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

João Pedro Silva de Melo

ANALISANDO ERROS COMETIDOS POR INICIANTES EM PROGRAMAÇÃO ORIENTADA A OBJETOS NO CURSO DE CIÊNCIA DA COMPUTAÇÃO DA UFCG

CAMPINA GRANDE - PB

2024

JOÃO PEDRO SILVA DE MELO

ANALISANDO ERROS COMETIDOS POR INICIANTES EM PROGRAMAÇÃO ORIENTADA A OBJETOS NO CURSO DE CIÊNCIA DA COMPUTAÇÃO DA UFCG

Trabalho de Conclusão de 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: Lívia Maria Rodrigues Sampaio Campos

CAMPINA GRANDE - PB

2024

JOÃO PEDRO SILVA DE MELO

ANALISANDO ERROS COMETIDOS POR INICIANTES EM PROGRAMAÇÃO ORIENTADA A OBJETOS NO CURSO DE CIÊNCIA DA COMPUTAÇÃO DA UFCG

Trabalho de Conclusão de 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:

Lívia Maria Rodrigues Sampaio Campos

Orientadora – UASC/CEEI/UFCG

Eliane Cristina de Araújo

Examinador – UASC/CEEI/UFCG

Francisco Vilar Brasileiro

Professor da Disciplina TCC – UASC/CEEI/UFCG

Trabalho aprovado em 15 de Maio de 2024.

CAMPINA GRANDE - PB

RESUMO

Cursos introdutórios de programação costumam ser desafiadores, especialmente para aqueles sem experiência prévia em lógica de programação. Ao ingressar em um curso de tecnologia, é necessário o aprendizado de terminologias relacionadas à Ciência da Computação e adaptar-se aos diferentes conceitos e paradigmas. Se tratando de Programação Orientada a Objetos (POO), utilizando Java como linguagem de programação, a dificuldade é ainda maior. Seu amplo conjunto de conceitos e sua sintaxe verbosa fazem parte de uma transição desafiadora, seguida de erros quanto aos conceitos aprendidos, refletindo muitas vezes a permanência do aluno no curso.

Sendo assim, com o objetivo de compreender e analisar estes erros, este trabalho se propõe a seguir os mesmos passos de um estudo anterior conduzido em uma turma da disciplina de Laboratório de Programação II no curso de Ciência da Computação da Universidade Federal de Campina Grande. A dificuldade de assimilação de conceitos de Java implica na observação de erros em atividades práticas e avaliativas, como foi constatado no estudo mencionado, que analisou principalmente o volume de código das submissões dos alunos. A intenção é fazer um comparativo entre os resultados, investigando padrões e tendências que possam variar conforme há mudança de amostra.

Uma vez identificando tais erros, bem como a sua frequência e impacto na nota do aluno, este estudo pode contribuir para potenciais adaptações nos métodos de ensino para que haja uma melhor aproximação na solução destes problemas, fortalecendo a compreensão dos conceitos fundamentais de Programação Orientada a Objetos, uma vez que é recorrente em outras disciplinas da grade curricular.

ANALYZING ERRORS MADE BY BEGINNERS IN OBJECT ORIENTED PROGRAMMING IN THE COMPUTER SCIENCE COURSE AT UFCG

ABSTRACT

Introductory programming courses are often challenging, especially for those with no prior experience in programming logic. When entering a technology course, it is necessary to learn terminologies related to Computer Science and adapt to different concepts and paradigms. When it comes to Object Oriented Programming (OOP), using Java as a programming language, the difficulty is even greater. Its broad set of concepts and its verbose syntax make part of a challenging transition, followed by errors regarding the concepts learned, often reflecting the student permanence on the course.

Therefore, with the aim of comprehending and analyzing these errors, this work proposes to follow the same steps as a previous study conducted in a Programming Laboratory II class in the Computer Science course at Universidade Federal de Campina Grande. The difficulty in assimilating Java concepts results in the discovery of errors in practical activities and tests, as demonstrated by the aforementioned study, which primarily analyzed the volume of code from student submissions. The intention is to compare the results, investigating patterns and tendencies that may vary with changes in the sample.

Once identifying such errors as well as their frequency and overall impact on the student grades, this study can contribute to potential adaptations in the teaching methods so that there is a better approach to solving these problems, strengthening the understanding of the fundamental concepts of Object Oriented Programming, as it is recurrent in other classes on the curriculum

ANALISANDO ERROS COMETIDOS POR INICIANTES EM PROGRAMAÇÃO ORIENTADA A OBJETOS NO CURSO DE CIÊNCIA DA COMPUTAÇÃO DA UFCG

João Pedro Silva de Melo
Universidade Federal de Campina Grande
Campina Grande, Paraíba, Brasil
joao.melo@ccc.ufcg.edu.br

Lívia Maria Rodrigues Sampaio Campos
(Orientadora)
Universidade Federal de Campina Grande
Campina Grande, Paraíba, Brasil
livia@computacao.ufcg.edu.br

RESUMO

Cursos introdutórios de programação costumam ser desafiadores, especialmente para aqueles sem experiência prévia em lógica de programação. Ao ingressar em um curso de tecnologia, é necessário o aprendizado de terminologias relacionadas à Ciência da Computação e adaptar-se aos diferentes conceitos e paradigmas. Se tratando de Programação Orientada a Objetos (POO), utilizando Java como linguagem de programação, a dificuldade é ainda maior. Seu amplo conjunto de conceitos e sua sintaxe verbosa fazem parte de uma transição desafiadora, seguida de erros quanto aos conceitos aprendidos, refletindo muitas vezes a permanência do aluno no curso.

