Aplicando métricas de código em uma fábrica de *software* acadêmica - Qualidade no Desenvolvimento

Matheus Vieira Gonçalves

Gabriel José de Melo

Centro Universitário de Anápolis – Unievangélica

Caixa Postal 122 e 901 – 75.083-515 – Anápolis – GO – Brasil

matheusvieira1900@hotmail.com, gabriel.josemelo@hotmail.com

***Abstract**: In recent years constant technological innovations have changed the labor market many times over. Functions that were previously dependent on human labor were now automated by computers. Given that a computer is nothing more than a machine that performs functions that have been programmed for it, it is of great importance to study the correct way to program such functions. To this end, the current work addresses code metrics, which are responsible for improving the way a computer / software is programmed.*

***Keywords:** Software engineering, code metrics, code quality.*

***Resumo:** Nos últimos anos as constantes inovações tecnológicas modificaram inúmeras vezes o mercado de trabalho. Funções que antes era dependentes da mão de obra humana passaram a ser automatizadas pelos computadores. Tendo em vista que um computador nada mais é do que uma máquina que realiza funções que lhes foram programadas, torna-se de grande importância estudar a forma corretar de programar tais funções. Com esse objetivo o atual trabalho abordar as métricas de código, que são responsáveis por melhorar a forma como um computador/software é programado.*

***Palavras-chave:** Engenharia de software, métricas de código, qualidade de código.*

1.Introdução

Com o passar dos anos a tecnologia passou por diversas transformações onde os computadores ganharam poder e velocidade de forma espantosa (FONSECA, 2007). Dessa forma, computadores que antes necessitavam de grandes espaços, incontáveis válvulas e diversos cabos passaram por uma transformação que reduziu consideravelmente o número e o tamanho de peças necessários para seu funcionamento. Um computador que no passado ocupava uma sala por inteira atualmente pode ser carregado em uma mochila, em um bolso de uma calça ou até mesmo na palma da mão (MOORE, 2007).

Tendo em vista que um computador é uma máquina que manipula dados a partir de uma lista de instruções (FERRARI, 2008), torna-se de fácil percepção que a evolução da tecnologia não está limitada somente aos hardwares (peças que compõem um computador), ela também engloba os *softwares* os quais representam a parte lógica cuja função é fornecer instruções

citadas anteriormente, para que o hardware possa operar. Dessa forma a evolução tecnológica também está ligada ao crescimento das linguagens de programação (FONSECA, 2007).

Com a evolução das linguagens de programação os softwares acabaram por impactar gradativamente o mercado de trabalho com o passar das décadas. Em destaque, os *softwares* livres (softwares que permitem que o usuário tenha livre acesso ao seu código-fonte) acabaram por ocupar um espaço significativamente grande dentro da economia mundial (BENKLER, 2006), impactando não somente as grandes empresas como também as pequenas empresas ou *startups*.

2. Referencial teórico

O desenvolvimento de *software* consiste basicamente da escrita de código, em forma de texto, que será interpretado por um computador e criará um algoritmo na qual executará as funções descritas no código. Contudo, as máquinas não interpretam esse texto que foi escrito, elas interpretam o código de máquina que será gerado a partir desse código em texto. A ideia mais lógica seria escrever esse código direto na forma em que a máquina fosse entender, porém fazer dessa forma deixa o conteúdo muito ruim para uma pessoa trabalhar.

Assim vieram as linguagens de programação com o intuito de ser de mais fácil compreensão para nós que escrevemos o código, com uma forma de programar utilizando de letras, números e símbolos comuns ao nosso dia a dia. Facilitando bastante o entendimento por parte de quem está escrevendo e lendo o código. É a partir das linguagens de programação que nós criamos algoritmos. Um algoritmo pode ser definido como uma sequência finita de passos (instruções) para resolver um determinado problema. Sempre que desenvolvemos um algoritmo estamos estabelecendo um padrão de comportamento que deverá ser seguido (uma norma de execução de ações) para alcançar o resultado de um problema (FERRARI, 2008).

