UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA
UNIDADE ACADÊMICA DE SISTEMAS E COMPUTAÇÃO
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

JOSÉ GLAUBER BRAZ DE OLIVEIRA

ESTUDO DE CASO:
ANÁLISE DO DESIGN DE CÓDIGOS DE ALUNOS INICIANTES EM PROGRAMAÇÃO ORIENTADA A OBJETOS

CAMPINA GRANDE - PB

2019

JOSÉ GLAUBER BRAZ DE OLIVEIRA

ESTUDO DE CASO:
ANÁLISE DO DESIGN DE CÓDIGOS DE ALUNOS INICIANTES EM PROGRAMAÇÃO ORIENTADA A OBJETOS

Trabalho de Conclusão Curso apresentado ao Curso Bacharelado em Ciência da Computação do Centro de Engenharia Elétrica e Informática da Universidade Federal de Campina Grande, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

Orientadora: Professora Dra. Eliane Cristina de Araújo.

CAMPINA GRANDE - PB

2019

O48e Oliveira, José Glauber Braz de.
Estudo de caso: análise do design de códigos de alunos iniciantes em Programação Orientada a Objetos. / José Glauber Braz de Oliveira. – 2019.

11 f.

Orientadora: Profa. Dra. Eliane Cristina de Araújo.
Trabalho de Conclusão de Curso - Artigo (Curso de Bacharelado em Ciência da Computação) - Universidade Federal de Campina Grande; Centro de Engenharia Elétrica e Informática.

1. Design de códigos. 2. Disciplina Programação Orientada a Objetos - UFCG. 3. Programação Orientada a Objetos. 4. Algoritmo k_Means. 5. Clusterização. 6. Qualidade de software - avaliação. I. Araújo, Eliane Cristina de. II. Título.

CDU:004(045)

Elaboração da Ficha Catalográfica:

Johnny Rodrigues Barbosa
Bibliotecário-Documentalista
CRB-15/626

JOSÉ GLAUBER BRAZ DE OLIVEIRA

ESTUDO DE CASO:
ANÁLISE DO DESIGN DE CÓDIGOS DE ALUNOS INICIANTES EM PROGRAMAÇÃO ORIENTADA A OBJETOS

Trabalho de Conclusão Curso apresentado ao Curso Bacharelado em Ciência da Computação do Centro de Engenharia Elétrica e Informática da Universidade Federal de Campina Grande, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

BANCA EXAMINADORA:

Professora Dra. Eliane Cristina de Araújo
Orientadora – UASC/CEEI/UFCG

Professor Dr. Cláudio de Souza Baptista
Examinador – UASC/CEEI/UFCG

Professor Dr. Tiago Lima Massoni
Disciplina TCC – UASC/CEEI/UFCG

Trabalho aprovado em: 25 de novembro de 2019.

CAMPINA GRANDE - PB

Estudo de caso: análise do design de códigos de alunos iniciantes em Programação Orientada a Objetos

Trabalho de Conclusão de Curso

José Glauber Braz de Oliveira
jose.oliveira@ccc.ufcg.edu.br
Universidade Federal de Campina Grande
Campina Grande, Paraíba, Brasil

Eliane Cristina de Araújo*
eliane@computacao.ufcg.edu.br
Universidade Federal de Campina Grande
Campina Grande, Paraíba, Brasil

RESUMO

A escolha do design é de fundamental importância no desenvolvimento de software de qualidade. Mesmo desenvolvedores experientes, costumam dedicar tempo considerável para modelar adequadamente um sistema. Diante disso, esse trabalho tem por objetivo fazer uma análise do design dos códigos desenvolvidos por alunos da disciplina de Laboratório de Programação 2, pois nem sempre o professor ou o próprio desenvolvedor tem um feedback para identificar os erros se escolher um determinado design.

Com objetivo de avaliar a escolha de design do aluno, foi realizada uma coleta de métricas que caracterizam o código desenvolvido. Logo em seguida, foi feita uma clusterização, utilizando o algoritmo k-Means, tendo como parâmetros de entrada as métricas extraídas. Com isso, foi possível verificar se existe um padrão entre os códigos que estão presentes no mesmo cluster e se existe uma correlação com a nota que o aluno alcançou. Foram encontrados três clusters, que agrupou os programas analisados nas proporções de 50%, 30% e 20%. Com a clusterização pudemos observar que os códigos com melhores valores de métricas tem correlação com notas altas obtidas, considerando o intervalo que foi pré-definido.

1 INTRODUÇÃO

Ao desenvolver um sistema, mesmo que simples, pode-se analisar duas vertentes muito importantes: funcionalidade e design. A primeira, é focada no funcionamento do software, ou seja, se esse realiza o que lhe foi proposto com corretude. Já a segunda analisa quais os melhores recursos que podem ou foram utilizados para que determinadas ações do software possam ser realizadas. Diante disso, verifica-se que existem inúmeros caminhos que podem ser seguidos para atingir a eficiência do software. Dentre esses, podemos avaliar situações em que o problema é solucionado com boas escolhas de design ou que é desenvolvido e atinge o esperado, porém sem muita preocupação com o design escolhido. Segundo [1], "GRASP patterns é hoje fundamental para que um design seja de boa qualidade", e é imprescindível destacarmos o quão pensar em uma boa solução, com boas práticas de design pode deixar um software robusto e reutilizável.