Sendo assim, com o objetivo de compreender e analisar estes erros, este trabalho se propõe a seguir os mesmos passos de um estudo anterior conduzido em uma turma da disciplina de Laboratório de Programação II no curso de Ciência da Computação da Universidade Federal de Campina Grande. A dificuldade de assimilação de conceitos de Java implica na observação de erros em atividades práticas e avaliativas, como foi constatado no estudo mencionado, que analisou principalmente o volume de código das submissões dos alunos. A intenção é fazer um comparativo entre os resultados, investigando padrões e tendências que possam variar conforme há mudança de amostra.

Uma vez identificando tais erros, bem como a sua frequência e impacto na nota do aluno, este estudo pode contribuir para potenciais adaptações nos métodos de ensino para que haja uma melhor aproximação na solução destes problemas, fortalecendo a compreensão dos conceitos fundamentais de Programação Orientada a Objetos, uma vez que é recorrente em outras disciplinas da grade curricular.

Palavras-Chave

Programação Orientada a Objetos, Erros, Programação, Java.

1. INTRODUÇÃO

Programação Orientada a Objetos (POO) é um paradigma que auxilia programadores a abstrair objetos reais e recriá-los em linhas de código [6]. Se sustenta na ideia de classes, um tipo de dado que inclui propriedades e métodos — blocos executados sempre que chamados — os quais se unem para oferecer uma funcionalidade específica. Sua importância reside em prover uma maneira mais intuitiva de descrição de objetos do mundo real com seus respectivos atributos, através do uso de classes e objetos. A estrutura organizada do paradigma, que divide os objetos em classes, garante uma melhor legibilidade de código e, consequentemente, uma fácil usabilidade.

A transição do estudante do aprendizado de programação procedural para POO pode se mostrar desafiadora, com motivos que variam desde a capacidade individual até a metodologia de ensino. Além disso, há uma diversificação de conceitos subjacentes, cujo aprendizado depende da forma como são abordados [7].

Durante os estudos de POO, a linguagem mais comumente utilizada é Java, uma vez que existe um amplo suporte para o paradigma. Por ser uma linguagem bastante popular e utilizada [8], ela se torna presente nos cursos de ensino superior de tecnologia ao redor do mundo.

Uma aprendizagem precária de POO resulta em erros e equívocos em atividades práticas e avaliativas [3]. A complexidade dos conceitos de orientação a objetos como Polimorfismo, Abstração, Herança, Encapsulamento e Coesão revelou-se tema de diversos estudos [1][2][5][9], visando trazer uma melhor compreensão dos erros encontrados em atividades de estudantes que estão se familiarizando com o paradigma.

Neste cenário, visando analisar os erros cometidos pelos estudantes na disciplina de Laboratório de Programação II, este estudo se baseia em um trabalho prévio [12] realizado no semestre 2019.2, cuja proposta era a mesma: identificar e analisar os erros cometidos por estudantes iniciantes no paradigma de POO. Os dados utilizados serão do semestre 2023.1, mas mantendo a categorização dos erros feita anteriormente.

Ao fim deste trabalho, espera-se haver uma contribuição positiva

para a disciplina de Laboratório de Programação II da UFCG, de forma que os insights fornecidos atuem para possíveis adaptações na metodologia de ensino ou aplicabilidade de atividades, beneficiando tanto os instrutores quanto a passagem dos estudantes no curso.

2. FUNDAMENTAÇÃO TEÓRICA

Os estudos que buscaram identificar erros durante o aprendizado de POO concentraram-se nos erros conceituais [5][12] e nos erros de sintaxe, semântica e lógica [6][10]. Uma linguagem complexa como Java pode ser desafiadora até mesmo para docentes [10][11], onde uma compreensão imprecisa dos conceitos e de sua própria estrutura influencia negativamente na correção de avaliações.

Dentre as abordagens utilizadas nos estudos, estão ferramentas de análise automática [4] e análises baseadas no nível de desempenho em disciplinas de programação [5]. Não são considerados apenas erros relacionados ao paradigma em si, mas àqueles associados à negligência da sintaxe da linguagem, como uso incorreto de operadores e a falta de semicolon[1] [6], sendo esse tipo de erro o mais prevalente [2]. Contudo, são erros fáceis de identificação e resolução, ao contrário de erros relacionados a semântica [5].

No estudo realizado por Jegede et al. [5], que buscava analisar os erros de estudantes de diferentes níveis, a amostra totalizava 124 alunos dos cursos de computação e engenharia da Universidade Obafemi Awolowo, na cidade de Ifé, na Nigéria. Houve a separação em três grupos com base no nível de desempenho (baixo, médio e alto) em provas anteriores ao semestre em estudo. Os autores identificaram 598 erros ao todo, sendo o grupo de baixo desempenho o que mais cometeu erros relacionados a conceitos de POO, assim como outros erros de sintaxe. Após os resultados, entrevistas foram feitas com um grupo de alunos selecionados aleatoriamente, com o propósito de compreender mais de perto a causa dos erros encontrados. Segundo os alunos entrevistados, as razões que levaram ao cometimento dos erros incluíam desde a restrição ao uso de IDEs[2] durante as provas até a dificuldade durante o aprendizado da linguagem Java.