Contudo, como um algoritmo ainda não é como um texto corrido, ele possui uma sintaxe e semântica próprias de cada linguagem, demandando estudo e aprofundando nessa linguagem para que ela seja entendida e seja criado algo conciso a partir dela. Além desse detalhe há o fato de que a estrutura do código, a lógica de sua execução e até detalhes simples como a forma que foi escrito dependem bastante de quem está escrevendo.

Um algoritmo mal escrito e/ou mal estruturado pode causar problemas tanto de entendimento, da própria pessoa e de outras também, como pode causar futuros problemas de execução do programa em desenvolvimento. Deixando assim o código pouco manutenível e reutilizável, prejudicando também outra pessoa caso seja necessário ela mexer no código legado que você deixou.

Visando resolver esse tipo de problema é necessário a utilização de métricas para avaliar e manter um bom código, que possa ser entendido bem pelo próprio criador e principalmente por outras pessoas, que possa ser reutilizável e com alta manutenibilidade por parte de qualquer um que precisar mexer nele. Isso pode ser visto como qualidade de código.

Métricas são utilizadas para estimar um cronograma e custos de desenvolvimento do software e medir a produtividade e qualidade do produto (MEIRELLES, 2013). A utilização desse método pode contribuir muito para manter a qualidade de vários processos dentro de uma empresa, um deles é o desenvolvimento do código de um *software*. Se há qualidade no

desenvolver de um código isto poderá prover qualidade as funcionalidades que aquele código exerce.

Nesse mercado, existem diversas linguagens de programação, que seguem diferentes paradigmas. Um desses paradigmas é a Orientação a Objetos, que atualmente é o mais difundido entre todos. Isso acontece porque se trata de um padrão que tem evoluído muito, principalmente em questões voltadas para segurança e reaproveitamento de código, o que é muito importante no desenvolvimento de qualquer aplicação moderna (GASPAROTTO, 2014). Além de métricas para medir a qualidade são necessários alguns métodos e/ou padrões pré-definidos que podem promover um código bem feito e estruturado. Alguns desses métodos ou padrões, podem tanto ser conceitos simples e inicias, como os referentes a orientação a objetos, e alguns mais complexos, por exemplo o *SOLID* ou alguns padrões de projeto.

É necessário frisar que quase sempre os conceitos iniciais são tão importantes ou até mais que os avançados, pois é a partir deles que vieram os conceitos mais aprofundados. Como por exemplo, os conceitos do *SOLID* e dos padrões de projeto se baseiam bastante na orientação a objetos. Sem o conhecimento básico de orientação a objetos, a pessoa terá dificuldades para entender e/ou aplicar os conceitos mais avançados.

Seguir boas práticas de desenvolvimento de software é ótima forma de garantir que seja escrito um código com qualidade. O mais recomendável é definir com sua equipe qual será o formato de organização utilizado, podendo ser tanto um específico da equipe, como um já conceituado no mercado por autores da área. Além disso é necessário o uso das métricas para verificar e garantir que esse formato está realmente sendo útil e/ou necessário, medindo dessa forma a eficiência do código e sua qualidade, assim como afirma os autores Meirelles e Júnior:

“A avaliação de qualidade de código fonte no início do desenvolvimento pode ser de grande valia para auxiliar equipes inexperientes durante as etapas seguintes do desenvolvimento de software” (MEIRELLES, 2013).

> Métricas de código fonte caracterizam bem o produto de software em qualquer estado do seu desenvolvimento. Existem vários tipos de métricas de código fonte. Entre as principais categorias, temos métricas de tamanho, complexidade, acoplamento e coesão (Júnior, 2015).

