Analisando a Qualidade do Código em Plataformas de Cursos Online Abertos e Massivos

Otávio Vinícius Rocha, Aline Brito, Cleiton Tavares, Laerte Xavier, Simone Assis

[1] Departamento de Engenharia de Software – Instituto de Ciências Exatas e Informática
Pontifícia Universidade Católica de Minas Gerais (PUC Minas)
Belo Horizonte – Brasil

otaviovgsr@gmail.com,alinebrito@pucminas.br,cleitontavares@pucminas.br

laertexavier@pucminas.br,simone@pucminas.br

Resumo. *Frequentemente, estudantes utilizam Cursos Online Abertos e Massivos para aprender novas tecnologias. Nessas plataformas, os educadores podem fornecer código-fonte dos projetos para que os alunos realizem exercícios práticos. Nesse contexto, a qualidade do código é um fator relevante, visto que problemas podem impactar centenas de alunos. Além disso, os alunos aprendem usando estes exemplos. Neste trabalho, investiga-se a qualidade do código-fonte destes cursos. Especificamente, analisam-se 352 projetos, envolvendo cinco linguagens de programação: Java, JavaScript, TypeScript, Python e Go. Detectou-se cerca de 8 mil problemas de código em mais de 11 mil arquivos. A maioria dos problemas refere-se a code smells de baixa gravidade. Além disso, não existe uma correlação significativa entre a popularidade e a qualidade do código. Em resumo, os resultados sugerem que a qualidade do código disponibilizado nestas plataformas é satisfatória para os estudantes. Por fim, conclui-se o trabalho prospectando futuras linhas de pesquisa e discutindo sobre a adoção de ferramentas de análise de código.*

1. Introdução

Com o avanço tecnológico do mundo moderno, observa-se uma demanda crescente por cursos de capacitação voltados para o desenvolvimento de *software* [Keuning et al. 2017, Gomes et al. 2021]. Neste contexto, plataformas de Cursos Online e Massivos (*Massive Open Online Courses*, MOOC) são populares. As mesmas oferecem cursos abertos, ofertados em formato assíncrono *online* e projetados para receber um número ilimitado de participantes, isto é, sem restrição de local ou tempo [Bey et al. 2018].

MOOCs vêm se tornando um fenômeno global que está transformando o ensino e o aprendizado com plataformas *online* como Coursera, Udemy, Udacity e Alura [Hyman 2012]. Por exemplo, em dezembro de 2021, segundo a *Class Central*,[1] 220 milhões de alunos relataram seu registro em um dos mais de 19,4 mil cursos dentre as 950 plataformas.

Como em qualquer sistema de *software*, o código mantido e disponibilizado pelos educadores em plataformas MOOC deve, idealmente, possuir boa qualidade. Em outras palavras, espera-se que os projetos funcionem corretamente, atendam aos requisitos de qualidade e sejam seguros, visto que são utilizados por milhares de programadores iniciantes. Entretanto, a maioria dos estudos preocupa-se principalmente com a qualidade do código produzido pelos

[1] https://www.classcentral.com/report/mooc-stats-2021

estudantes [Keuning et al. 2017, Gomes et al. 2021, Bezerra et al. 2022, Brito et al. 2022]. A qualidade do código mantido e elaborado pelos educadores não é um tema profundamente explorado. É importante reforçar que os exemplos disponibilizadas durante os cursos podem influenciar o processo de aprendizagem do estudante [Andrade Gomes et al. 2017].

Dessa forma, o objetivo geral deste trabalho é analisar a qualidade do código-fonte de educadores, que ministram cursos de programação para iniciantes em plataformas MOOCs. Assim, estuda-se neste artigo duas questões de pesquisa centrais : *(QP1) Qual é a frequência de problemas de código em plataformas MOOC? e (QP2) Qual é a gravidade dos problemas de código em plataformas MOOC?*

Para responder a estas questões de pesquisa, analisou-se o código de 352 cursos disponíveis em duas plataformas MOOC populares, Alura e Udacity. Os cursos referem-se a cenários distintos no contexto de desenvolvimento e manutenção de *software*. Além disso, a análise abrange cinco linguagens de programação populares: JavaScript, Java, TypeScript, Go e Python. Detectou-se aproximadamente 8 mil problemas relacionados à *code smells* (7.299, 92,93%), *bugs* (512, 6,51%), e vulnerabilidades (43, 0,40%). Além disso, apenas 10% dos cursos (37 ocorrências) não apresentem nenhum tipo de problema. Especificamente, os resultados mostram que a maioria dos problemas possui baixa gravidade, sugerindo que o código disponibilizado para os estudantes segue as boas práticas e recomendações de segurança. Entretanto, existe uma taxa significativa de *code smells*.