Atualmente, muitos estudos são desenvolvidos para caracterizar o design de um software já desenvolvido e em como isso influencia na funcionalidade do mesmo. De acordo com [2], para se obter um software de qualidade é necessário que se realizem medidas de extração que caracterizam esse código. Essas medidas de extração coletam métricas do código-fonte que, em conjunto, têm por objetivo identificar o quão robusto é um software com base no seu design.

Inúmeros ambientes acadêmicos utilizam programação orientada à objetos, fazendo com que a análise sobre como se comporta as escolhas do desenvolvedores seja bastante pertinente. A capacidade de abstrair um problema para o mundo real é um problema que desafia todos os desenvolvedores, até mesmo aqueles que já possuem experiência na área. Por esse motivo, os cursos de POO tem grande enfoque em atividades de resolução de problemas de programação. A opção por um ou outro design pode refletir o domínio do aluno sobre o tema que está sendo ensinado naquele momento, pois um problema atual consiste em, na maiorias dos casos, o professor não possuir uma resposta exata sobre como o aluno está se comportando dentro da disciplina em relação à escolha de design.

Sobre este ponto questiona-se: pode-se afirmar que desenvolvedores que escolhem determinada estratégia de design, previamente indicada ou não, possuem softwares mais robustos àqueles que não escolheram determinada implementação? Se agruparmos esses alunos com base em métricas extraídas, baseados no seu design, encontraremos padrões na nota desses estudantes?

Com o objetivo de conseguirmos agrupar alunos em clusters que representassem similaridades entre códigos, analisamos uma amostra com 50 códigos desenvolvidos pelos alunos da disciplina Laboratório de Programação II da Universidade Federal de Campina Grande. Foram extraídas dessa amostra, métricas que representassem melhor a escolha do design do aluno. Utilizando o K-Means, com essas métricas como parâmetro, geramos clusters que representam alunos que possuem códigos similares. Foram encontrados 3 clusters, com proporção de 50%, 30% e 20%, respectivamente. O cluster mais significativo foi o terceiro, que conseguiu um centróide com todas as métricas possuindo valores ótimos e também comprovou uma correlação entre os valores para as métricas extraídas e a nota obtida pelo aluno no código desenvolvido. Os dois primeiros clusters possuem centróides constituídos por métricas boas e ruins, o que contrasta em notas variadas, que são explicadas por os alunos terem obtidos valores ruins ou médios para pelo menos 1 das 6 métricas, fazendo com que estes fossem direcionados para outro cluster.

2 FUNDAMENTAÇÃO TEÓRICA

Em todo processo de desenvolvimento de um sistema é notável que a modelagem de um possível design sempre é uma etapa trabalhosa e que precisa ser analisada com cuidado. Diante disso, modelar e verificar se um software é bem desenvolvido é bastante importante para a área computacional.

2.1 Avaliação da qualidade de um software

Qualidade de um software é uma área consagrada na área de Engenharia de Software. Muitos estudiosos já discutiram, cada um com suas particularidades, o que seria um software ser de boa qualidade. Muitos afirmam que o fato de um componente ou processo realizar seu objetivo de acordo com os requisitos especificados e atender às expectativas do usuário já define que um software foi bem desenvolvido [3]. Para outros, essa qualidade é atingida quando temos um junção entre adequação entre os requisitos funcionais e desempenho, assim como boas escolhas de padrões de desenvolvimento [4]. Vale ressaltar que não há como comparar dois softwares em relação a eficiência apenas considerando se os mesmos cumpre seus requisitos funcionais [5].

Avaliar um software em relação ao seu design é diferente de analisar a sua funcionalidade, porém tanto design quanto funcionalidade estão interligados. A escalabilidade de um sistema pode ser diretamente influenciada de acordo com os padrões de design escolhidos, fazendo com que assim, o desenvolvedor precise tratar a etapa de modelagem de um software como essencial.

2.2 Trabalhos relacionados

Neste trabalho buscamos, através de métricas de design, agrupar códigos de alunos e analisar se os resultados das suas métricas, que representa sua escolha por determinado design, tem alguma correlação com a nota obtida. Portanto, extraímos do código suas características e o transformamos em um vetor. Outros pesquisadores já desenvolveram estudos, cuja metodologia vai nesta mesma direção, que se preocupam com a coleta de informações e em como ela pode trazer bons resultados para as análises.

O trabalho de [6], visa exclusivamente na extração de métricas e em conseguir interpretar o que cada uma delas representa, pois, segundo ele, muitas vezes existe só o trabalho de medir e não de justificar o esforço e o porquê de ter medido um software. Neste trabalho, foram coletadas 16 métricas de design e 19 de vulnerabilidade, com foco maior na última, pois métricas de design já são mais conhecidas e utilizadas no desenvolvimento de um software.