Em um contexto semelhante ao estudo de Jegede et al. [5], Leal [12] conduziu uma pesquisa em uma turma de Laboratório de Programação II do curso de Ciência da Computação da Universidade Federal de Campina Grande, buscando entender os erros cometidos por iniciantes em avaliações da disciplina. Para isto, ele utilizou a segunda prova do semestre de 2019.2 de uma turma de 91 alunos. Filtrando os grupos de alunos de acordo com o volume de código (baixo, médio e alto), foi possível analisar com mais precisão o impacto dos erros relacionados ao paradigma de POO nas notas dos alunos. Em sua conclusão, observou-se que alunos que produziram um volume alto de código apresentaram menos erros, e que diferentes tipos de erros tiveram impactos variados nas notas finais dos alunos. No entanto, submissões com baixo volume de código não necessariamente resultaram em baixas pontuações.

Resumindo, as abordagens utilizadas nestes trabalhos compartilhavam de um mesmo objetivo: identificar os erros cometidos por alunos iniciantes. Apesar de diferir nas metodologias, o objetivo era oferecer sugestões que pudessem auxiliar o corpo docente responsável por essas disciplinas com base nos resultados obtidos.

3. METODOLOGIA

Esse estudo possui a mesma metodologia de um trabalho anterior com um contexto semelhante, mas considerando um conjunto de dados diferente. Dessa forma, é uma pesquisa de natureza descritiva que segue um método quali-quantitativo, baseado na análise textual das anotações dos professores sobre a correção de atividades avaliativas de programação em Java, dentro do paradigma de POO. O estudo considera a disciplina Laboratório de Programação II (LP2), do curso de Ciência da Computação da UFCG. Com isso, espera-se responder às seguintes questões de pesquisa:

- QP1: Como os erros influenciam no desempenho dos estudantes?
- QP2: Como se caracterizam, dado o programa em que estão contidos e o programador que os cometeu, os diferentes tipos de erros?
- QP3: Qual a frequência dos erros?

Desta forma, será feita a comparação dos resultados obtidos neste estudo com os resultados do estudo anterior.

3.1 Contexto

A disciplina de LP2 é lecionada em conjunto com Programação II, e tem como objetivo proporcionar aos alunos o aprendizado do uso de ferramentas de desenvolvimento e, essencialmente, o aprofundamento prático dos conceitos de POO abordados na disciplina principal.

As avaliações são computadas considerando as médias obtidas nos laboratórios da disciplina e nas provas. Alguns semestres também contemplam projetos em equipe, onde os alunos podem demonstrar o conhecimento que obtiveram na disciplina desenvolvendo aplicações com utilidade no mundo real. No contexto desta pesquisa, foram utilizados os dados de correção da segunda prova de LP2, do semestre de 2023.1, aplicada em três turmas diferentes. As turmas tinham professores distintos, mas ainda assim o conteúdo lecionado e as avaliações eram sincronizadas entre as mesmas.

3.2 Amostra

Os dados incluem critérios de correção da prova e a pontuação associada a cada critério. A prova do semestre em questão consistia na implementação de um sistema de uma empresa fictícia, representando uma funcionalidade de um clube de descontos para os clientes (usuários). O objetivo era permitir que os mesmos pudessem cadastrar, acumular e aplicar cupons de diferentes tipos sobre suas compras. Além das funcionalidades básicas esperadas, também eram necessários testes e tratamento de exceções na implementação.

A nota total de um aluno é constituída de cinco notas, cada qual com seus respectivos pesos e comentários atribuídos pelos professores da disciplina, seguindo uma rúbrica de correção com o que se espera em cada uma dessas partes. Uma vez observando os comentários, foi possível ter uma noção de como categorizá-los.

A planilha com os dados continha ao todo 97 estudantes, mas desconsiderando os que faltaram a prova ou aqueles cuja

[1] Ponto-e-vírgula, utilizado no final de cada instrução individual, indicando o fim de uma entidade lógica.

[2] Softwares utilizados durante o desenvolvimento de aplicações, sendo IDE acrônimo para Ambiente de Desenvolvimento Integrado.

submissão por algum motivo não se encontrava no arquivo *.zip* da turma fornecido pelos professores, apenas 61 deles foram incluídos na amostra.

Assim como proposto por Leal [12], o fator predominante nas análises é o volume de código, sendo ele categorizado em "Baixo", "Médio" e "Alto".

3.3 Coleta de métricas sobre o código

Apesar de não ter sido feita nenhuma análise no código em si, fora utilizado, em cada submissão individual, o plugin do Eclipse® chamado *Metrics*[3], para extração das métricas de código.

As métricas escolhidas foram as mesmas do estudo anterior:

- Número de classes;
- Número total de linhas de código;
- Número de métodos;
- Número de atributos;
- Complexidade Ciclomática;
- Acoplamento.