As próximas seções do artigo estão divididas da seguinte forma: a Seção 2 apresenta a fundamentação teórica e a Seção 3 mostra a metodologia adotada neste estudo. As Seções 4 e 5, apresentam e discutem os resultados obtidos no trabalho, respectivamente. A Seção 6 discute os trabalhos relacionados. A Seção 7 detalha as ameaças a validade e suas mitigações. Finalmente, a Seção 8 apresenta a conclusão e possíveis trabalhos futuros.

2. Fundamentação Teórica

Esta seção apresenta uma fundamentação teórica sobre: análise estática de código-fonte, métricas e *Massive Open Online Courses*.

2.1. Análise Estática de Código e Métricas

A análise estática envolve a inspeção do código sem a necessidade de executar o programa, sendo suportada por diversas ferramentas e objeto de estudo na literatura recente [Bibiano et al. 2019, Vassallo et al. 2018, Marcilio et al. 2019]. Normalmente, estas ferramentas geram relatórios e identificam violações e falhas, causadas, por exemplo, por más práticas de programação. Elas mostram desvios dos padrões de qualidade de códigos existentes, bem como uma estimativa aproximada da qualidade geral do sistema de *software* [Andrade Gomes et al. 2017].

Em geral, as métricas fornecidas pelas ferramentas de análise de código estático indicam os seguintes tipos de problemas:

- *Bugs:* são erros de codificação que podem levar a um comportamento inesperado em tempo de execução. É recomendado que qualquer *bug* encontrado seja resolvido imediatamente [Andrade Gomes et al. 2017];
- *Vulnerabilidade*: são violações de segurança que se originam em erros de programação, que podem levar a incidentes de segurança, comprometendo a confiabilidade do *software* [Gasiba et al. 2021];

- *Code Smells*: são estruturas no código que indicam violação de princípios fundamentais de *design* que impactam negativamente em sua qualidade [Sharma and Chhabra 2024, Bibiano et al. 2019].

2.2. *Massive Open Online Courses*

Desde 2012, *Massive Open Online Courses* (MOOCs) são considerados uma forma de educação disruptiva e transformadora, que tem crescido exponencialmente ao redor do mundo [Hyman 2012]. O termo foi criado em 2008 para descrever um curso *online* aberto a ser oferecido pela Universidade de Manitoba, no Canadá [Liyanagunawardena et al. 2013]. MOOCs podem variar muito em formato, destacando-se algumas características definidoras:

- *Massive*: são cursos projetados para a inscrição de um número ilimitado de participantes, já que, se o número de participantes aumentar, nenhum esforço adicional será necessário para conduzi-los [Mahajan et al. 2019];
- *Open*: o aspecto aberto dos MOOCs refere-se ao conceito de que geralmente são gratuitos e acessíveis a qualquer pessoa, não exigindo um processo formal de admissão [Blackmon and Major 2017];
- *Online*: os cursos são ministrados por meio de recursos *online* via internet [Mahajan et al. 2019];
- *Course*: um curso completo é oferecido, incluindo o plano de metas de aprendizagem, e o material didático é disponibilizado na plataforma com o conteúdo do curso. Em geral, a avaliação dos participantes é formativa,[2] feita por meio de questionários, podendo ser empregado exames somativos para fins de certificação [Blackmon and Major 2017].

3. Metodologia

3.1. Mineração de Repositórios em Plataformas MOOC

Nesta pesquisa, considera-se as linguagens Javascript, Java, Python, Typescript e Go, sendo essas encontradas entre as mais populares nos *rankings* do GitHub[3] e TIOBE [Lopes and Hora 2022].[4] Visto que os cursos visam capacitar alunos, considera-se os mesmos como cursos voltados para iniciantes no conteúdo ministrado. Os projetos foram extraídos de cursos provenientes de duas plataformas populares, descritas a seguir.

Alura: A Alura é uma das maiores plataformas MOOC de ensino de programação paga no Brasil,[5] possuindo em seu catálogo mais de 1.400 cursos voltados para a área digital, além de possuir mais de 70 mil alunos ativos. Para obtenção do código-fonte dos cursos na plataforma, buscaram-se aqueles: (i) cursos categorizados como programação; (ii) com o código final do projeto disponível para *download*; (iii) que possuam como linguagem principal as selecionadas anteriormente para o estudo, sendo em sua forma base ou *frameworks*.

Udacity: A Udacity conta com mais de 11 milhões de usuários em todo o mundo.[6] Para obtenção do código-fonte dos cursos na plataforma, buscaram-se repositórios que: (i) possuem código disponibilizado durante o curso ou *templates* de exercício disponibilizado para os alunos; (ii) possuem como linguagem principal Javascript, Java, Python, Typescript ou Go.

