ANÁLISE DE MÉTRICAS PARA AVALIAÇÃO DE QUALIDADE DE SOFTWARE COM O USO DE PADRÕES DE CÓDIGO

Valdiriano do Nascimento Pereira (valdiriano@gmail.com)

Érika Höhn (erika.hohn@gmail.com)

Resumo

Este trabalho visa a analisar métricas de código para gerenciamento da qualidade de software, no contexto de uso de padrões de código no desenvolvimento de um sistema. Essa análise será feita comparando métricas de código utilizando dois frameworks diferentes, antes e depois de aplicar padrões de código, apresentando as diferenças entre os indicadores extraídos, e como tais métricas podem demonstrar quantitativamente o nível de qualidade de software.

Palavras-chave: Qualidade de Software. Métricas. Manutenibilidade. Padrões de Código.

Introdução

De modo geral, a definição de qualidade é imprecisa e varia de acordo com o ponto de vista e a finalidade do objeto a ser avaliado. Em se tratando de software, a qualidade pode ser caracterizada pela falta de erros ou defeitos encontrados, ou se os requisitos previamente definidos são devidamente atendidos; porém, a qualidade de um sistema pode ser medida em várias dimensões diferentes.

Duas das preocupações que existem no gerenciamento de qualidade de softwares são estabelecer frameworks e padrões que elevem o nível de qualidade do sistema, e definir as metas a serem atingidas. Desta forma, juntamente com os processos de desenvolvimento, devem ser definidos e escolhidos os padrões que serão utilizados. Entre esses padrões, estão alguns já existentes e consolidados, regulamentados pela ISO ou IEEE, por exemplo, ou padrões próprios internos da equipe.

Em muitos casos, garantir qualidade a uma aplicação significa somente definir tais processos, procedimentos e padrões; em outros, a garantia da qualidade inclui também gestão de configuração, verificação e validação, mesmo após a entrega do produto. Assim sendo, o código fonte do software deve seguir os padrões pré-

definidos e escolhidos pela equipe, de forma que tenha características que auxiliem na compreensão e manutenção do software pela equipe.

Para que a qualidade do processo de desenvolvimento e de manutenção seja medida, assim como a qualidade do próprio sistema em si, é necessário fazer tarefas sistemáticas, auditorias, com a utilização de métricas. Com o levantamento e o acompanhamento de tais métricas, é possível fazer análises quantitativas do processo ou do produto, podendo descobrir e analisar quais as causas das análises realizadas, e permitindo que a equipe tome ações que possam melhorar os resultados obtidos.

Sendo assim, o objetivo deste trabalho é analisar métricas de código e apresentar as diferenças nos resultados de métricas de software seguindo padrões e softwares sem tais padrões. Para realizar essa análise, foram extraídas métricas de código em um projeto utilizando dois frameworks diferentes, sem utilizar padrões de código, e então o código foi refatorado para se adequar aos padrões, sendo medido novamente.

1. Referencial teórico

1.1. Qualidade de Software

De acordo com Pressman (2011), qualidade de software pode ser definida como “uma gestão de qualidade efetiva aplicada de modo a criar um produto útil que forneça valor mensurável para aqueles que o produzem e para aqueles que o utilizam”. Essa definição pode ainda ser mais detalhada e alterada de acordo com várias características a serem medidas em um produto de software. Dessa forma, a qualidade de software acaba sendo uma combinação de fatores que dependem diretamente de determinadas aplicações, dos clientes que as solicitam e da equipe que desenvolve o projeto.

Poder avaliar e garantir a qualidade de um software é um ponto essencial para os *stakeholders*; além de agregar valor ao produto final, entregando ao cliente um produto eficiente, também reduz custos e riscos (PRESSMAN, 2011). A qualidade de software envolve não só a qualidade do produto em si, mas também a qualidade dos processos de desenvolvimento e de manutenção utilizados (VALE, 2013).

Quando se trata de um software, a qualidade deve ser medida de acordo com requisitos que o usuário pode encontrar, como corretitude, confiabilidade, usabilida-

de etc. Também pode ser medida de acordo com fatores estruturais do sistema, como portabilidade, testabilidade e manutenibilidade (VAN VLIET, 2009).