De acordo Júnior (2015) dentre essas métricas temos por exemplo:

- *Lines of Code (LOC)* - Linhas de código;
- *Number of Methods (NOM)* - Número de métodos de uma classe;
- *Average Methods Lines of Code (AMLOC)* - Média de linhas de código por método;
- *Average Cyclomatic Complexity per Method (ACCM)* - Média de complexidade ciclomática por método;
- *Response For a Class (RFC)* - Resposta para uma classe;
- *Depth in Inheritance Tree (DIT)* - Profundidade na árvore de herança;
- *Number Of Children (NOC)* - Número de subclasses;
- *Lack of Cohesion in Methods (LCOM)* - Falta de coesão em métodos;
- *Afferent Connections per Class (ACC)* - Conexões aferentes de uma classe;
- *Coupling Factor (COF)* - Fator de acoplamento.

3. Método de pesquisa

Após uma análise dos principais problemas relatados por uma equipe de desenvolvimento onde seus membros são inexperientes e iniciantes no ramo da programação, foi possível constatar que mais de 50% dos problemas relatados envolviam qualidade de código que era produzida pela equipe.

Com o intuito de corrigir e/ou amenizar tais problemas foi proposta uma pesquisa que visasse aumentar a qualidade do código produzida através de ferramentas e práticas que auxiliassem a medir, avaliar e estimar os códigos produzidos, possibilitando assim identificar postos a serem tradados e padrões a serem corrigidos.

Para atingir tal objetivo, o presente trabalho tem por objetivo estudar as métricas de código, tema o qual já foi introduzido anteriormente, afim de analisar quais das métricas de código existente melhor atenderia ao objetivo de solucionar ou ao menos amenizar os principais problemas relacionados a qualidade de código existente não apenas nessa equipe mas em qualquer equipe de desenvolvimento iniciante/inexperiente.

4. Abordagem Proposta / Estudo na FTT

Durante o decorrer deste tópico serão listadas as principais métricas de código existente, especificando sua funcionalidade e seus benefícios. Posteriormente será elaborado e abordado um plano de utilização das principais métricas em conjunto, afim de desenvolver uma solução que através da junção de várias abordagens, possa atingir um único objetivo final: a melhoria na qualidade de código.

4.1. Métricas de códigos

4.1.1 *Lines of Code (LOC) - Número de linhas de código*

Atualmente existem diversos tipos de formar de medir e avaliar o tamanho ou a quantidade de um código, dentre essas técnicas existe a *Line of Code. A* importância do monitoramento do volume de um código se baseia na capacidade de realizar comparações entre sistemas semelhantes (RONALDO, 2015), visando analisar e identificar códigos extensos que poderiam ser reduzidos e otimizados.

Para a realização dessa técnica é valido lembrar que só são contadas as linhas executais, ou seja, linhas em branco e comentários são descartados na contagem (MEIRELLES, 2013). Além disso, para realizar uma comparação entre dois sistemas, é necessário que ambos tenham sido desenvolvidos na mesma linguagem de programação (JONES, 1991), tal fator é necessário devido a diversidades de recursos diferentes que cada linguagem oferece durante o processo de implementação das funcionalidades.

4.1.2 *Number of Methods (NOM)* - Número de métodos

É uma métrica que possui o objetivo de medir a complexidade de um método de uma classe através da quantidade de linhas de código que ele possui, visando dessa forma estipular

sua importância e seu impacto dentro das funcionalidades implementadas pela classe. É valido ressaltar que nem sempre quantidade resulta em qualidade. Quando um método possui muitas linhas é necessário realizar uma análise sobre o mesmo, buscando analisar possíveis melhorias.

4.1.3 *Average Methods Lines of Code (AMLOC)* - Média de linhas de código por método

AMLOC seria a junção das métricas LOC e NOM, tendo pro objetivo realizar uma análise sobre a média de linhas contidas na construção dos métodos do sistema, observando se o código está bem distribuído entre eles. Quanto maior a quantidade de linhas presente nos métodos, mais "pesados" eles se tornam. Dessa forma é recomendando ter muitas operações de fácil entendimento do que poucas operações com alta complexidade (MEIRELLES, 2013).