Embora algumas métricas sejam intuitivas, outras podem ser menos familiares, como Complexidade Ciclomática e Acoplamento.

Complexidade Ciclomática refere-se ao número quantitativo de caminhos independentes e linearmente independentes em um programa, representando o número de decisões tomadas por meio de loops, condicionais e operadores lógicos.

Por outro lado, o conceito de Acoplamento diz respeito à dependência entre os dados que uma classe utiliza de outra classe. Isso ocorre quando uma classe tem acesso aos atributos de outra classe, seja dentro do mesmo pacote ou externamente.

A coleta destas métricas serviu para evidenciar a relação com os erros cometidos pelos alunos.

3.4 Tipos de erros

A Tabela 1 abaixo mostra os tipos de erros considerados neste estudo, bem como no estudo de Leal [12].

Erro	Descrição
Método	Métodos longos, com múltiplas funcionalidades ou nomes incoerentes.
Abstração	Não compreender corretamente o que fazer em uma funcionalidade do código, ignorar detalhes específicos ou adicionar desnecessários.
Lei de Deméter	Objetos que acessam diretamente métodos de objetos relacionados.
Encapsulamento	Estado interno de um objeto manipulado por outros objetos.
Coesão	Classes ou métodos realizando tarefas que não estão relacionadas com o que lhes fora especificado.
Compilação	Impede o código de ser executado.
Lógica	Não produz resultados esperados.
Completude	Ausência de partes necessárias que foram especificadas, podendo ou não interferir no funcionamento do programa.
Tratamento de Erros	Ausência de exceções ou verificações que tornem mais fácil o entendimento de bugs ou quaisquer problemas que possam surgir durante a codificação.

Tabela 1. Erros e suas respectivas descrições, de acordo com o livro *Clean Code.*

Erros e equívocos conceituais relacionados a herança, polimorfismo e composição foram deixados de fora neste trabalho. Ainda assim, podem ser considerados em investigações futuras, utilizando metodologias diferentes e mais robustas.

É de interesse frisar que a definição de um erro pode variar de acordo com níveis de experiência, estilos de programação e entendimento do problema. Portanto, é esperado que os resultados deste estudo possam diferir em relação ao estudo anterior.

3.5 Análise de dados

A planilha fornecida pelos professores com as notas dos alunos e seus comentários foram adicionados em uma outra planilha, já estruturada com as colunas de erros e métricas. Após organizar a nova planilha, adicionando as métricas extraídas e inspecionando isoladamente cada aluno, os dados foram armazenados em um Data Frame, utilizando a biblioteca de Python, Pandas, para as manipulações e análises.

Uma vez fazendo a limpeza de dados, retirando da amostra alunos que não compareceram a prova e valores nulos, houve a identificação da quantidade total de erros. Contemplando a análise do impacto do erro no desempenho, foram utilizados Coeficientes de Correlação para medir relações entre as variáveis de erro e nota. Em seguida, foi aplicado o Modelo de Regressão Linear Múltipla para averiguar quais erros possuem maior significância estatística.

Uma outra análise foi sobre o volume de código e os erros. Houve a filtragem das métricas para identificar quais seriam mais relevantes na dispersão de dados, seguida da separação dos grupos de acordo com o volume de código. Dito isto, foi utilizado o procedimento matemático conhecido como *PCA*[4](Análise de Componentes Principais), que sumariza grandes informações em grupos menores chamados de componentes. Observando os valores dos componentes e usando um correlograma para firmar esta observação, as métricas que mais contribuíram para caracterização do código foram:

- Número de Linhas;
- Número de Classes;
- Número de Atributos.

Com essas métricas, por fim, se fazia necessário rotular e agrupar, e para isso foi utilizado o algoritmo *K-Means*, que divide um conjunto de pontos em clusters, de maneira que os pontos dentro

[3] Eclipse Metrics Plugin.

[4] Principal Component Analysis

de um cluster possuam mais similaridades entre eles do que entre pontos de outros clusters. O agrupamento foi feito por semelhança entre as variáveis, conforme o número de clusters passado como entrada (*k=3*), assim obtendo as médias dos grupos. O valor dos clusters foi encontrado utilizando o *Elbow Method,* uma técnica que permite determinar o número ideal de clusters em um conjunto de dados quando se utiliza o algoritmo *K-Means.*

4. RESULTADOS

Esta seção apresenta os resultados obtidos a partir da análise dos dados do estudo, para responder às questões de pesquisa identificadas na seção 3.

4.1 Como os erros influenciam no desempenho dos estudantes?

Considerando os 61 alunos que fizeram a prova, foi confirmado um total de 432 erros independente do tipo.

Observando a Figura 1, percebe-se que a maioria dos alunos cometeu entre 4 e 6 erros. No entanto, é notável que uma parcela significativa apresentou 9 erros em seu total.