A ISO/IEC 25010 é uma norma internacional que entrou em vigor em 2011, substituindo a norma ISSO/IEC 9126. Esta norma define um modelo de qualidade de produto de software.

De acordo com a ISO/IEC 25010, que regulamenta a qualidade de produto de software, existem oito características principais a serem avaliadas em um software, ao avaliar sua qualidade. Cada uma dessas características possui sub-características, para refinamento e melhor avaliação.

As características definidas pela ISO/IEC 25010 são (ISO... 2017):

- Adequação funcional;
- Desempenho;
- Compatibilidade;
- Usabilidade;
- Confiabilidade;
- Segurança;
- Manutenibilidade;
- Portabilidade.

Assim como a ISO 25010 define oito características para avaliação da qualidade de software, há definições de fatores de qualidade definidos por Jim McCall, em 1977, e também por David Garvin, em 1987; porém, a equipe responsável pode definir quais fatores serão levados em consideração para prosseguir com a avaliação da qualidade de cada software (PRESSMAN, 2011).

1.2. Métricas de software

Medições de software são úteis para fazer avaliações sobre a qualidade do sistema. Usando medições, métricas podem ser extraídas de um software, e a partir delas, avaliações podem ser realizadas (SOMMERVILLE, 2011). Elas podem ser utilizadas tanto para atribuir algum valor aos atributos de qualidade de um sistema, quanto para identificar os atributos do sistema que não atingiram os padrões de qualidade (SOMMERVILLE, 2011).

Em Engenharia de Software, Pressman (2011) define medida como “uma indicação quantitativa da extensão, quantidade, capacidade ou tamanho de algum atributo de um produto”, e métrica como “uma medida quantitativa do grau com qual um sistema, componente ou processo possui determinado atributo”. Ou seja, uma métrica é a relação entre as medições e os atributos do produto, sendo possível gerar resultados (OLIVEIRA, 2010). É possível realizar medições de várias formas, para que seja possível obter resultados relevantes para analisar o processo do desenvolvimento e a qualidade do software (FERREIRA, 2012).

Sommerville (2011) divide as métricas em duas categorias: métricas de controle, que auxiliam no gerenciamento e nos processos utilizados no desenvolvimento, como esforço médio ou tempo de reparo; e métricas de previsão (ou métricas de produto), que são mais relacionadas ao próprio software, como complexidade ciclomática de um módulo do sistema.

1.2.1. Métricas de código-fonte

Métricas de código-fonte são quaisquer métricas que podem ser obtidas a partir do código-fonte e de qualquer representação gráfica que faça parte da especificação do sistema (OLIVEIRA FILHO, 2013).

Segundo Oliveira Filho (2013), as métricas de código-fonte podem ser divididas em métricas de tamanho e complexidade e métricas orientadas a objeto. As métricas de tamanho podem ser aplicadas a qualquer software, e as métricas orientadas a objeto só são aplicáveis a sistemas orientados a objeto.

1.3. Manutenção e manutenibilidade

Manutenção é definida por Vale (2013) como sendo qualquer tipo de mudança no sistema que ocorra após a entrega e o uso do sistema pelos usuários. Manutenção de software não está limitada apenas a correções de erros e defeitos; também pode envolver mudanças no ambiente ou nos requisitos do sistema (VAN VLIET, 2008).

Segundo Sommerville (2011), existem três tipos de manutenção de software:

- Correção de defeitos;
- Adaptação do ambiente;
- Adição de funcionalidades.

Pesquisas em geral apontam que a manutenção de software requer um custo maior do departamento de TI do que o próprio desenvolvimento: dois terços do orçamento são destinados à manutenção e um terço apenas para o desenvolvimento (SOMMERVILLE, 2011).

Manutenibilidade é a capacidade de um sistema poder ser modificado; o quanto ele facilita as atividades de manutenção e a qualidade de uso de quem está efetuando as modificações (VALE, 2013).

Van Vliet (2008) aponta que os principais problemas que levam um sistema à manutenção são:

- Código mal estruturado ou sistemas mal codificados;
- Falta de conhecimento do sistema por parte dos programadores envolvidos;
- Documentação ausente, desatualizada ou insuficiente.