[2] https://eric.ed.gov/?id=ED123239
[3] https://octoverse.github.com/2022/top-programming-languages
[4] https://www.tiobe.com/tiobe-index
[5] https://www.alura.com.br
[6] https://www.udacity.com

3.2. Análise da Qualidade do Código

Para análise da qualidade dos projetos, utilizou-se as ferramenta SonarQube (SQ). A escolha da ferramenta deve-se ao fato de que a mesma é popular no contexto de análise estática de código, provendo também informações sobre violações e nível de gravidade dos problemas [Campbell and Papapetrou 2013, Vassallo et al. 2018, Marcilio et al. 2019].

As *issues* do SonarQube são classificadas em categorias, de acordo com sua gravidade:

- *Blocker*: problemas graves que impedem o funcionamento correto do *software* ou afetam diretamente a segurança do sistema;
- *Critical*: problemas que podem causar falhas graves no *software* ou afetar negativamente a segurança do sistema;
- *Major*: problemas que afetam a qualidade do código e podem levar a problemas operacionais, mas não são tão graves como os classificados como críticos ou bloqueadores;
- *Minor*: problemas menores de qualidade de código que geralmente não afetam a funcionalidade do *software*, mas podem ser corrigidos para melhorar a qualidade do código;
- *Info*: problemas de baixa gravidade, porém é importante corrigi-los, uma vez que eles podem se acumular ao longo do tempo e afetar negativamente a qualidade do código.

A ferramenta fornece também a classificação para cada um dos grupos de métricas. Sendo, de A a E, onde A é pontuação mais alta.[7] A definição de classificação apresentada pelo SonarQube está descrita na Tabela 1. Métricas de *manutenibilidade* baseiam-se na dívida técnica do projeto, representada pelo esforço (tempo) estimado para corrigir todos os *code smells* encontrados, a *confiabilidade* envolve o número *bugs* encontrados no código, enquanto as métricas de *segurança* estão relacionadas ao número de vulnerabilidades, ou seja, partes do código que possuem possíveis falhas de segurança. Dessa forma, neste trabalho, a qualidade dos cursos é avaliada considerando essas categorias de métricas detectadas pela ferramenta.

Tabela 1. Classificação de métricas por categoria (SonarQube)

Nota	Classificação de Confiabilidade	Classificação de Segurança	Classificação de Manutenibilidade
A	Nenhum *bug*	Nenhuma vulnerabilidade	Índice de Dívida Técnica $< 5\%$
B	Pelo menos um *bug (minor)*	Pelo menos uma vulnerabilidade *(minor)*	Índice de Dívida Técnica $< 10\%$
C	Pelo menos um *bug (major)*	Pelo menos uma vulnerabilidade (major)	Índice de Dívida Técnica $< 20\%$
D	Pelo menos um *bug (critical)*	Pelo menos uma vulnerabilidade*(critical)*	Índice de Dívida Técnica $< 50\%$
E	Pelo menos um *bug (blocker)*	Pelo menos uma vulnerabilidade (*blocker)*	Índice de Dívida Técnica $>50\%$

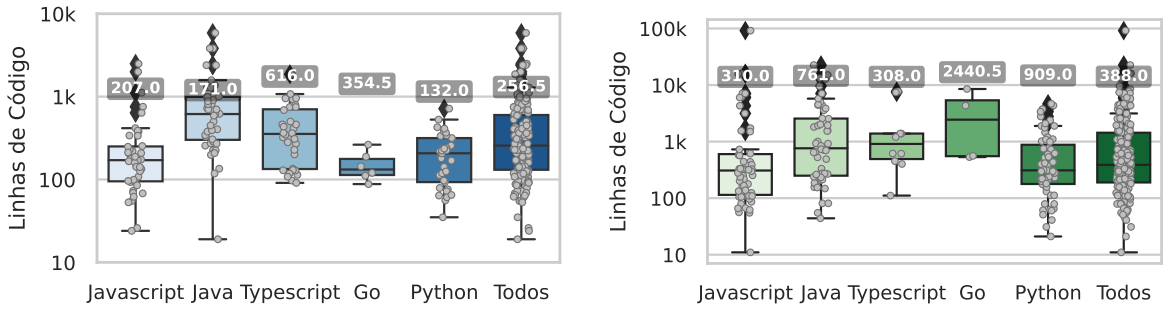
3.3. Caracterização do Conjunto de Dados

A caracterização dos dados coletados é realizada pela elaboração de gráficos *boxplot* na escala logarítmica, que permitem a visualização da distribuição dos valores obtidos (Figura 1). Em resumo, descreve-se o tamanho dos 352 cursos, provenientes das plataformas Alura e Udacity.

[7] https://docs.sonarqube.org/latest/user-guide/metric-definitions

Cursos Alura: Analisam-se 162 projetos da plataforma Alura (43 Javascript, 32 TypeScript, 48 Java, 33 Python e 6 em Go). A Figura 1(a) apresenta a distribuição do tamanho dos projetos, considerando o número de linhas de código por linguagem de programação. Avaliando todas as linguagens, a mediana é igual a 256, sendo que 75% dos cursos possuem até 600 linhas. O curso com o menor tamanho é "Maven: Gerenciamento de dependências e build" com 19 linhas, enquanto o curso com a maior quantidade de linhas é "Kafka: Fast deletate, evolução e cluster de brokers" com 5.866 linhas na linguagem Java. Ao todo, foram obtidos 2.523 arquivos das extensões dessas linguagens, sendo Java a linguagem com o maior número de arquivos (1.286), seguida por TypeScript (510), JavaScript (418), Python (282) e Go (27).

