1. Introdução

A utilização de medidas de software, auxilia o engenheiro de software a conhecer o produto que está sendo desenvolvido e como ele é estruturado, qual a complexidade para alterações no desenvolvimento e também sua qualidade [Abílio, 2013]. Para realização dessa tarefa, utiliza-se ferramentas existentes como PHPLoc, PHPStan e PHPMetrics. Também é possível realizar analises do software em busca de possíveis problemas com o código (PMD). Esta tarefa é realizada através de algumas ferramentas como o Qafoo, PHPMD, PHP Code Sniffer, dentre outras.

As medidas também são uteis para acompanhar a evolução do produto [Ribeiro *et al*, 2012]. Pode-se verificar o crescimento ou a diminuição do software através de medidas como número de linhas de código de versões diferentes do produto após manutenções. [Abílio, 2013]. Com estes objetivos, a ferramenta PHPMetrics foi desenvolvida e pode ser utilizada como um plugin do PHPStorm, por exemplo. Ela realiza medições de códigos em PHP, utilizando 27 medidas e apresenta formas de visualização das medidas, como gráficos e tabelas.

O PHPMetrics e o Qafoo realizaram analises das últimas 5 versões do software salonERP, com o intuito de verificar a evolução do sistema e detectar problemas de código. As versões que estavam disponíveis no repositório, não apresentavam os arquivos de mudanças.

2. Descrição das ferramentas

2.1. PHPMetrics

O PHPMetrics pode ser instalado globalmente com o gerenciador de pacotes de preferência do usuário ou como um plugin da ferramenta PHPStorm, que foi o caso deste estudo. Os resultados gerados, são exibidos em forma de relatórios pelos navegadores. Podemos visualizar os resultados através de 5 opções: *Overview, Evaluation, Relations map, Repartition* e *Explore*.

Na Figura 1, está a visualização da tela *Overview*:

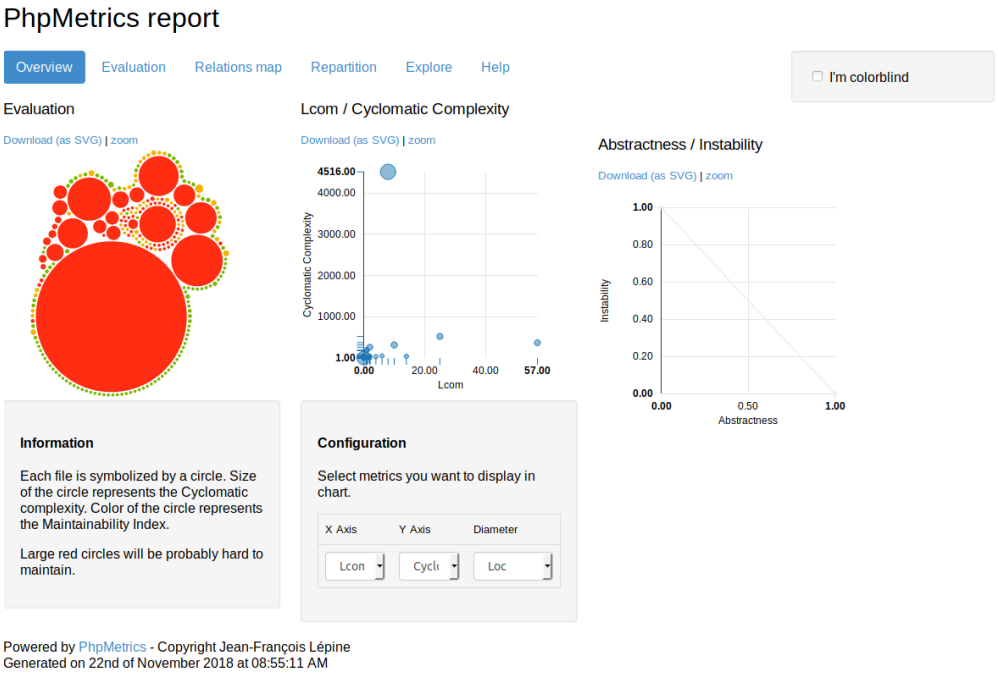


Figura 1: Visualização da tela *Overview*

O primeiro gráfico da tela *Overview*, mostra cada arquivo do projeto simbolizado por círculos. O tamanho do círculo representa a complexidade ciclomática e as cores representam o índice de manutenibilidade do software. Quanto maior o círculo vermelho, maior probabilidade de uma difícil manutenção [PHPMetrics].

O segundo gráfico, permite a configuração de três métricas que serão exibidas por ele, permitindo a configuração das mesmas. Pode ser escolhido métricas para o eixo X, eixo Y e diâmetro do gráfico. O gráfico exibe círculos que também representam os arquivos do projeto, o tamanho desses círculos variam de acordo com a métrica selecionada na configuração de diâmetro e a posição dos círculos no gráfico é definida pelas configurações de métricas dos eixos X e Y.

O terceiro gráfico, utiliza a métrica *Abrstractness* como eixo X, e *Instability* como eixo Y. A métrica *Abstractness* representa o número de classes abstratas e a métrica *Instability,* a resiliência das classes.

A Figura 2 mostra a tela de *Evaluation* :

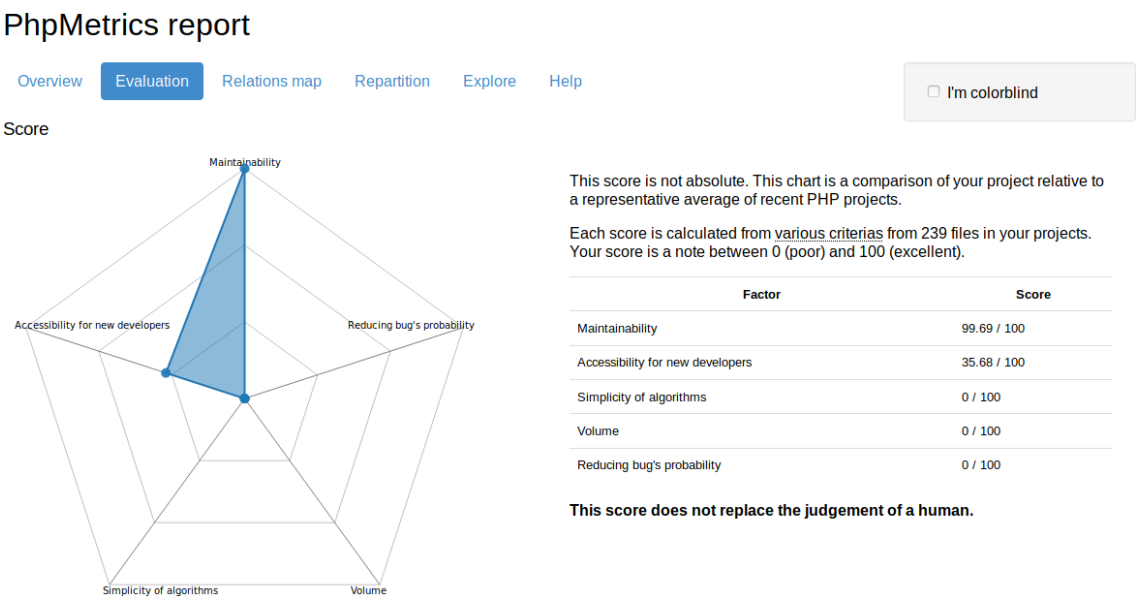


Figura 2: Visualização da tela *Evaluation*

O gráfico exibido nesta tela, pontua algumas características do software, que foram geradas pelas estatísticas das métricas, sendo estas características: manutenibilidade, acessibilidade para novos desenvolvedores, simplicidade dos algoritmos, volume e redução da probabilidade de bugs. A tabela exibida ao lado do gráfico, faz uma comparação da pontuação entre o produto de software avaliado e uma média representativa dos projetos recentes de PHP [PHPMetrics].