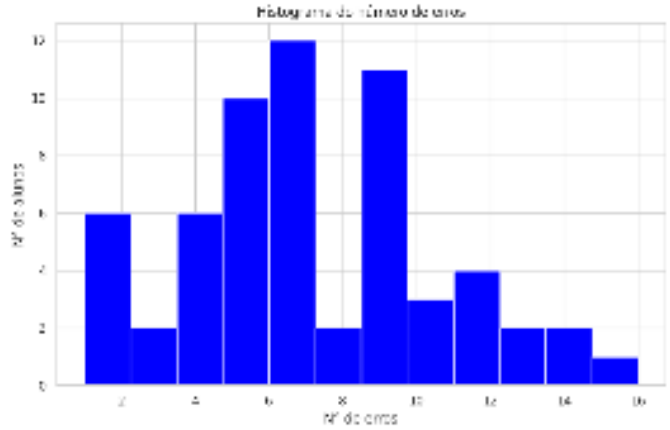

Figura 1. Histograma do número de erros cometidos por alunos.

Em relação a quantidade de erros e a nota obtida, alunos que totalizaram um número menor de erros tiveram uma nota maior, conforme mostra a concentração dos pontos no gráfico de dispersão abaixo.

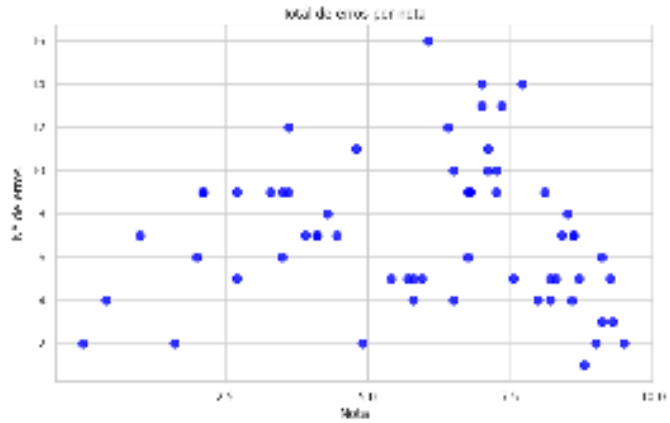

Figura 2. Dispersão do número de erros por nota.

Além disso, é interessante notar que, apesar de alguns alunos terem cometido um número significativo de erros, ainda conseguiram alcançar uma nota razoável — embora abaixo da média. Isso implica que diferentes tipos de erros podem ter pesos variados na determinação da nota final do aluno.

As correlações entre a quantidade de erros e a nota do aluno, sem considerar o volume de código, não foram suficientes para estabelecer uma relação conclusiva. Com a existência de outros determinantes que influenciam na pontuação, foi escolhido como candidato principal deste estudo o volume de código, que será discutido posteriormente. No mais, em essência, observa-se que alunos com notas mais altas tendem a cometer menos erros, como mostrado na Tabela 2.

Pearson	Spearman	Kendall
-0.11	-0.19	-0.18

Tabela 2. Coeficientes de correlação entre quantidade de erros e nota.

Alguns erros mostraram significância estatística considerável para provar seu impacto na nota. Por exemplo, um erro de Compilação pode reduzir em até 1.2 pontos a nota final do aluno. Erros de Abstração e Completude também apresentaram valores que reduzem substancialmente a nota, como mostrado na Tabela 3.

Erro	Dependent variable: *Nota*
Método	1.274 (1.250)
Abstração	-0.352*** (0.250)
Lei de Deméter	0.374 (0.810)
Encapsulamento	-0.050 (0.521)
Coesão	0.374 (0.811)
Compilação	-1.266*** (0.360)
Lógica	-0.007 (0.281)
Completude	-0.412*** (0.283)
Tratamento de Erros	0.659 (0.497)
Constant	7.485 (0.756)

Tabela 3. Modelo de Regressão Linear Múltipla para Tipos de Erros e Notas.

Apesar de ser perceptível o impacto dos erros na nota do aluno, se faz necessário outras análises para uma melhor compreensão deste trabalho. Sendo assim, como no estudo anterior de Leal [12], o volume de código será um fator adicional a ser considerado. A abordagem de agrupar a amostra em três grupos de volume de código, levando em conta a média das métricas selecionadas,

permitirá uma explicação mais robusta sobre a relação entre o desempenho dos alunos e os erros.

4.2 Como se caracterizam, dado o programa em que estão contidos e o programador que os cometeu, os diferentes tipos de erros?

Foi constatado que as variáveis com maior coeficiente absoluto são as referentes ao número de linhas, métodos e acoplamento. Isso significa que são responsáveis por uma maior dispersão dos dados e consequentemente viriam a dificultar a análise caso fossem consideradas, dado que podem ser variáveis redundantes.

	PC1	PC2	PC3	PC4	PC5	PC6
n_linhas	-0.96	-0.01	-0.16	-0.03	-0.23	-0.02
n_classes	0.27	0.01	-0.27	0.06	-0.92	0.01
n_metodos	0.1	-0.08	-0.89	-0.34	0.27	-0.01
n_atributos	0.02	-0.11	0.31	-0.91	-0.15	-0.2
complexidade_c	-0.01	0.5	0.03	-0.25	-0.02	0.83
acoplamento	-0.0	-0.85	0.06	0.01	-0.03	0.52

Tabela 4. PCA - Análise dos Componentes Principais.

A redundância é evidenciada no correlograma da Figura 4, onde as métricas referentes ao Número de Métodos, Complexidade Ciclomática e Acoplamento possuem altas correlações. Logo, as variáveis que restaram foram consideradas para a caracterização do volume de código.