Cursos Udacity: Considerando todas as linguagens da plataforma Udacity, a mediana da quantidade de linhas de código é de 388, sendo que 75% dos cursos possuem projetos com até 1.435 linhas. O projeto de menor tamanho é "Hashing Code Exercise", com apenas 11 linhas de código, enquanto o maior é "Building High Conversion Web Form", com 91.651 linhas de código, ambos desenvolvidos utilizando Javascript. Além disso, observa-se que a mediana do número de linhas varia de acordo com a linguagem utilizada (Figura 1(b)): 308 em Typescript, 909 em Python, 761 em Java, 310 em Javascript e 2440 em Go. No total, analisa-se 8.807 arquivos. Existem 5.063 arquivos referentes à linguagem de programação Java, 1.910 arquivos JavaScript, 870 arquivos Python, 821 em TypeScript, e 143 arquivos em Go.



(a) *Alura* (b) *Udacity*

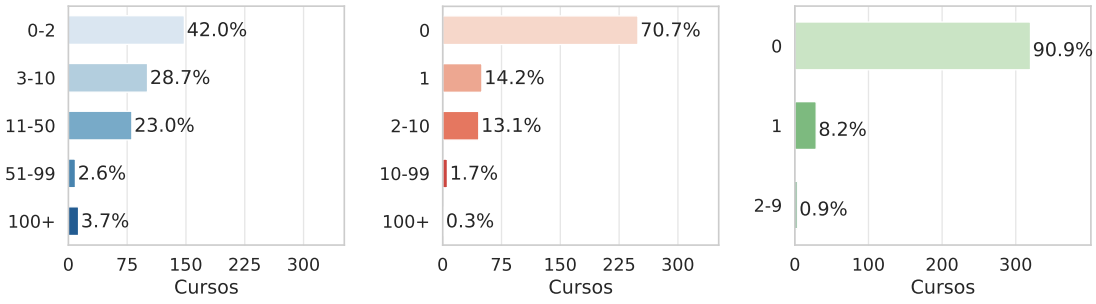
Figura 1. Distribuição da quantidade de linhas de código por linguagem

4. Resultados

4.1. (QP1) Qual é a frequência de problemas de código em plataformas MOOC?

Para responder a primeira questão de pesquisa, analisa-se a taxa de problemas de código por curso. A Figura 2 mostra a frequência de problemas de código por categoria: *code smells*, *bugs*, e vulnerabilidades.

Nota-se que a maioria dos cursos avaliados (42%) apresenta entre zero e dois *code smells*, enquanto, cerca de 29% dos cursos apresentaram entre três e dez. A maioria dos cursos avaliados (70,7% e 90,9%, respectivamente) não apresentou problemas de *bugs* e vulnerabilidades. No entanto, uma parcela significativa de cursos teve apenas uma ocorrência desses problemas: 50 cursos com um único *bug* (14,20%) e 29 cursos com uma única vulnerabilidades (8,23%).



(a) *Code Smells* (b) *Bugs* (c) *Vulnerabiliadde*

Figura 2. Quantidade de problemas de código por curso

Dentre os 352 projetos, existem 7.854 problemas de código, sendo 1.286 identificados na plataforma Alura (16,37%) e 6.568 na plataforma Udacity (83,62%). Nos próximos parágrafos, discute-se os resultados por plataforma.

Cursos Alura: A Tabela 2 exibe a frequência de problemas identificados nos cursos da plataforma Alura. Observa-se que os *code smells* foram detectados em 70,99% dos cursos (com 1.157 ocorrências de problemas na categoria). Destaca-se que somente os cursos ministrados nas linguagens Java e Typescript apresentaram vulnerabilidades. Assim, das 162 avaliações realizadas, apenas 18 cursos (11,11%) foram identificados com algum tipo de vulnerabilidade, sendo que nenhum deles apresentou mais de uma vulnerabilidade. A linguagem Go apresenta o maior percentual de cursos com *bugs* (87,5%), distribuídos em cinco projetos. O curso *Python Collections parte 2: conjuntos e dicionários* apresenta a maior taxa de ocorrências de *bugs* (23 ocorrências), correspondendo a 69,69% dos *bugs* encontrados na linguagem.