No estudo de [7], podemos encontrar justificativas sobre como o levantamento de métricas exclusivas de programação orientado à objetos podem ajudar na avaliação da qualidade de um software. Neste, foi desenvolvido uma ferramenta que foi inserida no ambiente de desenvolvimento SEA para o auxílio das extrações das métricas. Foram extraídas métricas de acoplamento, encapsulamento, complexidade, coesão e polimorfismo.

Os autores [2] analisam como a crescente demanda de desenvolvimento de software impulsiona também o estudo para extração de métricas. Este trabalho faz uma explicação de algumas métricas selecionadas e que caracterizam um programa orientado à objetos, porém com foco numa técnica capaz de extrair a métrica WMC (Weighted Methods per Class – Métodos Ponderados por Classe). Algumas das outras métricas medidas neste estudo são: profundidade na árvore de herança, número de filhos, acoplamento entre classes de objeto e falta de coesão em métodos.

O estudo de [8] visa buscar o quão igual são os sistemas analisados. Vale ressaltar que este trabalho, inicialmente procura encontrar um valor de referência que seja representativo para a métrica analisada, para que com isso possa explicar e utilizá-la de forma correta. Assim que esses valores são encontrados, verifica-se que: se um determinado sistema S1 é classificado como similar à um determinado sistema S2, através de um algoritmo de clusterização com n métricas dadas como entrada, e esse sistema S1 é considerado de alta qualidade, a probabilidade do sistema S2 também ser considerado bem desenvolvido é alta. Foram coletadas métricas que representam tamanho, acoplamento e complexidade.

3 METODOLOGIA

Nesta seção discutiremos a metodologia adotada neste trabaho. Partindo da observação que POO é repleta de conceitos, além da clusterização dos designs dos alunos, existirá um embasamento teórico sobre esses conceitos, tornando assim, justificável a decisão de escolher certas métricas na intenção de encontrar resultados.

Os alunos que desenvolveram o código que está em estudo são da Universidade Federal de Campina Grande da disciplina de Laboratório de Programação II. Essa escolha foi feita por motivos de que o primeiro contato entre os alunos e POO é feito no segundo período do curso, na disciplina citada acima. Para melhor demonstração da metodologia que será usada neste trabalho, foi escolhido um modelo BPMN (Business Process Model Notation), com 6 atividades principais representadas na figura 1.

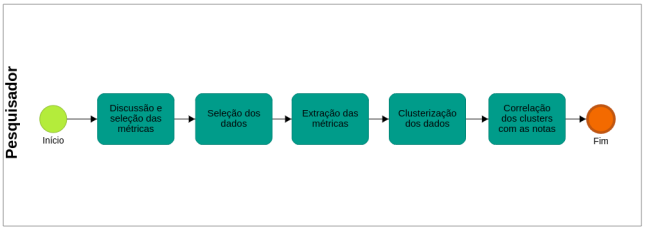

Figura 1: Modelo Business Process Model and Notation

A etapa 1 representa a atividade de seleção das métricas de design. A ideia é representar o código como um vetor composto pelo resultado da extração das métricas extraídas a partir do sistema desenvolvido. Diante disso, como o foco do trabalho é o desenvolvimento baseado no design, foram escolhidas 6 métricas que possam representar a modelagem escolhida. Vale ressaltar que, essa etapa de seleção e estudo das métricas é bastante relevante para o trabalho. É necessário cuidadoso estudo para a seleção de um conjunto de métricas que represente bem o código que está sendo analisado.

Na etapa 2, é feita a coleta de dados. Selecionamos as provas do terceiro estágio dos alunos. A escolha pelo terceiro estágio deu-se pelo fato desta ter maior abrangência de conteúdo, fazendo com que as métricas possam representar uma gama mais ampla de códigos desenvolvidos. Nesta etapa, foram analisados 50 códigos desenvolvidos por alunos de Ciência de Computação na UFCG.

Através de um plugin instalado no eclipse, é possível a extração das métricas na etapa 3. Dentre muitas ferramentas que são disponibilizadas para essa extração, o metrics foi o plugin selecionado pois ele abrange todas as métricas escolhidas e também por trabalhar com códigos desenvolvidos em Java. Vale ressaltar que, as métricas selecionadas são métricas de design conhecidas, o que facilita a extração das mesmas.

A etapa 4 dedica-se à clusterização dos códigos com bases nas métricas coletadas. Existem, na área de machine learning inúmeros algoritmos de clusterização com diferentes abordagens e propósitos. Como nosso objetivo é verificar a distância que códigos estão um do outro, foi escolhido o algoritmo de clusterização K-Means simples. Como o algoritmo trabalha de forma iterativa atribuindo os pontos aos dados que representam a menor distância, ou seja, ao grupo mais próximo, acredita-se que esse é um bom algoritmo de clusterização para o problema proposto. O parâmetro que será dado, para o algoritmo será k = 3, que irá representar a quantidade de clusters que para o estudo é válido.