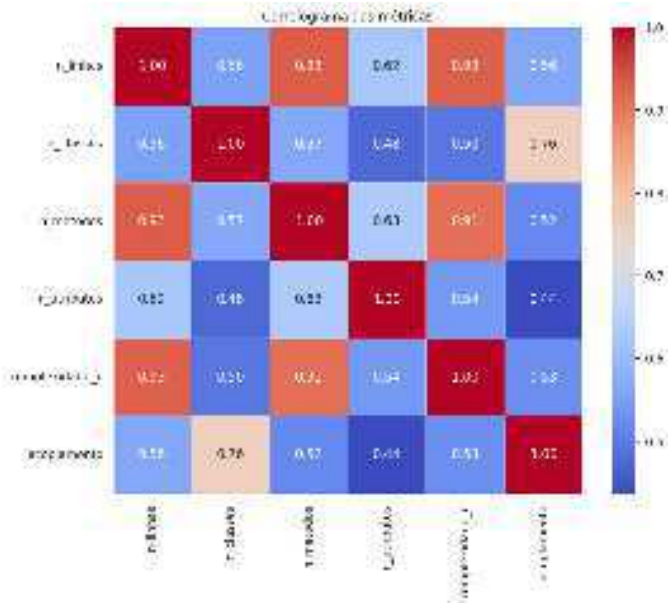

Figura 4. Correlograma com as métricas de código.

A eliminação das variáveis redundantes permitiu uma filtragem mais precisa de quais seriam consideradas para o estudo. Combinando este processo com a utilização do algoritmo de agrupamento *K-Means*, foi possível dividir os grupos e rotulá-los para uma análise minuciosa no impacto dos erros na nota do aluno.

Volume	m_linhas	m_classes	m_atributos
Baixo	102.80	5.47	10.27
Médio	180.57	7.14	13.46
Alto	248.72	7.28	14.50

Tabela 5. Grupos definidos e suas médias.

Foi observado que os 61 alunos da amostra se dividiram da seguinte forma:

- 28 alunos na categoria de volume de código “Baixo”;
- 18 alunos na categoria de código “Médio”;
- 15 alunos na categoria de código “Alto”.

A Figura 5 revela que o grupo de volume de código “Alto” teve uma menor concentração de erros em comparação aos outros dois grupos.

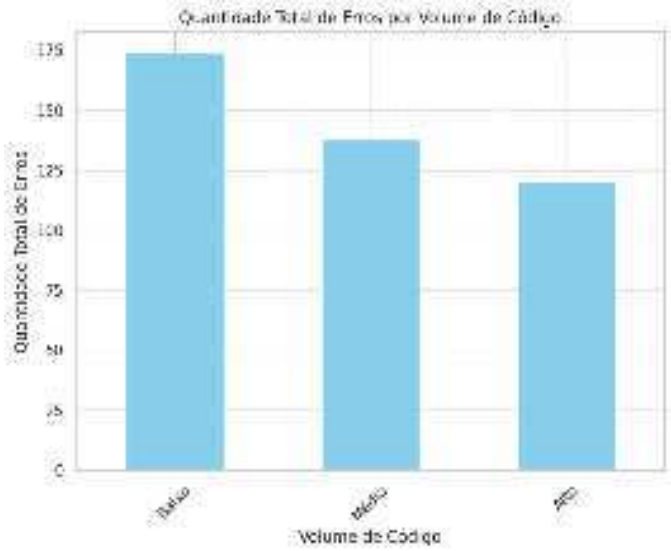

Figura 5. Quantidade de erros por volume de código.

A Tabela 2 na seção secundária 4.1 mostrou que as correlações entre o total de erros e a nota do aluno não levantaram números que sugerissem uma significância estatística relevante. Agora, adicionando o volume de código como um fator importante para o estudo, percebe-se que as correlações sofreram uma mudança considerável em seus valores, como consta na Tabela 6.

Volume	Pearson	Spearman	Kendall
Baixo	-0.12	-0.22	-0.20
Médio	-0.79	-0.75	-0.57
Alto	-0.88	-0.88	-0.76

Tabela 6. Coeficientes de correlação entre o total de erros e a nota de acordo com o volume de código.

Desta vez, é possível interpretar que, à medida que o volume de código aumenta, mais forte é a relação negativa entre o total de erros cometido pelos alunos e suas notas. É interessante notar que projetos com volume de código "Baixo" apresentam a correlação mais fraca entre os três grupos. Contudo, as correlações evidenciam a influência dos erros nas notas dos alunos.

4.3 Qual a frequência dos erros?

Foram considerados neste estudo nove tipos de erros, e sua identificação foi feita pela análise dos comentários dos professores nas notas dos alunos, bem como a observação da rubrica fornecida juntamente com a planilha de correção.

A Figura 6 mostra o total de ocorrências de cada erro na amostra estudada, independente do volume código.