Tabela 2. Frequência de problemas de código (Alura)

Linguagem	N.º de cursos	Bugs		Vulnerabilidade		Code Smells	
		Ocorr.	%	Ocorr.	%	Ocorr.	%
Go	6	5	87,50	0	0,00	4	16,67
Java	48	37	41,67	16	33,33	412	79,17
Javascript	43	17	13,95	0	0,00	480	81,40
Python	33	33	21,21	0	0,00	193	57,58
Typescript	32	19	40,63	2	6,25	68	68,75
Total	162	111	31,48	18	11,11	1.157	70,99

O Listing 1 mostra um exemplo de problema de código, onde o trecho contém um *code smell* de gravidade *major*.[8] Essa regra procura identificar blocos de código que podem ser simplificados para melhorar a legibilidade e manutenibilidade.

```
1 calcular_regras(self):
2     col = self.pacman.coluna_intencao
3     lin = self.pacman.linha_intencao
4     if 0 <= col < 28 and 0 <= lin < 29:
5         if self.matriz[lin][col] != 2:
6             self.pacman.aceitar_movimento()
```

Listing 1. Exemplo de problema de código (Code Smell em Python)

[8] https://rules.sonarsource.com/python/type/Code\%20Smell/RSPEC-1066

Cursos Udacity: A Tabela 3 apresenta a frequência dos problemas encontrados na plataforma Udacity. A linguagem Go não apresenta ocorrências de *bugs* e vulnerabilidades. No entanto, todos os cursos da linguagem Go apresentaram problemas de *code smell* (279 ocorrências). Java e Javascript possuem um número significativo de ocorrências de *bugs*, 73 e 270, respectivamente, o que representa mais de 30% dos cursos avaliados em ambas as linguagens. Em Python, existem 2.725 ocorrências de *code smells* (95,24%).

Tabela 3. Frequência de problemas de código (Udacity)

Linguagem	N.º de cursos	Bugs		Vulnerabilidade		Code Smells	
		Ocorr.	%	Ocorr.	%	Ocorr.	%
Go	4	0	0,00	0	0,00	279	100,00
Java	55	73	34,55	13	21,82	998	72,73
Javascript	58	270	34,48	12	3,45	2.003	86,21
Python	63	55	15,87	0	0,00	2.725	95,24
Typescript	10	3	30,00	0	0,00	137	90,00
Total	190	401	27,37	25	7,37	6.142	85,79

4.2. (QP2) Qual é a gravidade dos problemas de código em plataformas MOOC?

Para responder a segunda questão de pesquisa, analisam-se as métricas de confiabilidade, segurança, e manutenibilidade, considerando a classificação de qualidade disponibilizada pela ferramenta SonarQube.

Cursos Alura. A Tabela 4 mostra a classificação dos problemas na plataforma Alura. Observa-se que a maioria dos cursos na linguagem Go possuem a nota C em Confiabilidade (83.3%, 5 cursos). Assim como Python, todos os cursos em Go receberam boas classificações em relação métricas de segurança e manutenibilidade (Nota A). Javascript é a linguagem com maior porcentagem de notas A (37 cursos, 86%). Por fim, vale destacar que, embora cada linguagem tenha suas características próprias, Javascript e Python apresentaram o maior percentual de cursos com boas notas nas três métricas analisadas.

Tabela 4. Classificação de qualidade por tipo de problema de código (Alura)

Linguagem	Bugs Confiabilidade (%)					Vulnerabilidades Segurança (%)					Code Smell Manutenabilidade (%)				
	A	**B**	**C**	**D**	**E**	**A**	**B**	**C**	**D**	**E**	**A**	**B**	**C**	**D**	**E**
Go	16,7	0,0	83,3	0,0	0,0	100	0,0	0,0	0,0	0,0	100	0,0	0,0	0,0	0,0
Java	58,3	8,3	25,0	4,2	4,2	66,7	12,5	4,2	10,4	6,3	72,9	20,8	4,2	2,1	0,0
Javascript	86,0	0,0	9,3	4,7	0,0	100	0,0	0,0	0,0	0,0	93,0	7,0	0,0	0,0	0,0
Python	78,8	0,0	21,2	0,0	0,0	100	0,0	0,0	0,0	0,0	100	0,0	0,0	0,0	0,0
Typescript	56,3	6,3	31,3	0,0	6,3	93,8	0,0	0,0	0,0	6,3	100	0,0	0,0	0,0	0,0

Cursos Udacity. A Tabela 5 apresenta a classificação dos problemas de código na plataforma Udacity, variando de A a E, onde notas A indicam uma boa classificação. Pode-se observar que a maioria dos cursos em Go e Python possuem classificação A. Em Typescript, 20% de seus cursos tiveram uma nota intermediária nas métrica de confiabilidade. Assim como na plataforma Alura, pela Udacity a linguagem Java é a que possui mais pontos fracos, apresentando notas mais baixas em todas as três métricas avaliadas.