1.4. Padrões de código

Utilizar padrões e boas práticas de código e design é uma técnica muito útil para desenvolver projetos manuteníveis e reutilizáveis, além de diminuir o tempo e o custo do desenvolvimento do software (QUAN; ZONGYAN; LIU, 2008).

Van Vliet (2008) diz, também, que é importante saber identificar quando é necessário realizar a refatoração de um sistema, devido à sua constante evolução e mudança. Uma das características que indica que o sistema está degradado está diretamente envolvida com o código-fonte: o uso de diferentes padrões ao longo do sistema.

Existem alguns modelos de boas práticas diferentes, mas de modo geral todos focam em aumentar a legibilidade do código, centralizar a lógica para que a lógica não fique distribuída nas classes existentes, e deixar a escrita do código fluente, melhorando sua indentação, tamanho das linhas de código, espaços em branco, entre outros pontos que facilitam a leitura (FERREIRA, 2012).

1.4.1. PSR

PSRs (*PHP Standards Recommendations*) são guias de estilo para PHP criados pelo PHP-FIG (*Framework Interoperability Group*), um grupo cujo ideal é manter a comunicação entre a comunidade de desenvolvedores PHP, de forma que todos possam conversar e encontrar pontos em comum. O FIG é composto por grandes

representantes da comunidade PHP, responsáveis por projetos bastante conhecidos e utilizados, como o CakePHP, Doctrine, Drupal, entre outros.

Cada recomendação (PSR) aborda um assunto, desde o básico, até assuntos mais avançados e específicos, como cache, links, *autoloading* etc. Existem atualmente 8 PSRs aprovadas, enquanto outras ainda estão em fase de rascunho ou revisão.

As PSRs que abordam recomendações referentes a código são as PSR-0, PSR-1, PSR-2 e PSR-4, e já são utilizadas por grandes projetos utilizando PHP, como Drupal, Zend, Symfony, Laravel, CakePHP, phpBB, AWS SDK, FuelPHP, Lithium etc (PHP, 2017).

2. Metodologia

Para fazer a análise de métricas de código e poder extrair dados iniciais, serão utilizadas duas versões de um mesmo projeto em PHP, uma delas utilizando o framework CakePHP, versão 3.4, e a outra utilizando o framework CodeIgniter, versão 3.1. As duas versões do projeto serão equivalentes na finalidade e nas funcionalidades, para que a comparação dos resultados seja feita de forma justa.

Esses dois projetos a princípio não estarão seguindo as recomendações PSR, e algumas métricas de código serão extraídas dos projetos neste estado. Então, os dois projetos serão ajustados para se adequarem às PSR, e, após isso, os dois projetos serão avaliados novamente e métricas pós-ajuste serão extraídas. Desta forma, os projetos podem ser avaliados antes, durante e depois dos ajustes necessários para estarem seguindo os padrões.

O CakePHP, por si só, já exige que alguns padrões próprios do framework sejam utilizados para que haja o correto funcionamento do código, mas essa padronização se limita à nomenclatura de métodos e classes. Ademais, a escrita do código é livre, de acordo com o desenvolvedor.

Por outro lado, o CodeIgniter não necessita que haja uma padronização de nenhuma forma, desde que a estrutura de pastas e a hierarquia de classes sejam respeitadas.

Esses dois itens foram decisivos para a escolha destes dois projetos e frameworks, de forma que a análise a que este trabalho se propõe consiga analisar pelo menos dois cenários diferentes.

Várias ferramentas de análise de projetos e levantamento de métricas foram encontradas, entre elas, a Kalibro, Analizo, e outras, porém somente duas serão utilizadas para efeito de análise: PhpMetrics e PHP_CodeSniffer. As duas foram selecionadas por tratarem especificamente de projetos em PHP, pela quantidade de métricas abordadas, forma de apresentação dos resultados e por estarem atualizadas.

2.1. Ambiente de testes

Para a realização dos testes propostos, foram utilizadas as seguintes ferramentas:

- Servidor Apache;
- PHP 7.1;
- IDE PHP Storm;
- WakaTime;
- PHP_CodeSniffer;
- PhpMetrics.

2.2. Estudo de caso

O projeto utilizado será uma loja, cujas classes estão representadas no diagrama abaixo.

Abaixo segue diagrama de classes da loja.