A Figura 3, representa a tela *Relations map:*

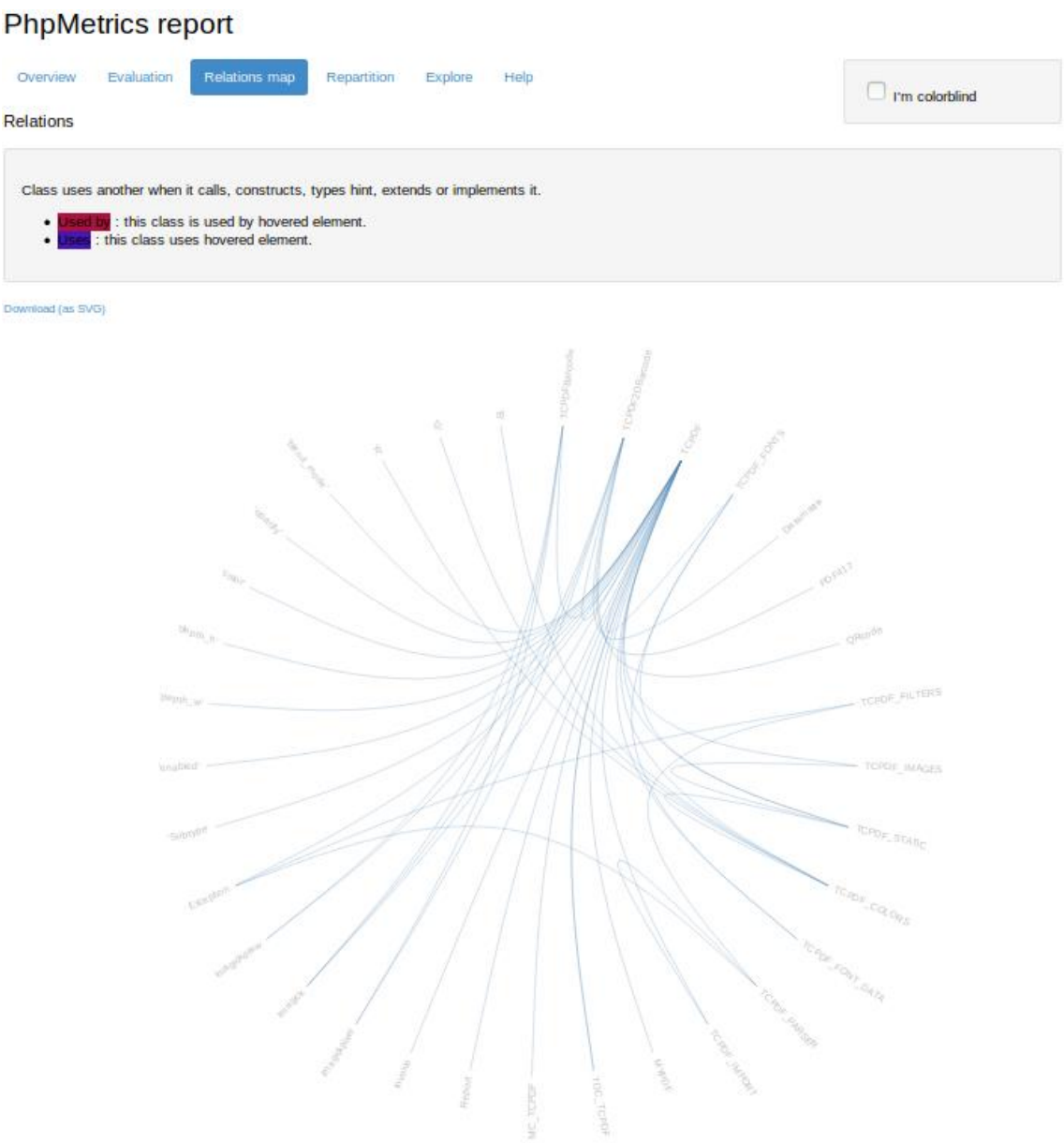


Figura 3: Visualização da tela *Relations map*

Esta tela, mostra o mapa de relação das classes. Uma classe utilizando a outra. Esta dividia em 2 tipos de relação:

Usada por (cor vermelha) - quando a classe é usada pelo elemento.

Usa (cor azul) – quando ela usa o elemento.

A Figura 4, apresenta a tela de *Repartitions:*

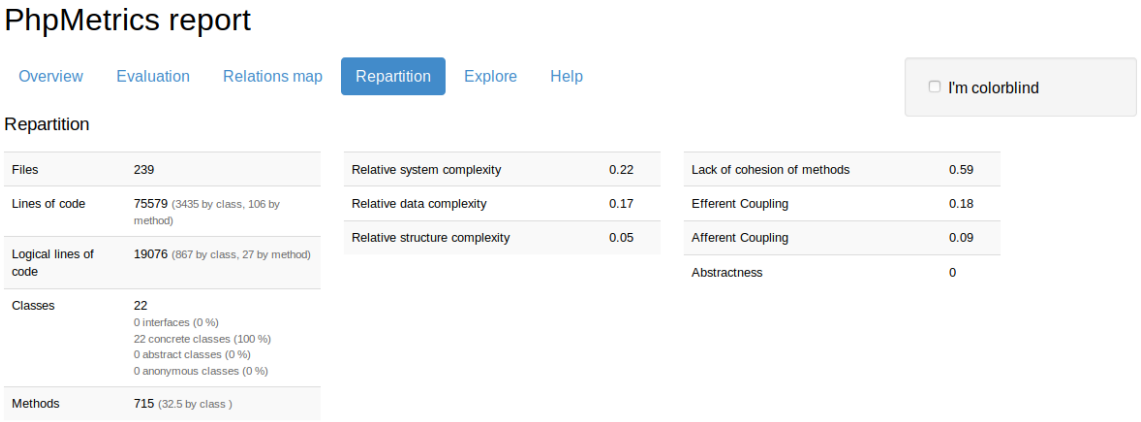


Figura 4: Visualização da tela *Repartitions*

Este relatório mostra o número de arquivos do projeto e também exibe os resultados das métricas. Alguns destes resultados, estão separados por repartições que as métricas podem ser agrupadas.

A Figura 5, mostra a tela *Explore*:

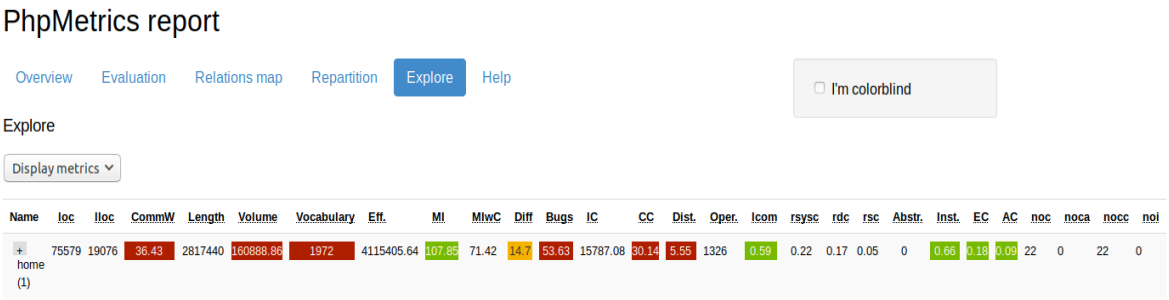


Figura 5: Visualização da tela *Explore*

A tela *Explore* exibe uma tabela que representa todas as métricas e o resultado da medição de cada uma delas. Também pode ser alterada para exibir somente as métricas essenciais.

A ferramenta também possui um menu para tela *Help,* como pode ser visto em todas as figuras. Essa tela contém informações dos recursos da ferramenta que estão disponíveis no GitHub, recurso de ajuda online em relação a problemas encontrados com a ferramenta e também informações sobre o desenvolvedor da mesma.