Tabela 5. Classificação de Qualidade por tipo de problema de código (Udacity)

Linguagem	Bugs Confiabilidade (%)					Vulnerabilidades Segurança (%)					Code Smell Manutenabilidade (%)				
	A	**B**	**C**	**D**	**E**	**A**	**B**	**C**	**D**	**E**	**A**	**B**	**C**	**D**	**E**
Go	100	0,0	0,0	0,0	0,0	100	0,0	0,0	0,0	0,0	100	0,0	0,0	0,0	0,0
Java	65,5	7,3	18,2	3,6	5,5	78,2	1,8	1,8	12,7	5,5	61,8	14,5	9,1	7,3	7,3
Javascript	72,4	0,0	20,7	1,7	5,2	94,8	1,7	0,0	0,0	3,4	98,3	1,7	0,0	0,0	0,0
Python	88,9	0,0	9,5	0,0	1,6	100,0	0,0	0,0	0,0	0,0	98,4	1,6	0,0	0,0	0,0
Typescript	80,0	0,0	20,0	0,0	0,0	100,0	0,0	0,0	0,0	0,0	100	0,0	0,0	0,0	0,0

Por fim, analisa-se a correlação entre a popularidade e a quantidade de problemas de código detectados nos cursos. A análise estatística concentra-se exclusivamente na plataforma Alura, visto que a mesma disponibiliza informações sobre o número de alunos matriculados, mesmo atributo utilizado em estudos anteriores [Qaralleh and Darabkh 2015]. As plataformas não são comparadas, visto que o conjunto de dados possui tamanhos distintos e não inclui informações de alunos matriculados em cursos da Udacity. Especificamente, obtém-se $rho = 0{,}2503$ e $p - value = 0{,}0013$, sugerindo uma correlação insignificante entre as variáveis [Borges and Valente 2018, Hinkle et al. 2003].

5. Discussão dos Resultados

Nesta seção são apresentas discussões geradas pelas questões respondidas neste estudo.

Implicações para autores de cursos em plataformas MOOC. Dentre os 352 cursos analisados, apenas 37 deles não apresentaram nenhum tipo de problema nas três categorias (*bugs*, *code smells* e vulnerabilidades). Identificaram-se mais problemas de código relacionados com *code smells*. De fato, *code smells* são problemas recorrentes e menos graves [Pereira et al. 2018], sendo alvo de pesquisas recentes em outros contextos [Bibiano et al. 2019]. Embora um grande número de cursos apresente problemas de código, é importante destacar que a taxa de problemas por curso é baixa. Por exemplo, 51,98% dos cursos possuem até 5 problemas de código. Entretanto, os resultados sugerem que os autores de cursos podem melhorar a qualidade dos projetos, fazendo uso, por exemplo, de ferramentas de análise da qualidade do código-fonte, como *SonarQube*.

Adicionalmente, mais de 50% dos cursos em todas as linguagens receberam a classificação A em todas as categorias, sugerindo que os problemas encontrados possuem uma baixa gravidade. Por exemplo, em Python, todos os cursos receberam classificação A em relação a problemas de segurança. Entretanto, observou-se a ocorrência de problemas de níveis intermediários, por exemplo, em cursos Java. É importante ressaltar que o impacto dos problemas pode variar de acordo com o contexto e o uso da linguagem de programação. Falhas de vulnerabilidade, por exemplo, podem causar danos significativos [Iannone et al. 2023]. Dessa forma, educadores podem se beneficiar também de ferramentas de análise de qualidade de código para mitigar problemas graves, que podem impactar os estudantes.

Implicações para desenvolvedores de ferramentas. Os sistemas Java possuem problemas de código em todas as categorias nas plataformas Alura e Udacity (A até E). Este resultado pode estar relacionado com a sua popularidade e nível de maturidade, visto que é uma linguagem reconhecida e amplamente adotada. SonarQube, por exemplo, define 656 regras para a

linguagem Java, enquanto apenas 38 regras são definidas para a linguagem Go. Dessa forma, este estudo também reforça a necessidade de atualização de ferramentas e métricas voltadas para tecnologias emergentes [Ferreira and Valente 2023, Vegi and Valente 2022].

6. Trabalhos Relacionados

Os educadores exercem um papel importante na preparação dos materiais em cursos. No contexto de cursos voltados para desenvolvimento de *software*, diversas investigações visam compreender o conhecimento técnico de educadores [Kirk et al. 2020, Boutnaru and Hershkovitz 2015, Gomes et al. 2021, Andrade Gomes et al. 2017], cujas abordagens e técnicas utilizadas inspiraram o presente trabalho. Existem também ferramentas que oferecem *feedback* sobre a qualidade do código, podendo ser integradas com plataformas MOOC [Birillo et al. 2022]. No lado dos estudantes, soluções têm sido propostas para melhorar suas habilidades de programação [Andrade Gomes et al. 2017]. Entretanto, esses estudos não se concentram na qualidade e segurança do código mantido pelos educadores em plataformas MOOC.