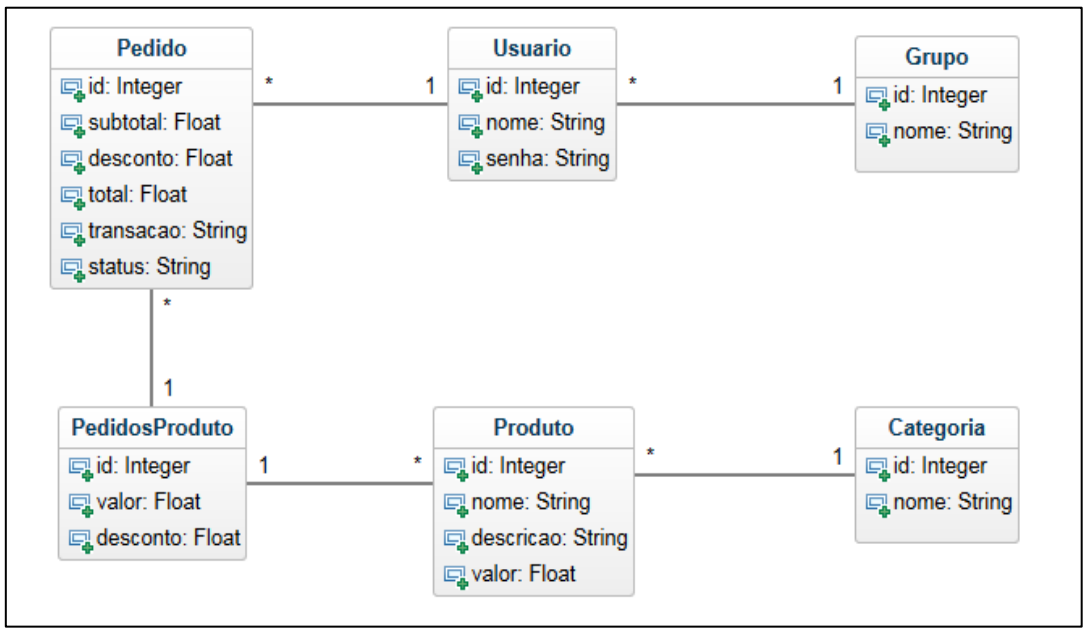

Figura 1 Diagrama de classes do projeto utilizado

2.3. Métricas extraídas e padrões utilizados

Para prosseguir com a análise dos resultados apresentados, foram extraídas métricas de código utilizando PHP_CodeSniffer e PhpMetrics.

A ferramenta PHP_CodeSniffer é utilizada para detectar variações de algum padrão de código definido; neste caso, estão sendo utilizadas as recomendações PSR-1 e PSR-2 como referência. A PhpMetrics, por outro lado, faz o levantamento de várias outras métricas de projetos e classes em PHP.

Também foi utilizado o WakaTime, integrado à IDE PhpStorm, para registrar o tempo utilizado para fazer as devidas correções.

3. Resultados obtidos

Os resultados obtidos, antes e depois da aplicação das recomendações de padrões de código PSR-1 e PSR-2, são apresentados a seguir.

3.1. Projeto utilizando CakePHP

Ao realizar as medições no projeto sem padrões de código, foram obtidos os seguintes resultados:

Quantidade de erros	259
Quantidade de avisos	1
Média de bugs por classe (Halstead)	0,12
Média de defeitos por classe (Kan)	0,39
Complexidade ciclomática média por classe	4,4
LOC	456

Tabela 1 Métricas do projeto em CakePHP antes da aplicação dos padrões de código

Após a aplicação dos padrões de código, foram encontradas as seguintes métricas:

Tempo para aplicar correções	30 minutos e 22 segundos
Quantidade de erros	0
Quantidade de avisos	0
Média de bugs por classe (Halstead)	0,12
Média de defeitos por classe (Kan)	0,39
Complexidade ciclomática média por classe	4,4
LOC	449

Tabela 2 Métricas do projeto em CakePHP após a aplicação dos padrões de código

3.2. Projeto utilizando CodeIgniter

No projeto utilizando CodeIgniter, sem padrões de código, foram obtidos os seguintes resultados:

Quantidade de erros	209
Quantidade de avisos	6
Média de bugs por classe (Halstead)	0,08
Média de defeitos por classe (Kan)	0,43
Complexidade ciclomática média por classe	5,14
LOC	323