4.1.4 *Average Cyclomatic Complexity per Method (ACCM)* - Média de complexidade ciclomática por método

A ACCM é utilizada para medir a complexidade do programa, analisando número de caminhos independentes que um software pode seguir em sua execução (RONALDO, 2015). Na prática, assim como afirma o autor Ronaldo, cada bloco condicional presente dentro do sistema incrementa mais 1 ponto no nível de complexidade. No geral ela é calculada a nível de método, posteriormente é feita uma média entre os resultados de todos os métodos de uma classe, possibilitando assim determinar a sua complexidade.

Afim de facilitar visualização, os resultados obtidos através dessa métrica podem ser exibidas em formato de gráfico de controle, onde os nós irão representar os as instruções sequenciais e o os arcos indicarão o sentido do fluxo de controle entre várias instruções.

4.1.5 *Response For a Class (RFC)* - Resposta para uma classe

RFC é uma métrica que conta o número de métodos de uma determinada classe que podem ser invocados em resposta a uma mensagem enviada para ela através de um de seus objetos, assim como afirma os autores Chidamber e Kemerer, utilizados como referência por Ronaldo.

Uma classe que apresenta um alto índice de RFC provavelmente possuirá um grande número de métodos ou então será dependente de diversas classes, dessa forma podendo indicar baixa coesão e alto nível de acoplamento (RONALDO, 2015).

4.1.6 Depth in Inheritance Tree (DIT) - Profundidade na árvore de herança

DIT é uma métrica que mede a profundidade em que uma classe se encontra dentro da arvore de herança (RONALDO, 2015). Em outras palavras, a DIT mede qual a distância entre a classe atual e a o nó raiz na árvore de herança, ou seja, a quantidade de classes as quais ela está ligada pelo sistema de herança.

Se uma classe não hera de nenhuma outra, então possui DIT igual a 0, caso ela herde de alguma outra classe então a profundidade/ DIT é igual a 1 e assim sucessivamente, sempre aumentando o DIT conforme a quantidade de classes das quais ela herda.

Nessa métrica são analisadas somente as classes do sistema, dessa forma classes de bibliotecas não são contabilizadas (MEIRELLES, 2013).

4.1.7 *Number Of Children (NOC)* - Número de subclasses

NOC é uma métrica semelhante a DIT, seu objetivo é medir a quantidade de filhos que uma classe possui. O valor de DIT quando uma classe é criada, ou seja, quando nenhuma outra classe ainda herdou dela é 0. À medida que novas classes começam a herdar seus atributos e métodos o valor do DIT é aumentado em 1 para cada nova classe que a estender. É valido ressaltar que é necessário que a herança seja direta, ou seja, filhos de filhos não são contabilizados (RONALDO, 2015).

DIT e NOC são métricas bem semelhantes, mas possuem objetivos diferentes. Através do DIT é possível medir a complexidade que uma classe possui ao fazer herança recursiva, ou seja, herdar de várias outras classes. Quando o valor do DIT de uma classe é alto, significa que a mesma possui maiores chances de se tornar imprevisível, podendo prejudicar o programa durante sua execução, entretanto maiores valores de DIT também significam maior reuso de código, fator que melhora o desemprenho e produtividade durante o desenvolvimento.

NOC também é responsável por medir o reuso de código, além disso por medir a quantidade de filhos que herdam dessa classe torna possível determinar o nível de impacto que uma mudança pode ocasionar ao programa. Altos valores de NOC por exemplo podem significar que o código apresenta grandes taxas de reuso, entretanto também pode significar que uma pequena mudança na classe pode causar grandes impactos ao software, graças a quantidade de classes filhas que serão afetadas por herdarem dados dela.