Neste contexto, Kirk et al. (2020) avaliaram o conhecimento técnico dos professores em relação às boas práticas de qualidade de código. Oito professores foram entrevistados e solicitados a avaliar a qualidade de código em exemplos de exercícios de alunos. Como resultado, observou-se que alguns professores fizeram correções parciais ou não possuíam conhecimento técnico para avaliar certos aspectos do código. No entanto, o trabalho não faz uso de ferramentas de análise estática de código. No presente estudo, utiliza-se a ferramenta SonarQube para uma análise em larga escala sobre qualidade do código disponibilizado por educadores em cursos.

Andrade Gomes et al. (2017) apresentam a ferramenta SMaRT (*Software Maintenance, Report and Tracker*), cujo objetivo é apoiar os alunos na melhoria do código e, consequentemente, de suas habilidades de programação. Assim como neste trabalho, a ferramenta também faz uso de relatórios fornecidos pelo SonarQube. Por meio de um estudo piloto, os autores observaram o potencial da ferramanta para auxiliar os alunos na gestão da qualidade do código.

7. Ameaças à Validade

Primeiro, como usual em estudos empíricos em Engenharia de *Software*, os resultados não podem ser generalizados para outros cenários. Entretanto, o trabalho inclui 352 cursos em duas plataformas MOOC populares, abrangendo cinco linguagens de programação distintas. Especificamente, analisa-se 11.315 arquivos.

Adicionalmente, neste estudo, utilizou-se a configuração padrão da ferramenta SonarQube, que inclui um conjunto predefinido de regras. Dessa forma, a análise da qualidade dos projetos é baseada nas métricas reportadas pela ferramenta, isto é, ocorrência de *bug*, *code smells* e vulnerabilidades. As regras que definem as métricas estão disponíveis na documentação oficial da ferramenta e também são fornecidas no pacote de replicação deste estudo.[9] Logo, é possível analisar os tipos de problemas detectados. Por exemplo, para Java, a ferramenta reporta cerca de 400 tipos de *code smells*. Visando mitigar falsos positivos, removeu-se regras que não se aplicam ao contexto deste trabalho. Por exemplo, foram removidas *tags* relacionadas a uma versão específica da linguagem de programação, como problemas

[9] `https://rules.sonarsource.com`

relacionados com a *tag* $java17$.[10] Essas *issues* específicas de versões representam apenas uma pequena fração dos resultados.

Por fim, adota-se a terminologia utilizada pela ferramenta para a classificação de *code smells* e *bugs*. Entretanto, a ferramenta é popular no contexto de análise estática de código [Vassallo et al. 2018, Marcilio et al. 2019]. Estudos futuros podem considerar outras ferramentas populares e nomenclaturas distintas para classificação de problemas de código [Bibiano et al. 2019, Sharma and Chhabra 2024, Lenarduzzi et al. 2020], bem como aprofundamento sobre as regras frequentemente detectadas pela ferramenta SonarQube.

Existe também a possibilidade de que alguns arquivos do curso sejam dependências externas. Para atenuar esta ameaça, pacotes usualmente utilizados para versionar bibliotecas externas foram removidos, como as pastas $node\_modules$, lib, $site\_packages$ e $vendor$. Além disso, consideraram-se apenas os arquivos nas extensões específicas de linguagem de programação, como $.js$, $.py$, $.ts$, $.go$ e $.java$.

8. Conclusão

Este estudo teve como objetivo investigar a presença de problemas de código em cursos de programação oferecidos por plataformas de ensino *online*. Ao todo, analisaram-se aproximadamente 400 cursos das plataformas Alura e Udacity, totalizando 11.315 arquivos. Especificamente, o trabalho concentrou-se em cursos baseados nas linguagens de programação Java, Javascript, Typescript, Python e Go. A análise de qualidade do código foi realizada através da ferramenta SonarQube. Apresenta-se a seguir os principais resultados:

- A maioria dos problemas de código referem-se à *code smells*. Especificamente, detectou-se aproximadamente 8 mil ocorrências: 7.299 *code smells* (92,93%), 512 *bugs* (6,51%), e 43 vulnerabilidades (0,40%).
- Apenas 37 cursos (10,51%) não apresentaram nenhum tipo de problema, isto é, sem *bugs*, *code smells* e vulnerabilidades.
- Em relação às linguagens de programação, detectou-se a maioria dos problemas em cursos Python (38,27%), seguidos por Javascript (35,42%) e Java (19,72%).
- A maioria dos problemas de código encontrados não são graves, mostrando que a qualidade do código disponibilizado nessas plataformas pode ser considerada satisfatória para os estudantes que utilizam os cursos *online* para aprendizado. Por exemplo, no caso de JavaScript, apenas 5.94% (6 ocorrências) dos cursos possuem a métrica de confiabilidade as classificações D ou E.