A ferramenta possui 27 métricas. A Tabela 1 informa quais são as métricas e a descrição de cada uma:

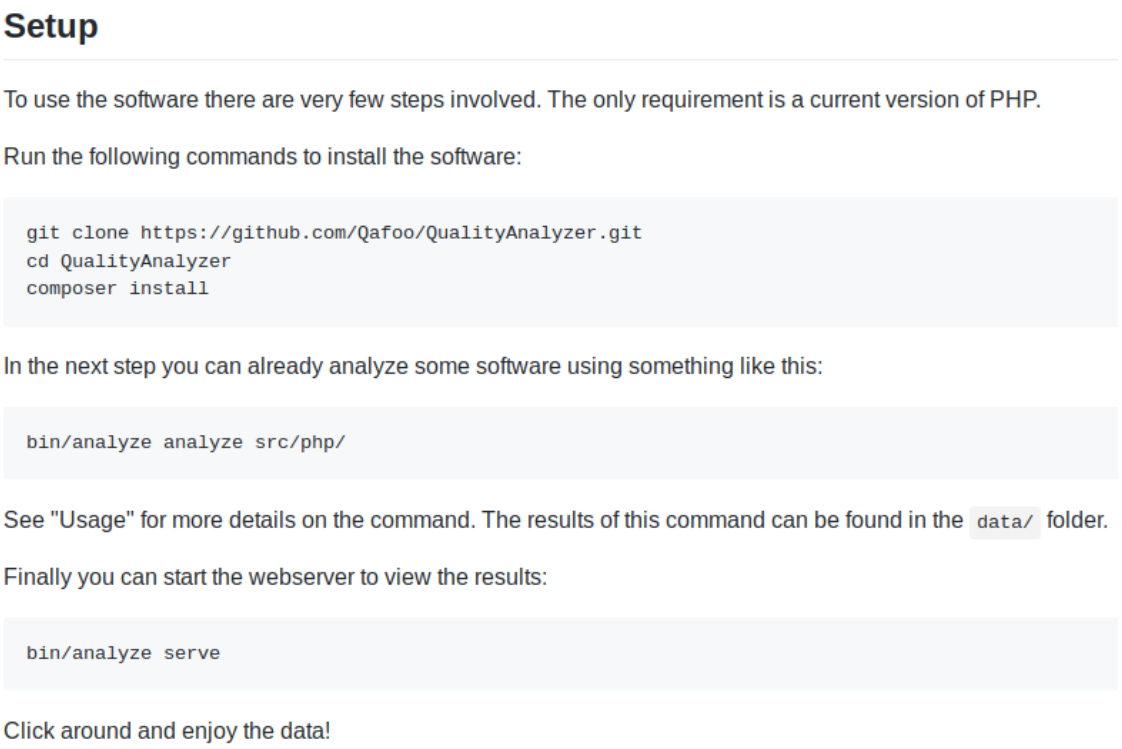
Métrica	**Descrição**
Ca (CA)	Número de classes afetada por essa classe
Bugs	Número estimado de bugs por arquivo
Commw	Mede a razão entre o código lógico e os comentários
Cc	Número de complexidade ciclomática
Dc	Complexidade de dados

Diff	Dificuldade do código
Ce (EC)	Número de classes que a classe depende
Effort	Esforço para entender o código
Instability	Indica a resiliência da classe
IC	Conteúdo inteligente
Icom	Falta de coesão de métodos e classes
Length	Tamanho do código
Loc	Número de linhas de código
Lloc	Número de linhas lógicas de código
MI	Índice de manutenibilidade
MIwC	Índice de manutenibilidade sem os comentários de código
Distance	Distância de Myer, derivada da complexidade ciclomática
Interval	Intervalo de Myer, indica a distância entre o número de complexidade ciclomática e o número de operadores
Noc	Número de classes
Noca	Número de classes abstratas
Nocc	Número de classes concretas
Rdc	Complexidade de dados relativos
Rsc	Complexidade estrutural relativa
Rsysc	Complexidade total do Sistema
Time	Tempo para ler e entender o código
Vocabulary	Vocabulário usado no código
Volume	Volume de código

Tabela 1: Métricas utilizadas pela ferramenta.

2.2. Qafoo

O Qafoo é uma ferramenta para análise de códigos PHP. Com ela podemos realizar tanto a análise de métricas quanto analise PMD, detecção de bugs, dentre outras funcionalidades. Neste estudo, foi utilizado apenas a funcionalidade PMD da ferramenta. A imagem abaixo mostra os passos para a instalação da ferramenta (Linux) e como executa-la para se obter os resultados:



Setup

To use the software there are very few steps involved. The only requirement is a current version of PHP.

Run the following commands to install the software:

```
git clone https://github.com/Qafoo/QualityAnalyzer.git
cd QualityAnalyzer
composer install
```

In the next step you can already analyze some software using something like this:

```
bin/analyze analyze src/php/
```

See "Usage" for more details on the command. The results of this command can be found in the `data/` folder.

Finally you can start the webserver to view the results:

```
bin/analyze serve
```

Click around and enjoy the data!

Figura 6: Instalação e execução da ferramenta Qafoo

Como pode ser observado na Figura 6, os resultados são exibidos através de um Webserver que é criado pela ferramenta que mostra o conteúdo da pasta /data gerada após a análise de código. Depois de executar o comando “bin/analyze serve” o Webserver fica ativo e podemos acessar o resultado pelo endereço local na porta 8080 (http://localhost:8080):

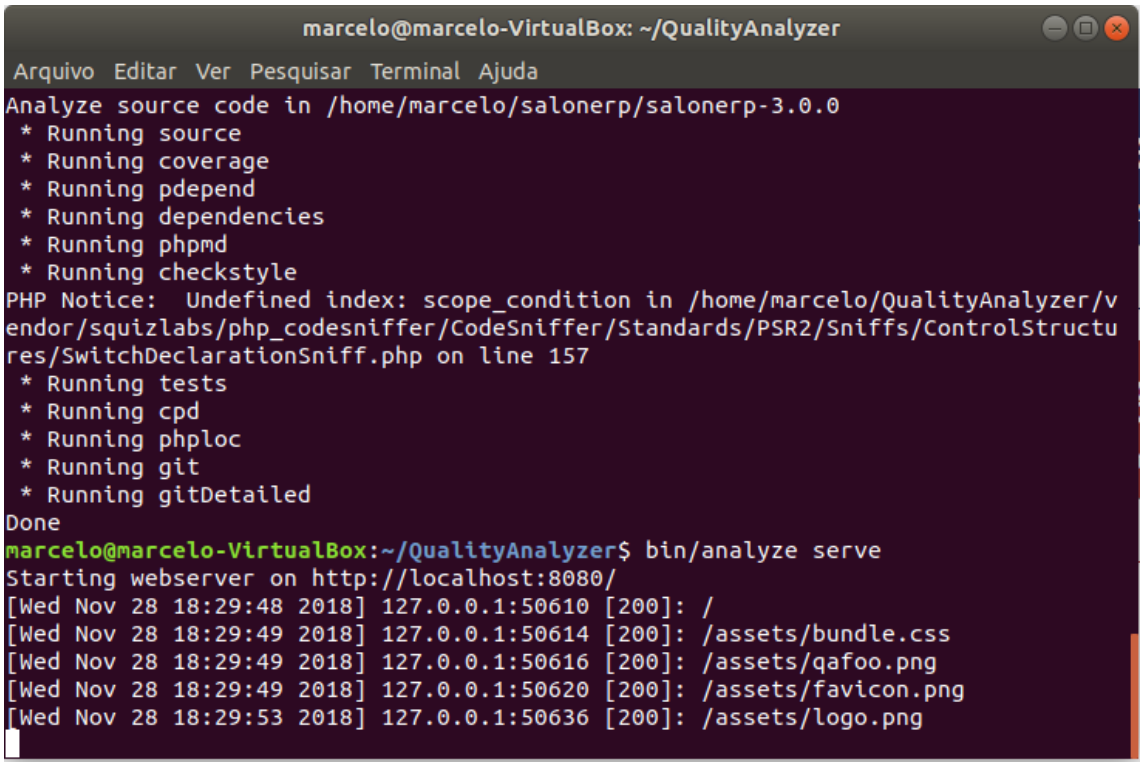


Figura 7: Webserver em execução

A primeira tela exibida ao acessar o endereço, mostra todas as análises contidas na ferramenta, como mostra a figura abaixo:

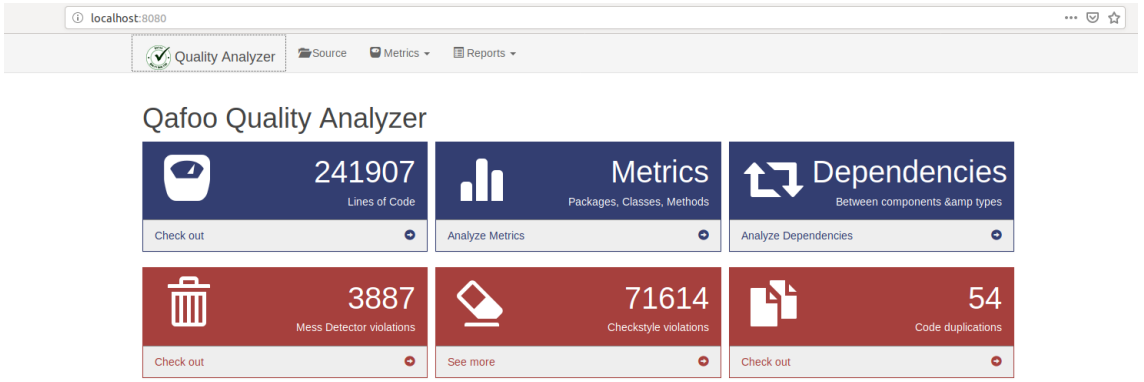


Figura 8: Primeira tela dos resultados da ferramenta Qafoo

Para este estudo, foram analisados apenas os resultados da opção "*Mess Detector Violations*", e a forma de visualização dos resultados é como mostra a figura abaixo:

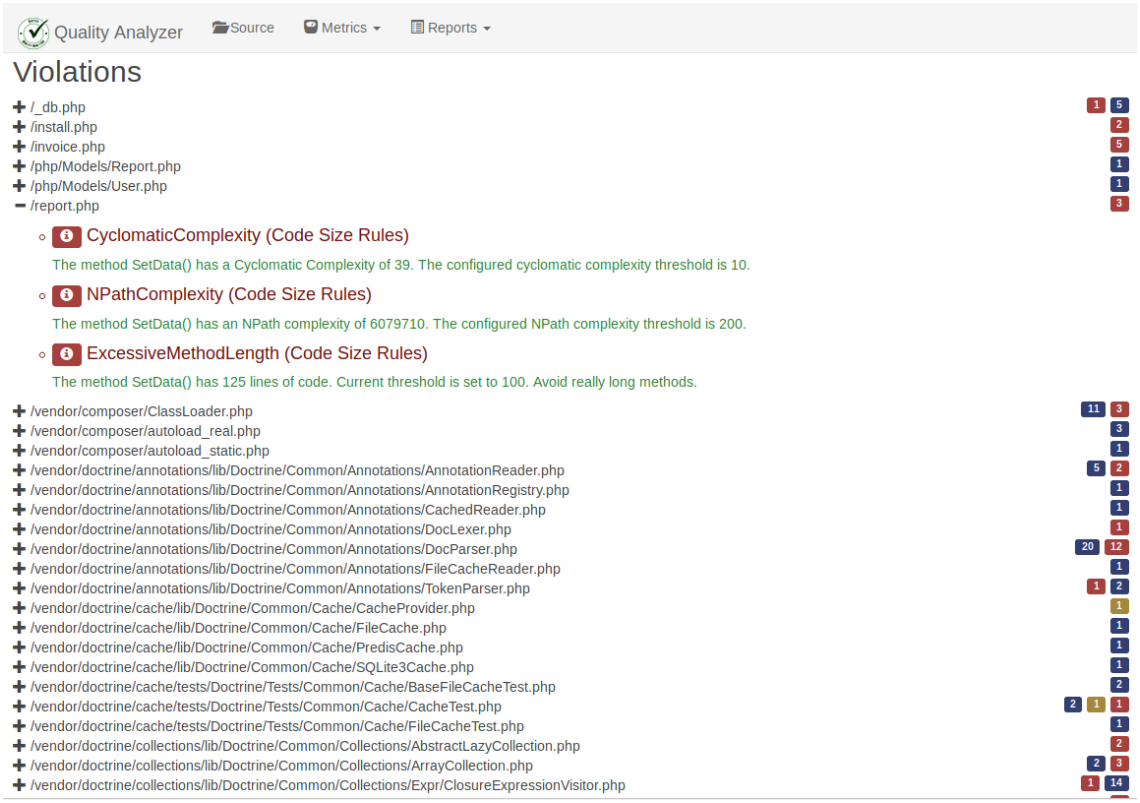


Figura 9: Resultados da ferramenta Qafoo (Mess Detect Violations)

Os resultados são exibidos arquivo por arquivo, e podemos expandi-los para visualização mais detalhada. Ao lado direito, mostra a quantidade de problemas encontrados, onde cada cor representa uma categoria (Clean Code Rules, Code Size Rules, Controversial Rules, Design Rules, Naming Rules e Unused Code Rules) de problema diferente. Cada categoria possui diversas “Regras” e estão listadas na tabela a seguir:

Regra	Descrição
BooleanArgumentFlag (Clean Code Rules)	Indicador confiável para violação do Principio de Responsabilidade Única (SRP).
ElseExpression (Clean Code Rules)	Uma expressão if com uma ramificação else nunca é necessária.
StaticAccess (Clean Code Rules)	O acesso estático causa dependências imutáveis a outras classes e leva a códigos difíceis de testar
CyclomaticComplexity (Code Size Rules)	A complexidade é determinada pelo número de pontos de decisão em um método. Geralmente, 1-4 é de baixa complexidade, 5-7 indica complexidade moderada, 8-10 é de alta complexidade e 11+ é de complexidade muito alta.
NPathComplexity (Code Size Rules)	A complexidade do NPath de um método é o número de caminhos de execução acíclicos através desse método
ExcessiveMethodLength (Code Size Rules)	Violações desta regra geralmente indicam que o método está fazendo muito. Tente reduzir o tamanho do método criando métodos auxiliares e removendo qualquer código copiado / colado.
ExcessiveClassLength (Code Size Rules)	Arquivos com classes extensas são indicações de que a classe pode estar tentando fazer muito. Tente dividi-lo e reduza o tamanho para algo gerenciável.

ExcessiveParameterList (Code Size Rules)	Listas de parâmetros longos podem indicar que um novo objeto deve ser criado para envolver os diversos parâmetros. Basicamente, tente agrupar os parâmetros juntos.
ExcessivePublicCount (Code Size Rules)	Um grande número de métodos e atributos públicos declarados em uma classe pode indicar que a classe pode precisar ser dividida, pois será necessário um esforço maior para testá-la completamente
TooManyFields (Code Size Rules)	As classes que possuem muitos campos podem ser reprojetadas para ter menos campos, possivelmente por meio de algum agrupamento de objetos aninhados de algumas das informações. Por exemplo, uma classe com campos de cidade / estado / cep pode ter um campo de endereço.
TooManyMethods (Code Size Rules)	Uma classe com muitos métodos é provavelmente um bom suspeito para a refatoração, a fim de reduzir sua complexidade e encontrar uma maneira de ter objetos mais refinados. Por padrão, ele ignora os métodos que começam com 'get' ou 'set'.
TooManyPublicMethods (Code Size Rules)	Uma classe com muitos métodos públicos é provavelmente um bom suspeito para a refatoração, a fim de reduzir sua complexidade e encontrar uma maneira de ter objetos mais refinados. Por padrão, ele ignora os métodos que começam com 'get' ou 'set'.
ExcessiveClassComplexity (Code Size Rules)	O Weighted Method Count (WMC) de uma classe é um bom indicador de quanto tempo e esforço são necessários para modificar e manter essa classe. A métrica WMC é definida como a soma das complexidades de todos os métodos declarados em uma classe. Um grande número de métodos também significa que essa classe tem um impacto potencial maior em classes derivadas.
Superglobals (Controversial Rules)	Acessar uma variável super global diretamente é considerado uma prática ruim. Essas variáveis devem ser encapsuladas em objetos que são fornecidos por uma estrutura, por exemplo.
CamelCaseClassName (Controversial Rules)	É considerado uma boa prática usar a notação CamelCase para nomear classes.
CamelCasePropertyName (Controversial Rules)	É considerado uma boa prática usar a notação camelCase para nomear atributos.
CamelCaseMethodName (Controversial Rules)	É considerado uma boa prática usar a notação camelCase para nomear métodos.
CamelCaseParameterName (Controversial Rules)	É considerado uma boa prática usar a notação camelCase para nomear parâmetros.
CamelCaseVariableName (Controversial Rules)	É considerado uma boa prática usar a notação camelCase para nomear variáveis.
ExitExpression (Design Rules)	Uma expressão de saída dentro do código regular não pode ser testada e, portanto, deve ser evitada. Considere mover a expressão de saída para algum tipo de script de inicialização em que um código de erro / exceção é retornado para o ambiente de chamada.
EvalExpression (Design Rules)	Uma expressão “eval” não pode ser testada, um risco de segurança e uma má prática. Por isso, deve ser evitado. Considere substituir a expressão “eval” por código regular.
GotoStatement (Design Rules)	Goto torna o código mais difícil de ler e é quase impossível entender o fluxo de controle de um aplicativo que usa essa construção de linguagem. Por isso, deve ser evitado. Considere substituir o Goto por estruturas de controle regulares e métodos / funções separados, que são mais fáceis de ler.