4.1.8 Lack of Cohesion in Methods (LCOM) - Falta de coesão em métodos

A LCOM é uma métrica que possui como objetivo medir o nível de coesão que uma classe possui, para atingir tal objetivo existe diversas formas de calcular a coesão total de uma classe, uma delas é através da análise dos métodos dela verificando se eles estão coesos e utilizam os mesmos atributos dentro da classe. Caso seja constatado que um método não utiliza nada da classe, então provavelmente ele está inserido em um local errado (Ronaldo, 2015).

Essa métrica foi criada originalmente por Chidamber e Kemerer em 1994 possuindo diversas variações. Uma das mais atualizadas é a LCOM4, que fui utilizada anteriormente como exemplo, o qual analisa os métodos para verem se estão inseridos no local correto. Ela também apresenta alguns outros conceitos que aplicados sobre os métodos da classe, entretanto eles são serão abordados afim de não prolongar além do necessário o presente trabalho.

4.1.9 *Afferent Connections per Class (ACC)* - Conexões aferentes de uma classe

A ACC é uma métrica que tem como objetivo medir a conectividade de uma classe. Por exemplo, quando uma classe A utiliza métodos de uma classe B, significa então que existe uma certa conectividade entre essas duas classes. Tal fator significa também que se a classe A sofrer mudanças então a classe B pode ser alterada. Dessa forma, um alto valor de ACC pode significa alta taxa de reuso de código, entretanto é necessário que sejam tomados maiores cuidados durante as futuras manutenções, pois uma pequena alteração em uma classe pode impactar diversas outras que estejam implementando seus métodos.

4.1.10 *Coupling Factor (COF)* - Fator de acoplamento.

É uma métrica utilizada basicamente para medir o nível de acoplamento em um software. Em outras palavras essa métrica é utilizada para medir o grau de independência que os módulos possuem. “Um software fortemente conectado apresenta maior COF, indicando um baixo grau de independência entre módulos, alta complexidade e difícil entendimento e manutenção” (MEIRELLES, 2013).

Assim é recomendado que os módulos sejam estruturados de forma a torna os níveis de COF o mais baixo possível, evitando futuros problemas durante uma manutenção, tanto corretiva quanto evolutiva.

4.2. Plano de utilização das métricas

Tendo em vista a grande variedade de métricas existente, torna-se claro o ganho que pode ser gerado com a sua adoção na criação de softwares de qualidade, os quais apresentam um código limpo, eficiente e de fácil manutenibilidade.

Para uma equipe inexperiente no ramo da programação, a adoção de algumas das métricas apresentadas a cima seria de grande benefício, entre elas podemos destacar a LOC, AMLOC e a LCOM. Com a utilização da LOC e da AMLOC seria possível identificar códigos que ficaram muito extensos para a execução de alguma atividade, dessa forma algum plano de melhoria poderia ser utilizado, como por exemplo estudos afins do problema/funcionalidade a ser desenvolvido pelo código ou da própria sintaxe da linguagem. É valido ressaltar que códigos muito extensos podem impactar gradativamente no desempenho de um software, portanto identificá-los e combate-los é um dos primeiros passos a serem tomados por uma equipe de iniciantes.

O segundo maior problema associado a equipes inexperientes é a falta de coesão existente entre as classes e seus métodos. É bem comum no atual cenário de desenvolvimento nos depararmos por códigos de iniciantes, onde uma classe implementa métodos que não dizem respeito ao seu real objetivo, ou seja, códigos que estão implementados em um local inadequado. Tal fator pode causar inconsistência no código, problemas de manutenibilidade e principalmente duplicação de código, afetando o desempenho e a produtividade.

Com a utilização da métrica LCOM, poderia ser identifica blocos de códigos alocados em locais inapropriados, e a partir dessa identificação poderia ser realizado as devidas correções, evitando dessa forma os possíveis danos citados anteriormente.

As demais métricas de código também apresentam alto valor na otimização de desenvolvimento de um *software*, entretanto as que foram citadas nesse tópico são mais fáceis de serem executadas por pessoas inexperientes, além disso podem gerar um grande impacto positivo em um curto período de tempo.