Na etapa 5, é feita uma análise estatística através da correlação dos clusters encontrados com as notas dos alunos. Como citado acima, o número de clusters necessita ser previamente definido, então o algoritmo trabalha para agrupar os dados analisados na quantidade de cluster que é especificada. Como essa etapa visa contrastar os resultados da clusterização com as notas, será levado em consideração três intervalos de nota: [0;4,0], [4,1;7] e [7,1;10], o que nos leva a selecionar 3 clusters. A parte principal desta etapa é verificar se os alunos que estão no mesmo cluster, possuem suas notas dentro desses intervalos ou se o algoritmo de clusterização, com as métricas de design como parâmetros, não conseguiu identificar uma relação entre o cluster e sua nota.

3.1 Escolha das métricas

Como já discutido anteriormente, é possível quantificar o que se entende por qualidade software através da extração de métricas do código-fonte. Segundo [6], existem duas situações em que o uso de métricas acaba não sendo bom para o desenvolvimento, são elas: quando os desenvolvedores as usam de forma incorreta, não agregando valor ao desenvolvimento do software, ou quando apenas não as usam. Diante desses pressupostos, discutiremos nesta seção a razão pela qual cada uma das métricas foi utilizada neste trabalho.

Os sistemas que estão em análise neste estudo são programas pouco complexos. Como o presente trabalho é focado em analisar o design dos códigos, foram colhidas 6 métricas que visam abranger conceitos conhecidos em programação orientada a objetos, como: herança, coesão, encapsulamento e uso de controller. Além de métricas que representam quantidade, sendo elas: linhas de código e número de classes.

- **DIT (Depth of Inheritance Tree) - Profundidade da árvore de herança:** Essa métrica foi escolhida por representar herança, conceito bastante importante em POO. Ela mede a profundidade que uma classe se encontra na árvore de herança. Para linguagens com herança múltipla prevalece o DIT de maior hierarquia. Para esse estudo, foi levada em consideração a média dessa métrica, com base na quantidade de classes que o sistema tem e a quantidade que se relacionam por herança. Quanto maior o valor do DIT maior é a quantidade de classes que herdam de outras recursivamente, o que pode nos trazer um sistema com pelo menos uma classe muito complexa, porém em contrapartida, por se tratar de herança, um alto valor de DIT também representa alto reuso de código. Se uma classe não herda de nenhuma, seu valor de DIT será zero, se herda de uma, seu valor será um e assim sucessivamente;
- **LCOM (Lack of Cohesion of Methods) - Falta de coesão dos métodos:** Essa métrica explora um conceito bastante conhecido em POO, que é a coesão. Como uma classe é composta de atributos e métodos tem por objetivo modificar o valor de atributos de outras classes, essa métrica irá verificar quantos atributos serão modificados por métodos, de maneira independente, ou seja, quanto mais atributos um método de determinada classe modificar, mais responsabilidade essa classe irá ter e maior o seu valor de LCOM. Essa métrica irá medir se uma classe é coesa ou não, como estamos tratando de média, ela irá verificar todas as classes dos sistemas. Um alto valor de LCOM indica que o sistema possui muitas classes, ou pelo menos uma, que não são coesas;
- **ANPM (Average Number of Parameters of Methods) - Média do número de parâmetros por método:** Essa métrica foi escolhida por representar o princípio de atribuir responsabilidades únicas a um método, nos trazendo o conceito de coesão. É calculada a média de parâmetros por método, assim como nos dá o resultado da média por sistema. Seu valor mínimo é 0 e não possui um valor máximo, mas leva-se em consideração que um método que possui essa média muito alta, possui muitas responsabilidades, o que pode ferir os princípios de design do sistema;
- **Número de interfaces:** Uma interface no sistema permite uma melhor e maior interação entre o usuário e os objetos, facilitando assim a experiência de desenvolvimento do programador. Diante disso, essa métrica foi selecionada, por representar no nosso contexto, uma melhor solução de design. Para esse estudo, é interessante analisar quais alunos utilizaram interface e quais não, visto que esse é um assunto cobrado na disciplina e consequentemente na avaliação;
- **LOC (Lines of Code) - Número de linhas de códigos:** Essa métrica faz parte de um conjunto de métricas que representam quantidade, ela foi escolhida para avaliar o quanto ela influencia em nosso sistema. Deseja-se verificar se alunos que possuem mais linhas de código desenvolvido estarão em um cluster que representam alunos com melhores notas, daí podemos verificar se existe alguma relação entre essas variáveis;
- **Número de classes:** Assim como a métrica LOC, essa também representa quantidade. Desejamos verificar se o código que possue mais classes desenvolvidas tem um código mais robusto e se tem uma relação com notas altas, pois acredita-se que quanto mais classes, mais código desenvolvido e assim, mais funcionalidade e design para o código sendo analisado;

4 RESULTADOS

Essa seção tem por objetivo apresentar os resultados encontrados diante da aplicação da metodologia, citada anteriormente. Foram clusterizados 50 códigos de alunos da disciplina de laboratório de programação 2 da UFCG, de acordo com as métricas extraídas que representavam o seu design.