NumberOfChildren (Design Rules)	Uma classe com um número excessivo de filhos é um indicador para uma hierarquia de classes desequilibrada. Você deve considerar refatorar essa hierarquia de classes.
DepthOfInheritance (Design Rules)	Uma classe com muitos pais é um indicador para uma hierarquia de classes desequilibrada e errada. Você deve considerar refatorar essa hierarquia de classes.
CouplingBetweenObjects (Design Rules)	Uma classe com muitas dependências tem impactos negativos em vários aspectos de qualidade de uma classe. Isso inclui critérios de qualidade como estabilidade, capacidade de manutenção e capacidade de compreensão.
DevelopmentCodeFragment (Design Rules)	Funções como var_dump (), print_r () etc. são normalmente usadas apenas durante o desenvolvimento e, portanto, tais chamadas no código de produção são um bom indicador de que elas foram esquecidas.
ShortVariable (Naming Rules)	Detecta quando um campo, local ou parâmetro tem um nome muito curto.
LongVariable (Naming Rules)	Detecta quando um campo, variável formal ou local é declarado com um nome longo.
ShortMethodName (Naming Rules)	Detecta quando nomes de métodos muito curtos são usados.
ConstructorWithNameAsEnclosingClass (Naming Rules)	Um método construtor não deve ter o mesmo nome que a classe envolvente, considere usar o método de construção PHP 5
ConstantNamingConventions (Naming Rules)	Nomes de constantes de classe / interface devem sempre ser definidos em maiúsculas.
BooleanGetMethodName (Naming Rules)	Procura métodos chamados 'getX ()' com 'boolean' como o tipo de retorno. A convenção é nomear esses métodos como 'isX ()' ou 'hasX ()'.
UnusedPrivateField (Unused Code Rules)	Detecta quando um campo privado é declarado e / ou atribuído a um valor, mas não é usado.
UnusedLocalVariable (Unused Code Rules)	Detecta quando uma variável local é declarada e / ou atribuída, mas não é usada.
UnusedPrivateMethod (Unused Code Rules)	Detecta quando um método privado é declarado, mas não é usado.
UnusedFormalParameter (Unused Code Rules)	Evite passar parâmetros para métodos ou construtores e, em seguida, não usar esses parâmetros.

Tabela 2: Descrição das regras Qafoo (Mess Detect Violations)

As regras que o Qafoo utiliza para gerar os resultados de "*Mess Detector Violations"* são as regras da ferramenta PHPMD, com a diferença que os resultados são exibidos de outra maneira mais fácil de ser visualizada pelo site que o Webserver ativa com ferramenta Qafoo, enquanto o PHPMD exibe os resultados no terminal.

2.3. Descrição do Software SalonERP

O SalonERP é um pequeno sistema ERP que foi projetado para atender salões de beleza. A página principal do software é um calendário que mostra uma visão geral dos compromissos. O calendário pode ser alternado para uma visão geral diária ou mensal. O Software também inclui páginas para gerenciar clientes e produtos. É possível criar relatórios como receita, melhores clientes, entre outros. É compatível com os idiomas Inglês e Alemão.

Características do produto:

Interface de calendário; Gerenciamento de clientes e produtos; Especialmente projetado pra salões de beleza; possui 2 idiomas; Inclui gestão básica; Inclui Relatórios e Relatórios BI; Visualização gráfica de diferentes componentes; Customizável (aparência); Gera faturas; Compatível com smartphones e tablets; Pode ser instalado com SQLite ou MySQL; Disponibilidade de contato com o desenvolvedor para recursos adicionais a ser implementado; Faturas personalizáveis para impressoras de recibo.

As Figuras 8 e 9 mostram a tela de login e a tela inicial do programa, respectivamente.

Figura 10: Tela de login do SalonERP

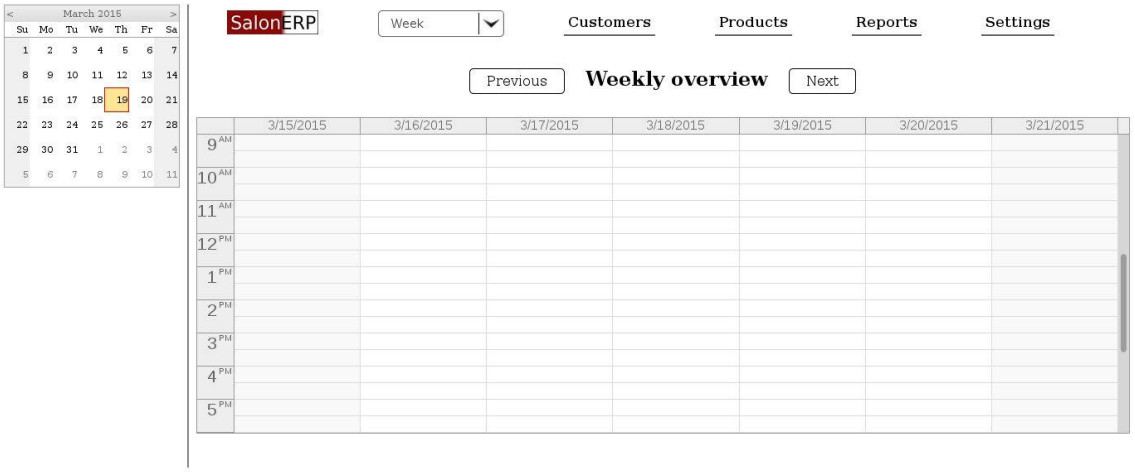


Figura 11: Tela inicial do programa

A tela de login é composta por *Database Type*, onde se seleciona MySQL ou SQLite. Abaixo está o campo *MySQL Server*, que é preenchido com o endereço do banco de dados. Depois vem o campo *MySQL Database*, onde se preenche o nome da base de dados que será acessada. *MySQL User* deve conter o usuário do banco de dados e *MySQL Password* a senha do banco de dados. Os ultimos 3 campos a serem preenchidos são referentes ao usuário do sistema e a sua senha, que deve ser repetida.

A tela inicial mostra um pequeno calendário do lado esquerdo e ao lado direito, mostra a agenda e os compromissos e horários agendados nela para o dia que está selecionado no calendário. Onde está selecionado *Week* na tela, é possível selecionar outras opções que a agenda pode exibir: por dia, por semana ou por mês.

