

    } else

        $exclude ++;

}

```

```

// retira os getters e setters da coesão

$cohesion -= $exclude;

// Percorre as variáveis dos métodos e o compara com os outros
// se encontrar o mesmo atributo sendo usado em outro método
// decrementa a coesão em 1

foreach ($methodVariables as $key1 => $method) {
    $isFind = false;

    foreach ($method as $variable)
        foreach ($methodVariables as $key2 => $compare)
            if ($key2 > $key1 and in_array($variable,
                $compare))
                $isFind = true;

    if ($isFind)
        $cohesion --;
}

```

---

```

// Cálculo da métrica de complexidade ciclomática

$complexity = 0;

// Percorre os métodos da classe e seu conteúdo,
// se o token atual pertence a um array de tokens pré-definidos,
// de comandos de desvio do PHP, irá incrementar a complexidade

foreach ($class->methods as $method) {
    $complexity ++;

    foreach ($method['content'] as $token)
        if (is_array($token))
            if (\in_array($token[0], self::$included))
                $complexity ++;
}

```

```
}
```

---

```
// Cálculo para métrica de acoplamento eferente
```

```
$efferent = 0;
```

```
// Percorre os tokens da classe, se o token estiver
```

```
// incluído em um array pré-definido com comandos
```

```
// de importação do PHP, irá incrementar a métrica
```

```
foreach ($class->token as $token)
```

```
    if (\is_array($token))
```

```
        if (\in_array($token[0], self::$included))
```

```
            $efferent ++;
```

---

```
// Cálculo para métrica de tamanho de código
```

```
$size = 0;
```

```
// Percorre os métodos da classe e então incrementa o tamanho
```

```
// do método a eles
```

```
foreach ($class->methods as $method)
```

```
    $size += isset($method['content'])
```

```
        ? count($method['content'])
```

```
        : 0;
```

## **ANEXO 2: ARQUIVO YML GERADO PARA O SLIM FRAMEWORK**

name: slim-framework

export: json

git: 'yes'

rank: '4'

hide:

- interface

- file

only:

- Slim

views:

- overview: insphptor-overview