4.1 Avaliação e visualização individual sobre as métricas selecionadas

Foram selecionadas seis métricas, como citadas na seção da metodologia, para serem usadas como parâmetro para nossa clusterização. Inicialmente vamos discutir os resultados dos nossos dados e analisar o quão isso é representativo para o estudo, e logo após iremos analisar a clusterização dos códigos. Os dados são mostrados nos gráficos a seguir:

- **Número de classes:**

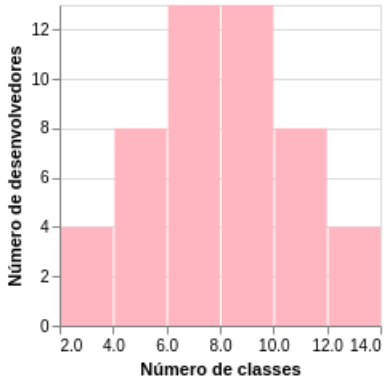

Figura 2: Gráfico que demonstra a relação entre número de classes e o número de desenvolvedores

De acordo com a figura 2, podemos visualizar que a maioria dos desenvolvedores, optaram por um sistema entre 6 à 10 classes. O gráfico é bem simétrico, ou seja, existem alunos que desenvolveram o sistema com poucas classes e com muitas classes. Considera-se para essa métrica, que um valor médio de classes seja bom, pois um sistema com poucas pode indicar menos funcionalidades, porém com muitas pode indicar um sistema que faz além do que o especificado e uma possível falta de refatoramento.

- **Profundidade da árvore de herança:**

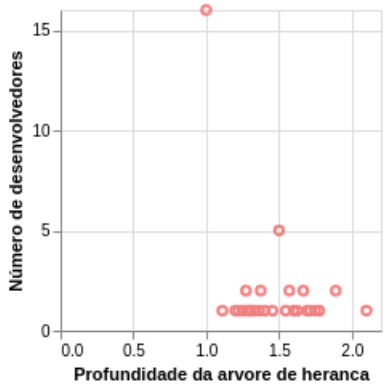

Figura 3: Gráfico que demonstra a relação entre a profundidade da árvore de herança e o número de desenvolvedores

A figura 3 mostra a dispersão dos pontos que relacionam a métrica profundidade da árvore de herança com seu respectivo desenvolvedor. Podemos verificar que existe uma grande concentração entre os valores acima de 1 e abaixo de 2. Em contrapartida, existem outliers que representam uma quantidade grande de desenvolvedores, como o caso do valor da métrica ser 1. Essa métrica tem por objetivo verificar herança no código desenvolvido, ou seja, valores representativos para essa métricas seriam os mais centrais e altos, pois quanto maior, mais indicação sobre reuso de código por herança.

- **Falta de coesão dos métodos**

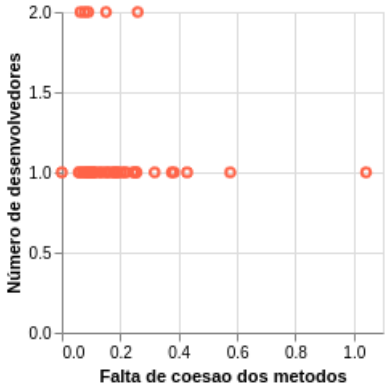

Figura 4: Gráfico que demonstra a relação entre a falta de coesão dos métodos e o número de desenvolvedores

Quanto menor o valor dessa métrica, melhor é a coesão do sistema desenvolvido. Podemos verificar na figura 4 uma forte concentração no valor 0 à pouco mais de 0.4, o que nos mostra bons resultados, pois indica que na maioria dos códigos desenvolvidos, o valor da falta de coesão do sistema foi baixo, ou seja, existem códigos mais coesos de acordo com essa métrica. Vale ressaltar que existe um outlier, que possui o valor dessa métrica maior do que 1.

- **Média dos parâmetros**

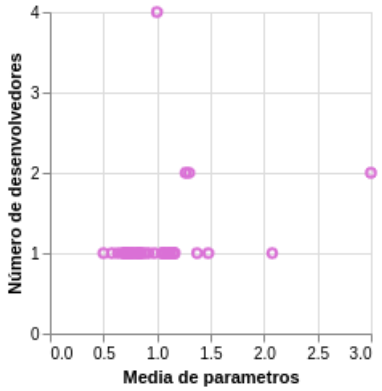

Figura 5: Gráfico que demonstra a relação entre as médias dos parâmetros e o número de desenvolvedores

O ideal para essa métrica é que ela possua valores mais baixos, pois quanto maior a média de parâmetros de um sistema, mais existem métodos que possuem muitas responsabilidades, ou seja, uma classe que realiza muitas operações no sistema, o que fere o Princípio de Responsabilidade Única. Podemos verificar, através do gráfico de dispersão, que os valores estão bem distribuídos, porém com uma forte concentração entre 0,5 à 1,5, mas com presença de outliers.