3. Configuração do ambiente

Os experimentos foram feitos em uma máquina virtual. A Tabela 3 apresenta as configurações da máquina física e a Tabela 4 apresenta as configurações definidas para a máquina virtual:

Processador	AMD FX™-8350 Eight-Core Processor 4.00 GHz 8MB de cache
HD	SATA 2TB 7200rpm
Memória RAM	16GB
Sistema Operacional	Windows 10 Pro

Tabela 3: Configuração da máquina física

Processador	AMD FX™-8350 4.00 GHz (4 núcleos) 8MB de cache
HD	SATA 100GB 7200rpm
Memória RAM	6GB
Sistema Operacional	Ubuntu 18.04
Máquina Virtual	Oracle VM VirtualBox 5.2.10

Tabela 4: Configuração da máquina virtual

4. Análise Quantitativa

4.1. PHPMetrics

Após os experimentos e a coleta dos valores, realizou-se análise quantitativa para avaliação do comportamento das medidas das versões do software que a ferramenta PHPMetrics nos trouxe. A Figura 12 contém os resultados:

Medidas	Versões				
	1.0	1.1	2.0.0	2.1.0	3.0.0
Loc	75579	75485	76009	76072	3777
Lloc	19076	19023	19148	19158	971
CommW	36,43	36,56	36,67	36,75	18,08
Length	2817400	2817137	2818359	2818433	7294
Volume	160888,86	161557,83	161598,07	160924,61	1966,14
Vocabulary	1972	1980,06	1980,57	1972,31	50,16
Eff.	4115405,64	4132954,58	4138440,98	4121420,88	112874,1
MI	107,85	108,13	108,1	108,58	81,93
MIwC	71,42	71,57	71,43	71,84	63,85
Diff	14,7	14,7	15,05	14,99	16,16
Bugs	53,63	53,85	53,87	53,64	0,65
IC	15787,08	15852,92	15852,99	15786,69	80,81
CC	30,14	30,24	30,55	30,44	11,8
Dist.	5,55	5,57	5,6	5,58	0,72
Oper.	1326	1326	1333	1334	18
Icom	0,59	0,6	0,59	0,59	1,48
Rsysc	0,22	0,22	0,23	0,23	0,81
Rdc	0,17	0,17	0,17	0,17	0,63
Rsc	0,05	0,05	0,06	0,06	0,18
Abstr.	0	0	0	0	0,07
Inst.	0,66	0,66	0,65	0,65	0,74
EC	0,18	0,18	0,18	0,18	0,68
AC	0,09	0,09	0,1	0,1	0,24
Noc	22	22	22	22	15
Noca	0	0	0	0	1
Nocc	22	22	22	22	14
Noi	0	0	0	0	0

Figura 12: Resultados das medições.

Os gráficos e demais resultados apresentados nas Figuras 1, 2, 3, 4 e 5, são prints da tela dos resultados da versão SalonERP 1.0. Como pode-se observar na Figura 8, as versões 1.0, 1.1, 2.0.0 e 2.1.0 apresentaram valores muito parecidos nos resultados, o que torna os gráficos e tabelas gerados pela ferramenta praticamente iguais. Com isso, não houve a necessidade de colocar o gráfico das versões 1.1, 2.0.0 e 2.1.0, mas, a versão 3.0.0 apresentou significativa redução em diversas métricas como Loc, Lloc, CommW, Length, Volume, Vocabulary, entre outras. Observou-se uma redução de código, porém, algumas métricas onde os valores mais baixos são mais interessantes, a

versão 3.0.0 apresentou aumento em relação as demais versões. Exemplos: EC, AC, Inst., Diff, entre outras. As Figuras 13, 14, 15, 16, 17 mostram os gráficos e resultados da versão 3.0.0:

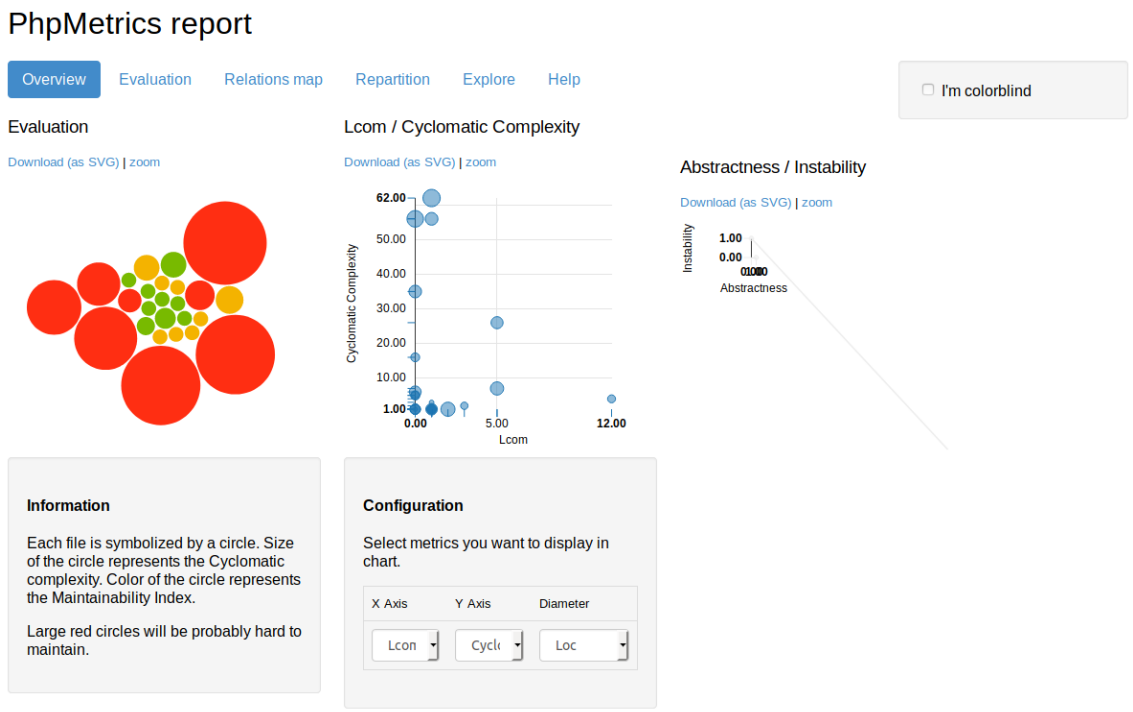


Figura 13: Resultados na tela *Overview* da versão SalonERP 3.0.0

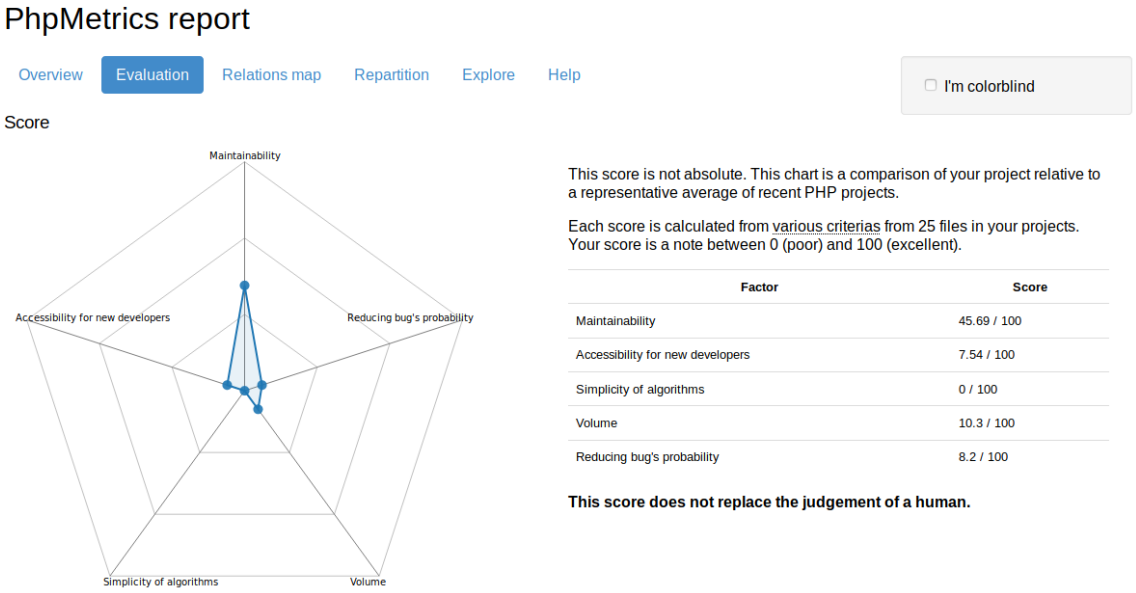


Figura 14: Resultados na tela *Evaluation*da versão SalonERP 3.0.0

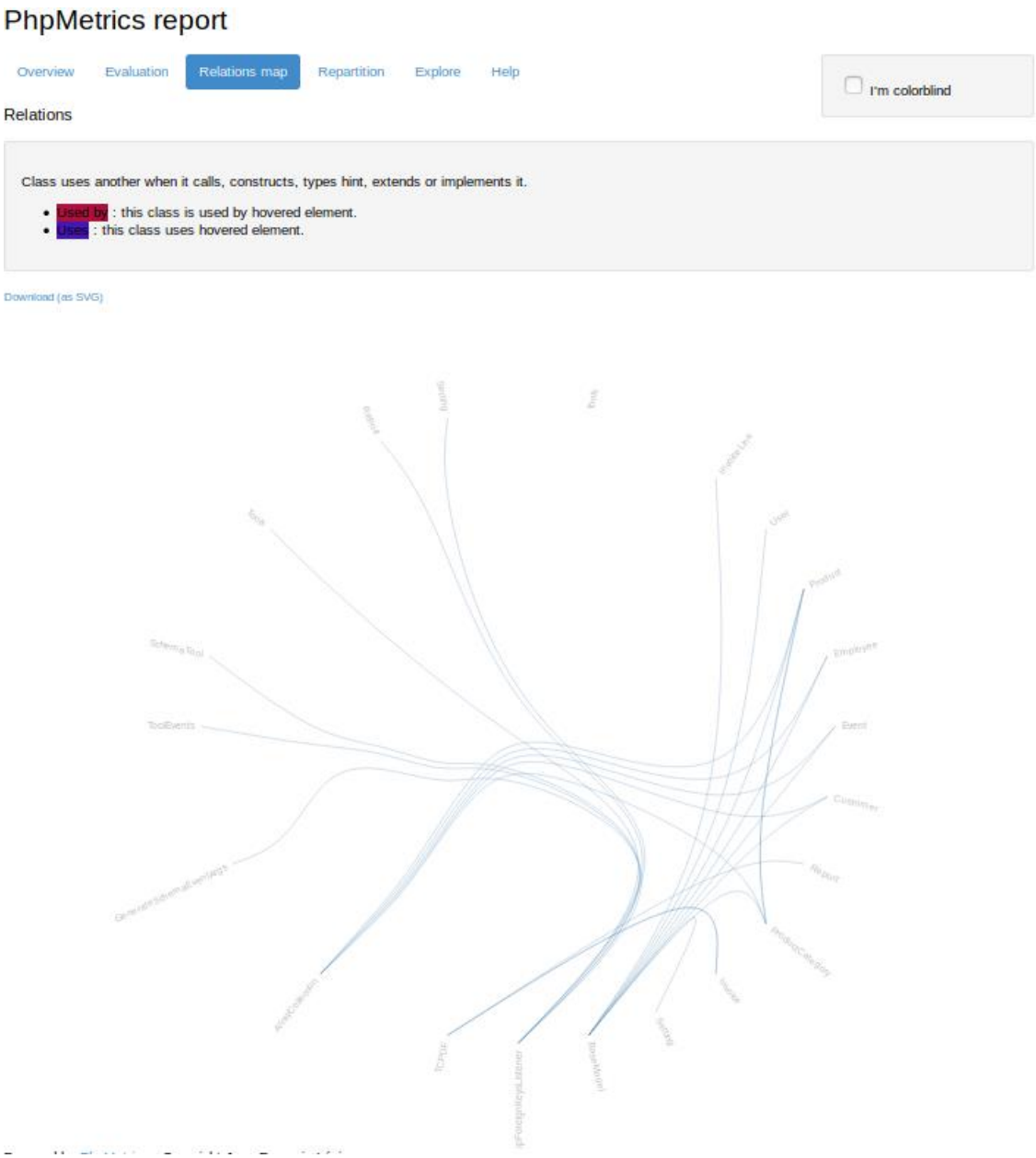


Figura 15: Resultado da tela *Relations map* da versão SalonERP 3.0.0

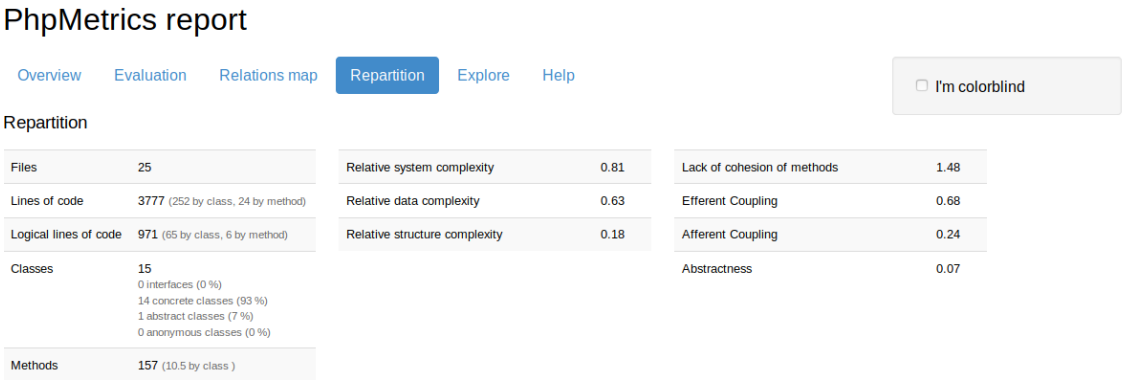


Figura 16: Resultado da tela *Repartition* da versão SalonERP 3.0.0

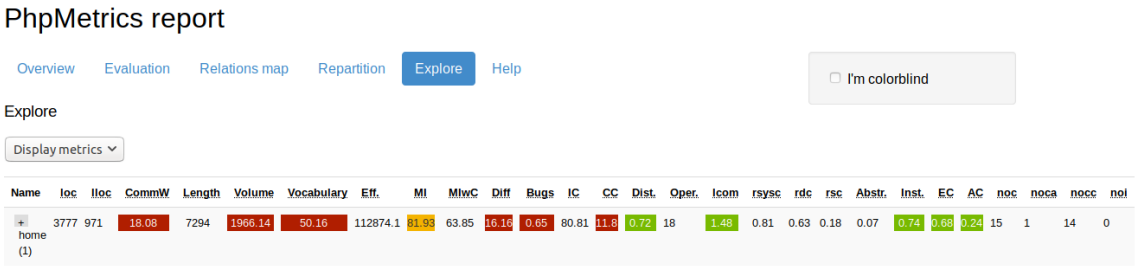


Figura 17: Resultado da tela *Explore* da versão 3.0.0

Ao observar as Figuras 2 e 14, nota-se que apesar da versão SalonERP 3.0.0 apresentar uma significante redução de código das demais versões, o resultado da pontuação exibida no gráfico e na tabela da Figura 14 mostra que o código regrediu em relação as demais versões. Comparando as demais versões com a pontuação da média representativa dos projetos recentes de PHP, elas apresentam um resultado razoável, o que não acontece com a versão 3.0.0, principalmente na manutenibilidade e acessibilidade para novos desenvolvedores. A Tabela 3 mostra a comparação dos resultados da versão 1.0 (que representa as demais) e da versão 3.0.0 em relação aos resultados do gráfico da tela *Evaluation*

	salonERP 1.0	**salonERP 3.0.0**
Manutenibilidade	99.96/100	45.69/100
Acessibilidade para novos desenvolvedores	35.68/100	7.54/100
Simplicidade dos algoritmos	0/100	0/100
Volume	0/100	10.3/100
Redução da probabilidade de bugs	0/100	8.2/100

Tabela 4: Tabela dos resultados da tela *Evaluation*

O gráfico exibido nesta tela, pontua algumas características do software, que foram geradas pelas estatísticas das métricas, sendo estas características: manutenibilidade, acessibilidade para novos desenvolvedores, simplicidade dos algoritmos, volume e redução da probabilidade de bugs. A tabela exibida ao lado do gráfico, faz uma comparação da pontuação entre o produto de software avaliado e uma média representativa dos projetos recentes de PHP [PHPMetrics].