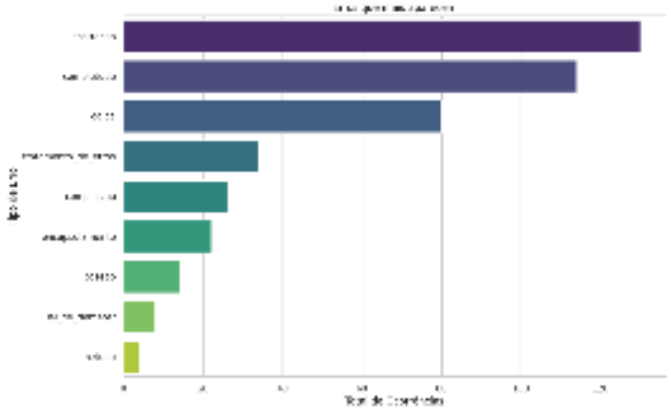

Figura 6. Ocorrências dos erros.

Erros de Abstração, Completude e Lógica, respectivamente, foram os mais comuns, ocupando as primeiras posições no gráfico de ocorrências. Isso foi algo observado também por Leal [12], embora erros de Lógica tenham liderado. Outros estudos focaram essencialmente em erros de sintaxe [5][6][10][11].

De maneira análoga, ao considerar o volume de código, o grupo de baixo volume liderou a categoria de erros de Abstração, mas teve menos erros de Lógica do que o grupo de médio volume. Como esperado, o grupo de baixo volume também apresentou a maior incidência nos erros de Completude. Isso é evidente pela média das métricas desse grupo: menor número de linhas, atributos e classes. Por outro lado, o grupo de alto volume registrou menos erros dentre os três mencionados.

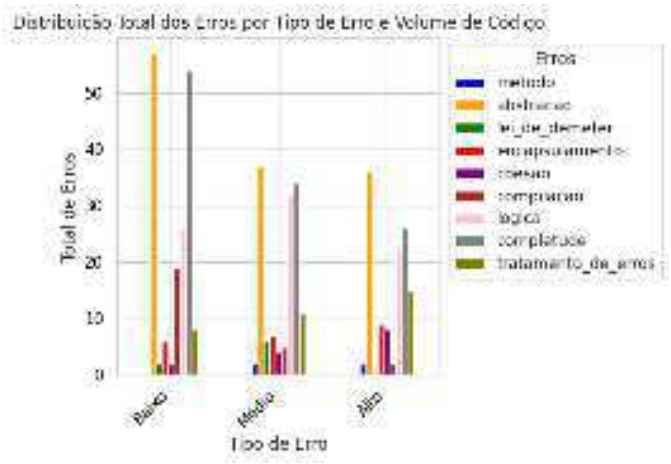

Figura 7. Ocorrências dos erros de acordo com o volume de código.

Erros relacionados à Lei de Deméter, Coesão, Encapsulamento e Método foram menos frequentes. Essa baixa ocorrência pode ser atribuída à variação nos comentários feitos pelos professores nas notas dos alunos, os quais não seguem um padrão definido e tampouco oferecem a mesma profundidade quanto ao que está sendo descrito. Programas com erros de Compilação ainda foram considerados, visto que existem cinco partes que compõem a avaliação. No entanto, tais erros tiveram maior concentração no grupo de baixo volume.

De acordo com o que foi mencionado anteriormente, estudos anteriores identificaram erros de Lógica como os mais frequentes

entre as ocorrências observadas. É crucial entender que a definição de um erro de Lógica, independente da linguagem de programação sendo utilizada, pode variar de uma pessoa para outra. Por exemplo, a concepção mais comumente difundida diz que erros de Lógica são cometidos pelo programador de forma que as saídas da aplicação não resultem no esperado, embora a sintaxe esteja correta [13]. Contudo, a diversidade nos níveis de aprendizado e experiência podem criar ramificações para esta definição [14], consequentemente tendo resultados diferentes em suas análises.

5. DISCUSSÕES

O estudo anterior proposto por Leal [12], considerando o semestre 2019.2, observou uma quantidade de 400 erros, independente do tipo, numa turma de 91 alunos. Apesar da disciplina de Programação II ter mantido sua metodologia de ensino ao longo dos anos, os resultados obtidos na pesquisa atual implicam na existência de outros fatores que influenciaram negativamente no desempenho dos alunos. Sendo a turma do período em estudo composta por 61 alunos, foram encontrados 432 erros, independente do tipo. Essa discrepância nos valores levanta a necessidade de uma investigação mais precisa no que pode estar contribuindo para este aumento de erros, mesmo sendo uma turma relativamente menor.

Quanto ao número de erros por aluno, os resultados foram parecidos. Em 2019.2 os alunos cometeram em sua maioria entre 3 e 6 erros, enquanto neste estudo a faixa predominante foi entre 4 e 6 erros por aluno.

Tratando-se do impacto direto dos erros na nota dos alunos, foi observado em ambos os estudos que erros relacionados à Abstração, Completude e Compilação exerceram a maior influência na redução da pontuação, dado os valores encontrados no Modelo de Regressão Linear Múltipla. No entanto, a seleção das variáveis para caracterizar os grupos foi divergente. Leal [12] optou por utilizar a métrica de Acoplamento, ao passo que aqui foi utilizado a métrica do Número de Classes.

Embora não explicitada a quantidade de alunos por grupo no estudo de 2019.2, seu método proposto de agrupamento baseado no volume de código contribuiu diretamente para os resultados aqui obtidos.