- **Total de linhas de código**

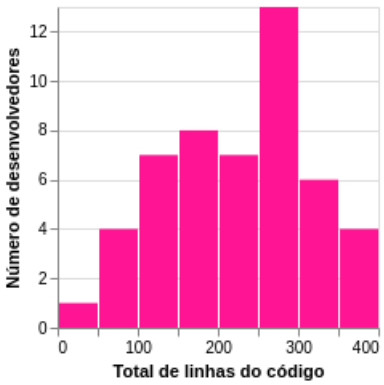

Figura 6: Gráfico que demonstra a relação entre o total de linhas de códigos e o número de desenvolvedores

O resultado para esta métrica é bastante variado: existem muitos alunos em quase todos os intervalos, porém com uma maior concentração em códigos entre 250 à 300 linhas desenvolvidas.

- **Número de interfaces**

Figura 7: Gráfico que demonstra a relação entre o uso da interface e o número de desenvolvedores

O uso de uma interface num sistema, como citado anteriormente, ajuda a melhorar a interação entre usuário e sistema, deixando o sistema mais ágil. O gráfico mostra que apenas 50% dos alunos utilizaram interface. Não existe um valor ideal para essa métrica, porém nesse contexto, o sistema ficaria melhor implementado se fizesse uso de uma interface, visto que ela era destacada na especificação.

4.2 Clusterização (K-means)

Diante da apresentação de cada uma das métricas extraídas, a discussão sobre a clusterização nos trará o porquê de cada código estar agrupado em determinado cluster. A ferramenta utilizada para realizar o tratamento das variáveis e realização do agrupamento foi a Weka. Esta ferramenta é bastante conhecida na área de machine learning e auxilia encontrando padrões e gerando hipóteses para os dados em questão.

Primeiramente, na demonstração acima, não fizemos normalização das variáveis, pois no tópico anterior o objetivo era só visualizar os dados que foram extraídos do código e não como eles se comportam dentro da clusterização. Para a etapa do agrupamento dos dados, as variáveis precisam ser normalizadas pois estamos tratando de variáveis numéricas, que entre si, possuem muita divergência de intervalos, como por exemplo: falta de coesão de métodos e número de linhas de código.

Após o processo de normalização das variáveis, os resultados da nossa clusterização são encontrados. O algoritmo utilizado foi o K-Means simples, que é não-supervisionado, com o número de clusters pré-definido igual a 3. Os primeiros resultados a serem discutidos são mostrados na figura 8 abaixo:

```
=== Run information ===

Scheme:       weka.clusterers.SimpleKMeans -init 0 -max-candidates 100 -periodic-pruning 10000
Relation:     dados-tcc - Página1 (2)-weka.filters.unsupervised.attribute.Normalize-S1.0-T0.0
Instances:    50
Attributes:   8
              Numero de classes
              Profundidade da arvore de heranca
              Falta de coesao dos metodos
              Media de parametros
              Total de linhas do codigo
              Numero de interfaces
Ignored:
              Nome
              Nota
Test mode:    evaluate on training data
```

Figura 8: Variáveis selecionadas e ignoradas

O trabalho foi realizado com 6 variáveis que foram levadas em consideração para a clusterização, sendo 2 delas ignoradas pois não era relevantes para fazer a separação, que são nome e data. Para entender melhor como foi feita a clusterização em grupos, discutiremos alguns conceitos no contexto do Weka.

```
=== Clustering model (full training set) ===


kMeans
======

Number of iterations: 6
Within cluster sum of squared errors: 10.078637148857279

Initial starting points (random):

Cluster 0: 0.363636,0.454545,0.109405,0.1,0.412429,1
Cluster 1: 0.636364,0.201818,0.144914,0.2216,0.548023,1
Cluster 2: 0.818182,0.495455,0.06142,0.1392,0.621469,1

Missing values globally replaced with mean/mode

Final cluster centroids:
                                        Cluster#
Attribute                 Full Data           0           1           2
                             (50.0)      (25.0)      (15.0)      (10.0)
=============================================================================
Numero de classes           0.5109      0.4182      0.4848      0.7818
Profundidade da arvore de heranca       0.3106      0.3469      0.1327      0.4865
Falta de coesao dos metodos 0.1757      0.2158       0.157      0.1034
Media de parametros         0.2192      0.2802      0.1856      0.1171
Total de linhas do codigo   0.5025      0.3994      0.5147      0.7421
Numero de interfaces          0.5           0           1           1
```

Figura 9: Valores encontrados para cada centróide

Inicialmente, para cada código (dado), é calculado uma distância euclidiana com os seus valores de métricas, posicionando-os no plano da dimensão. Feito isso, o algoritmo do k-means gera centróides iniciais aleatórios, como demonstrado na figura acima, e é executado n vezes sempre recalculando o valor destes, com base nos elementos que agora estão presentes no cluster, verificando se os dados pertencem a um novo cluster ou não até estes não se moverem mais na dimensão analisada. O número de iterações no nosso algoritmo foi 6, ou seja, o algoritmo conseguiu agrupar melhor cada dado no seu centróide na 6ª iteração. Para o algoritmo saber qual foi a iteração que melhor classificou os dados ele usa a soma dos erros quadrados, que representa a distância entre os dados e seu centróide, o que tiver o menor valor de SSE (sum of squared errors) é escolhido para representar a melhor iteração. Uma boa maneira de diminuir o valor do SSE é aumentando o valor de clusters, pois assim, reduziríamos mais ainda a quantidade de dados por cluster, fazendo-os estarem mais próximos do seu centróide. A figura 9 destaca o valor médio de cada métrica para cada um dos clusters.

```
Time taken to build model (full training data) : 0.02 seconds

=== Model and evaluation on training set ===

Clustered Instances

0       25 ( 50%)
1       15 ( 30%)
2       10 ( 20%)
```