4.2. Qafoo

Os resultados da ferramenta Qafoo (*Mess Detect Violations*), foram agrupados por projeto (versão) e estão representados na figura a seguir:

MESS DETECT	VERSÕES				
	salonERP1,0	salonERP1,1	salonERP2,0,0	salonERP2,1,0	salonERP3,0,0
BooleanArgumentFlag	167	167	167	167	324
ElseExpression	802	802	802	802	>1000
StaticAccess	573	568	568	568	>1000
CyclomaticComplexity	138	138	140	140	377
NpathComplexity	106	106	108	108	246
ExcessiveMethodLength	69	67	67	67	118
ExcessiveClassLength	7	7	7	7	29
ExcessiveParameterList	19	19	19	19	19
ExcessivePublicCount	2	2	2	2	23
TooManyFields	3	3	3	3	11
TooManyMethods	4	4	4	4	31
TooManyPublicMethods	4	4	4	4	79
ExcessiveClassComplexity	10	10	11	11	86
Superglobals	0	0	0	0	0
CamelCaseClassName	0	0	0	0	0
CamelCasePropertyName	0	0	0	0	0
CamelCaseMethodName	0	0	0	0	0
CamelCaseParameterName	0	0	0	0	0
CamelCaseVariableName	0	0	0	0	0
ExitExpression	5	5	5	5	6
EvalExpression	0	0	0	0	12
GotoStatement	0	0	0	0	0
NumberOfChildren	0	0	0	0	6
DepthOfInheritance	0	0	0	0	5
CouplingBetweenObjects	2	2	2	2	41
DevelopmentCodeFragment	1	1	1	1	1
ShortVariable	0	0	0	0	0
LongVariable	0	0	0	0	0
ShortMethodName	0	0	0	0	0
ConstructorWithNameAsEnclosingClass	0	0	0	0	0
ConstantNamingConventions	0	0	0	0	0
BooleanGetMethodName	0	0	0	0	0
UnusedPrivateField	0	0	0	0	0
UnusedLocalVariable	0	0	0	0	0
UnusedPrivateMethod	0	0	0	0	0
UnusedFormalParameter	0	0	0	0	0
TOTAL →	1912	1905	1910	1910	3887

Figura 18: Resultados do experimento *Mess Detect Violations.* Agrupados por projeto

Para conseguir agrupar por projeto, foi necessário, na tela dos resultados (Figura 9), expandir o conteúdo de todos os arquivos e, pelo browser, procurar regra por regra (Ctrl F) presentes na Tabela 2. O browser exibia a quantidade localizada de cada regra, porém, como pode ser visto na versão salonERP3.0.0, nas regras *ElseExpression* e *StaticAccess,* a quantidade máxima que o browser "conta" é 1000, mostrando o resultado como "Mais de 1000 ocorrências".

Como no experimento anterior, as 4 versões anteriores a 3.0.0, apresentaram resultados praticamente iguais. Nota-se no entanto, um grande aumento nos números de problemas encontrados na versão 3.0.0 com a análise da ferramenta Qafoo. Apesar da versão 3.0.0 ter tido uma diminuição significativa de código, o número de problemas encontrado (3887) foi praticamente o dobro das demais versões (1912).

Todas as versões apresentavam os principais erros nas categorias *"Clean Code Rules"* e *"Code Size Rules".* As regras que obtiveram o maior numero de erros encontrados, também em todas as versões, foram *BooleanArgument, ElseExpression, StaticAccess, CyclomaticComplexity, NpathComplexity* e *ExcessiveMethodLength.* Na versão 3.0.0, além do número total de erros ser praticamente o dobro das demais versões, algumas regras onde não existiam erros nas versões anteriores, passaram a apresentar erros, como nos exemplos *EvalExpression, NumberOfChildren* e *DepthOfInheritance.*

5. Análise Qualitativa

5.1. PHPMetrics

A análise qualitativa deve ser feita pelos arquivos de mudança de versão, porém, o software apresentado não possuía esses arquivos para análise. No entanto, o que pode ser observado com os experimentos realizados com a ferramenta PHPMetrics foi que, as 4 versões anteriores a versão 3.0.0, praticamente não sofreram nenhuma alteração relativa as métricas exibidas nos resultados. Isso, muito provavelmente, significa que não houve grandes mudanças de uma versão para outra e também não houve muitas adições de novas funcionalidades, e, as principais alterações que provavelmente foram ajustadas nessas versões eram referentes a correção de bugs.

Na versão 3.0.0, o que pode ser percebido é que houve uma significativa redução de código, porém, os resultados mostram, de acordo com as métricas, que houve regressão de código em relação as versões anteriores. A redução do código deve significar que funcionalidades foram removidas do software, e como em todas as versões, também foram corrigidos bugs. Mas, a diferença entre as 4 versões anteriores com a versão 3.0.0, indica que os "donos" do projeto não realizam a medição das métricas em busca de melhorar o código a cada versão.

5.2. Qafoo

Como descrito na seção anterior, não foi possível realizar a análise qualitativa das versões do software pelos arquivos de mudança de versão, visto que o projeto não os possuía. A ferramenta Qafoo foi utilizada para encontrar problemas de código e, todas as versões, principalmente a versão 3.0.0, mostraram um grande número de problemas encontrados. As versões anteriores a versão 3.0.0, praticamente não sofreram nenhuma alteração quanto ao numero total de erros encontrados, enquanto na versão 3.0.0, o número de erros praticamente dobrou.

Os resultados do experimento realizado no Qafoo, mostram, assim como na seção anterior, que as versões anteriores a 3.0.0 não devem ter sofrido grandes mudanças e adição de muitas funcionalidades entre uma versão e outra, e sim, a principal diferença entre elas está na correção de bugs. A versão 3.0.0 aparenta ter sofrido redução de funcionalidades e correção de bugs em relação as demais, porém, também é possível perceber que o os desenvolvedores da ferramenta não utilizam nenhuma análise de possíveis problemas de código (PMD), devido a última versão apresentar o dobro de erros, mesmo com a diminuição do código.

6. Conclusões

As fermentas apresentadas no trabalho que realizam as análises de códigos, se mostraram muito útil na avaliação da qualidade interna do software, mostrando que é possível verificar a evolução de um sistema ou software, a medida que novas versões são lançadas. O objetivo deste estudo foi realizar a medição das últimas versões do software salonERP e analisar as diferenças encontradas entre as versões. As medições foram realizadas por métricas contidas na ferramenta PHPMetrics e por regras de "Mess Detection Violations", originários da ferramenta PHPMD, porém, presentes nas análises realizadas com a ferramenta Qafoo.

As versões analisadas do software, praticamente não apresentaram alterações entre as métricas e as regras de PMD, exceto na versão 3.0.0, onde houve redução de código, mas, apesar disso, o código obteve piores resultados que os demais. Com isso, conclui-se que os desenvolvedores não utilizam as medições presentes no trabalho ao disponibilizar uma nova release, indicando que o código deve passar por melhorias internas na qualidade.

Referências

RIBEIRO, W.; RIBEIRO, D. D. C.; PLASTINO, A.; MURTA, L. G. P.. **Acompanhamento da Evolução de Software via Métricas**. In: Workshop de Manutenção de Software Moderna, 2012.

ABÍLIO, R.S. **Relatório Técnico:** Análise da evolução de três sistemas de software com uso de medidas. Disciplina de Tópicos Em Computação I, Gestão da Qualidade de Software, UFLA, 2013.

PHPMetrics, **Documentation**. Disponível em < http://www.phpmetrics.org/documentation/index.html> Acessado em: 28 de novembro de 2018.

QAFOO/QUALITY ANALYZER, **GitHub**, Disponível em < https://github.com/Qafoo/QualityAnalyzer> Acessado em: 28 de novembro de 2018.