Um fator interessante observado nesta pesquisa é que, em contraste com amostras de pesquisas anteriores, nas quais o número de alunos que as constituíam era significativamente maior [5][10] e a distribuição de erros mais equilibrada, o grupo de baixo volume apresentou a maior quantidade de erros entre os três grupos. Esse resultado também difere do que foi observado por Leal [12], onde o grupo de médio volume liderou nesse aspecto. Teoricamente, essa disparidade era esperada, uma vez que esse grupo tinha a maior concentração de alunos; ainda assim, se tratando de uma amostra menor, esperava-se um número menor de erros do que o total de 174 constatados.

Apesar da disparidade encontrada, um número maior de erros indica também que houve uma maior codificação nas avaliações. Em teoria, pode-se dizer que é um ponto positivo, já que existiu a tentativa dos alunos em realizar o que estava sendo pedido na especificação do problema.

Erros de Abstração, Completude e Lógica tiveram maiores ocorrências, assim como no trabalho feito por Leal [12]. Embora não haja informações sobre o conhecimento dos estudantes de forma individual, isto é, se tiveram contato prévio com Java e Programação Orientada a Objetos, a predominância destes erros era esperada. Os conceitos deste paradigma são amplos e complexos [15], e julgando pela carga horária do semestre em estudo, onde os alunos tiveram um tempo limitado para realizar as atividades da disciplina, é plausível concluir que o aprendizado tenha sido comprometido, sendo uma influência negativa em suas notas.

6. CONCLUSÃO

Conforme apresentado neste trabalho, erros de design apresentam uma certa influência na nota do aluno. Uma análise superficial, sem considerar variáveis adicionais — neste caso, volume de código — não teria sido suficiente para obter resultados concisos.

Alunos com menor incidência de erros obtiveram notas maiores; aqueles com mais erros, notas menores. Contudo, foi também notado situações em que o aluno atingiu uma pontuação alta, apesar de ter cometido um número considerável de erros, bem como alunos que cometeram poucos erros mas alcançaram uma nota baixa. Esse último cenário pode ser explicado pela falta de complexidade nos códigos, visto que alunos do grupo de baixo volume tiveram menores pontuações. Portanto, menos código, menor nota, menos erros a serem avaliados — uma vez que alguns comentários dos professores resumiam-se a “não fez” ou “incompleto”.

Em resumo, erros influenciam substancialmente nas notas dos alunos, e quando considerado o volume de código, é possível perceber este detalhe com maior acurácia. Futuras pesquisas podem investigar outras variáveis como nível de habilidade e realizar uma inspeção individual quanto a experiência prévia do aluno com os conceitos do paradigma. Além disso, uma amostra mais robusta, seja em termos de número de alunos ou de atividades avaliativas, e.g. minitestes, laboratórios e provas, poderiam expandir o conjunto de soluções disponíveis para os docentes, oferecendo um melhor aproveitamento da disciplina para os alunos.

REFERÊNCIAS

[1] Morais, C. G. B., Mendes Neto, F. M., & Osório, A. J. M. (2020). Dificuldades e desafios do processo de aprendizagem de algoritmos e programação no ensino superior: uma revisão sistemática de literatura.

[2] Caton, S., Russell, S., & Becker, B. A. (2022). What Fails Once, Fails Again: Common Repeated Errors in Introductory Programming Automated Assessments.

[3] Langhout, C., & Aniche, M. (2021). Atoms of Confusion in Java.

[4] Qian, Y., & Lehman, J. D. (2019). An Investigation of High School Students’ Errors in Introductory Programming: A Data-Driven Approach.

[5] Jegede, P. O., Olajubu, E. A., Ejidokun, A. O., & Elesemoyo, I. A. (2019). Concept-based Analysis of Java Programming Errors Among Low, Average and High Achieving Novice Programmers.

[6] Hristova, M., Misra, A., Rutter, M., & Mercuri, R. (2014). Identifying and Correcting Java Programming Errors for Introductory Computer Science Students.

[7] Sivitanides, M., & White, G. (2005). Cognitive Differences Between Procedural Programming and Object Oriented Programming.

[8] Eluri, P., & David, D. (2023). Insights to Improve Computer Science Education by Enhancing the Java Language and Tools.

[9] Saidova, D. E. (2022). Analysis of the Problems of Teaching Object-Oriented Programming to Students.

[10] Cobb, M. J., & Carver, C. A. (2005). Identifying Top Java Errors for Novice Programmers.

[11] Brown, N. C. C., & Altadmri, A. (2014). Investigating novice programming mistakes: educator beliefs vs. student data.

[12] Leal, M. H. A. (2021). Entendendo os erros de Programação OO de estudantes principiantes.

[13] Samara, G. (2017). A Practical Approach for Detecting Logical Error in Object Oriented Environment.

[14] Umezawa, K. (2020). Analysis of Logic Errors Utilizing a Large Amount of File History During Programming Learning.

[15] Benaya, T. (2008). Understanding Object Oriented Programming Concepts in an Advanced Programming Course

---

Sobre o autor

João Pedro Silva de Melo é graduando em Ciência da Computação na Universidade Federal de Campina Grande.