Figura 10: Proporção de códigos por cluster

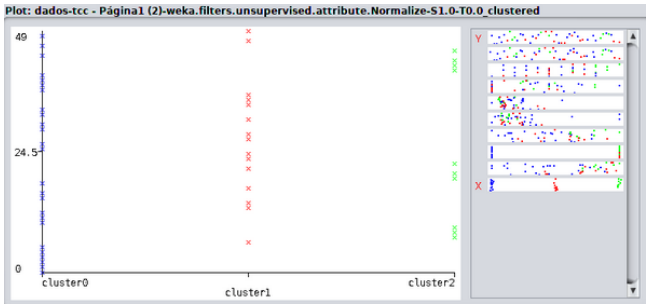

Figura 11: Clusterização código versus instância do código

Os dados foram divididos em 3 clusters, como pode-se verificar na figura 10, sendo o primeiro deles com metade dos dados, o que representa 50%. O segundo com 15 dados e o terceiro com 10, representando, respectivamente 30% e 20%. Para mostrar o resultado da clusterização, a figura 11 representa um gráfico que seu eixo x é a instância do dado, ou seja, um número simbólico que o representa (1 à 50) e seu eixo y representa o cluster à qual pertence.

4.3 Clusterização versus nota

Essa etapa consiste em visualizar se, dada a clusterização, existe alguma relação entre os clusters e as notas que os alunos receberam pelo desenvolvimento do software. Para isso, a figura 12 nos traz um gráfico que seu eixo x representa o cluster e o eixo y representa a nota obtida. Nela, podemos verificar que existe uma maior distribuição no cluster 0, seguida pelo cluster 1 e finalizada com o cluster 2. Isso nos mostra que existe no cluster 0, alunos com notas muito diferentes em todos os intervalos, diferente dos outros dois clusters, que abriga alunos com notas dentro de intervalos mais restritos.

Figura 12: Clusterização código versus nota

5 ANÁLISE

Para entendermos melhor qual o cluster que conseguiu agrupar desenvolvedores que possuíam as melhores métricas, vamos observar a tabela abaixo:

Tabela 1: Tabela de relação entre valores desejados nas métricas e valores (centróides) obtidos em cada cluster

Métrica	Valor almejado	Cluster 0	Cluster 1	Cluster 2
Número de classes	VALOR ALTO	VALOR BAIXO	VALOR MÉDIO	VALOR ALTO
Profundidade da árvore de herança	VALOR ALTO	VALOR MÉDIO	VALOR BAIXO	VALOR ALTO
Falta de coesão dos métodos	VALOR BAIXO	VALOR ALTO	VALOR MÉDIO	VALOR BAIXO
Média de parâmetros	VALOR BAIXO	VALOR ALTO	VALOR MÉDIO	VALOR BAIXO
Total de linhas de códigos	VALOR ALTO	VALOR BAIXO	VALOR MÉDIO	VALOR ALTO
Número de interfaces	VALOR IGUAL À 1	VALOR IGUAL À 0	VALOR IGUAL À 1	VALOR IGUAL À 1

Podemos verificar que o cluster 2 é considerado o melhor em relação ao design, pois o centróides de cada métrica possui os valores mais próximos do desejado, de acordo com o que já foi discutido na fundamentação teórica levando em consideração outros trabalhos presentes na literatura. Estar nesse cluster significa que para todas as métricas escolhidas neste estudo, seus valores foram bons, o que pode resultar em um melhor design do sistema.

5.1 Análise individual dos clusters

Com base na figura 12, essa seção tem por objetivo discutir as particularidades encontradas em cada cluster.

5.1.1 Cluster 0. Esse cluster agrega a maior quantidade de códigos. Uma das características importantes desse cluster é: todos os códigos que obtiveram nota abaixo de 4 estão presentes nele, ou seja, ele tem uma relação direta com alunos que tiraram nota baixa. Todos os desenvolvedores que não optaram por usar interfaces, foram alocados para este, o que justifica o fato de alguns alunos que tiraram nota alta estarem nesse agrupamento. Como citado acima, as métricas juntas nos trazem um resultado, ou seja, se alunos obtiveram resultados bons em determinada métrica mas ruins em outras isso irá influenciar na sua posição no plano em estudo, o que muda diretamente para qual cluster ele irá ser alocado. Podemos classificar este inicial, como àquele que possui alunos de variadas notas, mas que abrigam majoritariamente alunos com notas baixas, porém com presença de outliers, que são os alunos que obtiveram notas boas na correção, tendo como justificativa estes possuírem métricas com valores bons, mas também algumas que não têm valores bons, como por exemplo: o não uso de interface.