5. Resultados esperados

Com a utilização das métricas de código os primeiros resultados esperados estão ligados diretamente a pequenas melhorias que impactam no desempenho do sistema e na produtividade da equipe, sendo elas códigos mais organizados e enxutos, redução dos códigos duplicados e melhor estruturação do mesmo.

Resultados mais complexos como uma arquitetura mais organizada ou até mesmo um código que previna problemas durante a manutenção, são obtidos através do tempo com um maior amadurecimento da equipe na utilização das métricas de código.

6. Considerações Finais

A implementação de métricas de código principalmente para equipes inexperientes pode ser uma tarefa árdua, graças a falta de conhecimento necessário para o desenvolvimento correto de um código de qualidade. Entretanto é válido ressaltar que o esforço necessário para a utilização de tais métricas é um fator que deve ser superado, isso pois, os benefícios que podem ser gerados pela utilização das métricas influenciam de forma positiva gradativamente o *software* conforme o seu desenvolvimento, aumentando a produtividade e evitando diversos problemas.

Um *software* que possui seu código guiado por métricas pode apresentar índices mais favoráveis ao sucesso do que um *software* não orientado a métricas. Tal fator ocorre, pois, uma das principais funções das métricas de qualidade é possibilitar que o *software* possa ter em um futuro, seja ele próximo ou distante, a possibilidade de sofrer manutenções, sejam elas corretivas ou evolutivas , e levando em consideração que o atual mercado torna-se cada dia mais exigente, é necessário que o *software* sempre esteja apto a aderir mudanças e melhorias.

Dessa forma é possível notar a importância da utilização correta das métricas de desenvolvimento de código na atualidade, as quais possuem fator fundamental na criação de *softwares* de sucesso.

7. Referencias

CHIDAMBER, S. R.; KEMERER, C. F. **A metrics suite for object oriented design. Software Engineering, IEEE Transactions on, IEEE**, v. 20, n. 6, p. 476–493, 1994.

FERRARI, F. **Introdução a algoritmos e programação,** Bagé - RS, 2008. Disponível em: https://lief.if.ufrgs.br/pub/linguagens/FFerrari-CCechinel-Introducao-a-algoritmos.pdf. Acesso em: 2 dez. 2019

FONSECA, C. F. **História da Computação: O caminho do pensamento e da tecnologia**. [s.n.], 2007. Disponível em: <http://www.pucrs.br/edipucrs/online/historiadacomputacao.pdf>. Acessado em 27 de novembro de 2019.

GASPAROTTO, H. M. **Os 4 pilares da Programação Orientada a Objetos.** [*S. l.*], 2014. Disponível em: https://www.devmedia.com.br/os-4-pilares-da-programacao-orientada-a-objetos/9264. Acesso em: 2 dez. 2019.

JONES T. C. **Applied Software Measurement: Assuring Productivily and Quality.** McGraw-Hill, New York. 1991.

JÚNIOR, M. R. P. **Estudo da correlação entre métricas de código fonte do sistema Android e seus aplicativos**, Brasília - DF, 2015. Disponível em: http://fga.unb.br/articles/0001/0528/marcos_ronaldo_090124511.pdf. Acesso em: 2 dez. 2019.

MEIRELLES, P. R. M. **Monitoramento de métricas de código-fonte em projetos de software livre**. São Paulo, 2013. Disponível em: < https://www.teses.usp.br/teses/disponiveis/45/45134/tde-27082013-090242/publico/tesePauloMeirelles.pdf>. Acessado em 27 de novembro de 2019.

MOORE, A. **Mobile as 7th of the mass media: an evolving history**. 2007. Disponível em: < http://siteresources.worldbank.org/EXTEDEVELOPMENT/Resources/TomiAhonenMobile7thMassMediaExcerpt.pdf?resourceurlname=TomiAhonenMobile7thMassMediaExcerpt.pdf>. Acessado em 02 de dezembro de 2019.