Tabela 3 Métricas do projeto em CodeIgniter após a aplicação dos padrões de código

Após a aplicação das PSRs, os resultados abaixo foram alcançados:

Tempo para aplicar correções	34 minutos e 31 segundos
Quantidade de erros	0
Quantidade de avisos	0
Média de bugs por classe (Halstead)	0,06
Média de defeitos por classe (Kan)	0,43
Complexidade ciclomática média por classe	5,14
LOC	449

Tabela 4 Métricas do projeto em CodeIgniter após a aplicação dos padrões de código

4. Conclusão

A utilização de padrões de código, sejam as PSRs utilizadas nos testes ou qualquer outro que a equipe tenha preferência, facilita o entendimento posterior da equipe durante a manutenção do sistema, aumenta a legibilidade do código e aumenta a qualidade do software.

Devido ao tamanho do software utilizado para os testes, e também ao fato de as mudanças de código se aplicarem somente a padrões de código e não a refatoração do código para ajuste de lógica e de métodos, poucas métricas de código-fonte mudaram antes e depois da aplicação de padrões.

Contudo, os testes realizados conseguem mostrar que aplicar tais padrões em um sistema já pronto acaba necessitando de tempo, gerando um custo necessário para atingir um nível de qualidade melhor; começar os projetos seguindo algum tipo de padrão de código acaba sendo a melhor opção.

Referências

CAKE SOFTWARE FOUNDATION INC. **CakePHP:** Cookbook. Disponível em: <https://book.cakephp.org/2.0/pt/getting-started/cakephp-conventions.html>. Acesso em: 31 mar. 2017.

FERREIRA, Isaias Alves. **Análise do impacto da aplicação de padrões de projeto na manutenibilidade de um sistema orientado a objetos.** 2012. 78 f. TCC (Graduação) - Curso de Sistemas de Informação, Universidade Federal de Lavras, Lavras, 2012.

ISO 25000. Disponível em: <http://iso25000.com/index.php/en/iso-25000-standards/iso-25010>. Acesso em: 22 abr. 2017.

MEIRELLES, Paulo Roberto Miranda. **Levantamento de Métricas de Avaliação de Projetos de Software Livre**. São Paulo, 2008.

OLIVEIRA, Jaqueline Faria de. **Métricas para avaliação do grau de quantificação de sistemas orientados por aspectos.** 2010. 98 f. Dissertação (Mestrado) - Curso de Informática, Pontifícia Universidade Católica de Minas Gerais, Belo Horizonte, 2010.

OLIVEIRA FILHO, Carlos Morais de. **Kalibro: interpretação de métricas de código-fonte.** 2013. 89 f. Dissertação (Mestrado) - Curso de Ciência da Computação, Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2013. Disponível em: <http://www.teses.usp.br/teses/disponiveis/45/45134/tde-25092013-142158/en.php>. Acesso em: 05 de maio de 2017.

PHP-FIG. Disponível em: <www.php-fig.org/>. Acesso em: 30 mar. 2017.

PHP: The Right Way. The Right Way. Disponível em: <http://www.phptherightway.com/#code_style_guide>. Acesso em: 16 abr. 2017.

PRESSMAN, Roger S.. **Engenharia de Software:** Uma abordagem profissional. 7. ed. Porto Alegre: Amgh, 2011.

QUAN, Long; ZONGYAN, Qiu; LIU, Zhiming. Formal Use of Design Patterns and Refactoring. **Communications In Computer And Information Science,** [s.l.], p.323-338, 2008. Springer Berlin Heidelberg. http://dx.doi.org/10.1007/978-3-540-88479-8_23.

SOMMERVILLE, Ian. **Engenharia de Software.** 9. ed. São Paulo: Pearson Prentice Hall, 2011.

VALE, Gustavo Andrade do. **Avaliação da manutenibilidade de sistemas de software derivados de linhas de produtos de software.** 2013. 111 f. TCC (Graduação) - Curso de Sistemas de Informação, Ciência da Computação, Universidade Federal de Lavras, Lavras, 2013.

VAN VLIET, Hans. **Software engineering:** principles and practice. 3. ed. Chichester: John Wiley & Sons, 2008.