5.1.2 Cluster 1. Podemos verificar que o cluster 1 possui alunos com notas médias e altas, ou seja, notas baixas não estão presente nesse cluster. Diferente do primeiro, aqui temos alunos que optaram pelo uso da interface, porém se analisarmos a tabela 1, podemos verificar que os valores dos centróides deste cluster são sempre valores medianos, o que justifica os alunos que tiraram entre 4 à 7 terem sido alocados para esse cluster. O valor de profundidade da árvore de herança nessa região é bastante pequeno, o que nos mostra que esses alunos usaram pouco reuso por herança. Existem os outliers também, que são os estudantes que tiraram nota alta, tendo como justificativa a mesma para o cluster 0.

5.1.3 Cluster 2. Esse cluster é o mais representativo ao fazer o contraste com as notas. Todos os dez alunos que estão presentes neste possuem nota acima de 7. Vale ressaltar que a maior nota também está presente nesse cluster. Se analisarmos a tabela 1, podemos verificar que esse resultado era o esperado, pois esse cluster foi o melhor classificado para as métricas selecionadas. Podemos dizer que o fato de as melhores notas estarem presentes aqui, significa que para todas as métricas esses alunos obtiveram bons valores, o que nos indica uma correlação entre as métricas extraídas e a nota obtida.

6 CONCLUSÃO

É imprescindível destacarmos o quão um bom conjunto de métricas é importante para uma boa clusterização, pois assim é possível provar se existe uma similaridade entre os alunos presentes no cluster (mesmo que pequena) ou uma possível relação entre bons valores de métrica de design e a nota obtida. Como analisado no estudo, mesmo que um conjunto de métricas seja ótimo e consiga definir e caracterizar um cluster, existem problemas que podem invalidar o cluster analisado, são esses os causadores dos outliers. O principal causador citado no estudo foi alunos que possuem métricas com bons valores, mas também possuem métricas com valores ruins, ou seja, a clusterização é bastante dependente do valor das métricas, pois esses valores extraídos depende do código desenvolvido por outro desenvolvedor. Por causa desses outliers, pode verificar que não é necessário ter bons valores em todas as métricas para o aluno ter uma boa nota, porém todos os alunos que estão com valores bons nas métricas extraídas estão agrupados num cluster onde todos os códigos estão com nota acima de 7, o que faz com que o software seja mais robusto. A validação prévia das métricas seria bastante enriquecedor para o estudo. Um experimento piloto com uma amostra menor seria interessante para fazer um teste com o conjunto métricas para saber se este é representativo. A metodologia do piloto pode ser mostrada na tabela abaixo.

Tabela 2: Tabela com metodologia do piloto para trabalhos futuros

Passos	Ações
1	Levantamento do diagrama de classe de cada código
2	Formulário perguntando quais códigos visualmente são mais parecidos com o outro
3	Espécie de clusterização visual de acordo com o formulário sem relação com as métricas
4	Verificar se os clusters possuem as métricas em comum

Na etapa 4 do piloto acima, ao fazermos a verificação podemos concluir que, se depois da clusterização manual os códigos que

estiverem no mesmo cluster possuírem métricas muito diferentes, talvez essa não seja uma métrica que nos trará bons resultados, pois quando formos clusterizar com base nelas, não ocorrerá uma boa divisão. Porém esse estudo é muito dependente da avaliação feita através dos formulários, fazendo com que seja necessário um bom número de respostas dessa pesquisa para a classificação visual não ser enviesada.

É importante sempre procurar realizar o estudo com novas métricas, pois atualmente existem inúmeras que representam design, assim é possível compreender melhor o intuito de uma métrica e também o quão ela pode nos informar sobre um sistema. Um novo conjunto delas, poderia para esse estudo, nos trazer resultados melhores para o cluster 0 e 1, ajudando na melhor identificação de padrões nos mesmos.

REFERÊNCIAS

[1] R. R. Gudwin, "Componentes, frameworks e design patterns," 2010.

[2] L. V. Martinez, M. S. Arrieira, and C. M. Betemps, "Extração da métrica wmc a partir de código java," *ENCONTRO ANUAL DE TECNOLOGIA DA INFORMAÇÃO E SEMANA ACADÊMICA DE TECNOLOGIA DA INFORMAÇÃO*, vol. 5, pp. 228–234.

[3] I. S. C. Committee *et al.*, "Ieee standard glossary of software engineering terminology (ieee std 610.12-1990). los alamitos," *CA: IEEE Computer Society*, vol. 169, 1990.

[4] R. S. Pressman, *Software engineering: a practitioner's approach*. Palgrave Macmillan, 2005.

[5] P. Simmons, "Quality outcomes: Determining business value," *IEEE Software*, vol. 13, no. 1, pp. 25–32, 1996.

[6] L. K. Duarte, "Análise de métricas de código fonte: Além das métricas de design de código,"

[7] W. R. d. Fonseca *et al.*, "Ferramenta de extração de métricas para apoio à avaliação de especificações orientadas a objetos," 2002.

[8] P. M. de Oliveira, H. S. Borges, M. T. Valente, and H. A. X. Costa, "Uma abordagem para verificação de similaridade entre sistemas orientados a objetos,"