Como trabalhos futuros, sugere-se realizar estudos qualitativos, considerando a perspectiva dos educadores sobre o uso de ferramentas de análise estática de código para garantir a qualidade dos cursos. Também é possível utilizar outras ferramentas de análise da qualidade de código, avaliando, por exemplo, categorias distintas de problemas. Além disso, é possível aprofundar a análise realizada, detalhando os tipos de *code smells*, vulnerabilidades e *bugs* detectados pela ferramenta SonarQube.

O pacote de replicação deste estudo encontra-se disponível em: `https://github.com/alinebrito/vem2024-replication-package-qualidade-mooc`

[10] `https://rules.sonarsource.com/java/tag/java17`

Referências

Andrade Gomes, P. H., Garcia, R. E., Spadon, G., Eler, D. M., Olivete, C., and Correia, R. C. M. (2017). Teaching software quality via source code inspection tool. In *2017 IEEE Frontiers in Education Conference (FIE)*, pages 1–8. Ieee.

Bey, A., Jermann, P., and Dillenbourg, P. (2018). A comparison between two automatic assessment approaches for programming: An empirical study on moocs. *Journal of Educational Technology & Society*, 21(2):259–272.

Bezerra, C., Damasceno, H., and Teixeira, J. (2022). Perceptions and difficulties of software engineering students in code smells refactoring. In *10th Workshop de Visualização, Evolução e Manutenção de Software (VEM)*, pages 41–45.

Bibiano, A. C., Garcia, E. F. D. O. A., Kalinowski, M., Fonseca, B., Oliveira, R., Oliveira, A., and Cedrim, D. (2019). A quantitative study on characteristics and effect of batch refactoring on code smells. In *13th International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 1–11.

Birillo, A., Vlasov, I., Burylov, A., Selishchev, V., Goncharov, A., Tikhomirova, E., Vyahhi, N., and Bryksin, T. (2022). Hyperstyle: A tool for assessing the code quality of solutions to programming assignments. In *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education V. 1*, pages 307–313.

Blackmon, S. and Major, C. (2017). Wherefore art thou mooc: Defining massive open online courses. *Online Learning Journal*, 21(4).

Borges, H. and Valente, M. T. (2018). What's in a github star? understanding repository starring practices in a social coding platform. *Journal of Systems and Software*, 146:112–129.

Boutnaru, S. and Hershkovitz, A. (2015). Software quality and security in teachers' and students' codes when learning a new programming language. *Interdisciplinary Journal of E-Skills and Lifelong Learning*, 11:123–148.

Brito, A., Hora, A., and Valente, M. T. (2022). Understanding refactoring tasks over time: A study using refactoring graphs. In *25th Ibero-American Conference on Software Engineering (CIbSE)*, pages 1–15.

Campbell, G. A. and Papapetrou, P. P. (2013). *SonarQube in action*. Manning Publications Co.

Ferreira, F. and Valente, M. T. (2023). Detecting code smells in react-based web apps. *Information and Software Technology*, 155:107111.

Gasiba, T. E., Hodzic, S., Lechner, U., and Pinto-Albuquerque, M. (2021). Raising security awareness using cybersecurity challenges in embedded programming courses. In *2021 International Conference on Code Quality (ICCQ)*, pages 79–92. IEEE.

Gomes, P. H., Garcia, R. E., Eler, D. M., Correia, R. C., and Junior, C. O. (2021). Software quality as a subsidy for teaching programming. In *2021 IEEE Frontiers in Education Conference (FIE)*, pages 1–9. IEEE.

Hinkle, D. E., Wiersma, W., and Jurs, S. G. (2003). *Applied statistics for the behavioral sciences*, volume 663. Houghton Mifflin college division.

Hyman, P. (2012). In the year of disruptive education. *Communications of the ACM*, 55(12):20–22.

Iannone, E., Guadagni, R., Ferrucci, F., De Lucia, A., and Palomba, F. (2023). The secret life of software vulnerabilities: A large-scale empirical study. *IEEE Transactions on Software Engineering*, 49(1):44–63.

Keuning, H., Heeren, B., and Jeuring, J. (2017). Code quality issues in student programs. In *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education*, pages 110–115.

Kirk, D., Tempero, E., Luxton-Reilly, A., and Crow, T. (2020). High school teachers' understanding of code style. In *Koli Calling'20: Proceedings of the 20th Koli Calling International Conference on Computing Education Research*, pages 1–10.

Lenarduzzi, V., Lomio, F., Huttunen, H., and Taibi, D. (2020). Are sonarqube rules inducing bugs? In *27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 501–511.

Liyanagunawardena, T. R., Adams, A. A., and Williams, S. A. (2013). Moocs: A systematic study of the published literature 2008-2012. *International Review of Research in Open and Distributed Learning*, 14(3):202–227.

Lopes, M. and Hora, A. (2022). How and why we end up with complex methods: a multi-language study. *Empirical Software Engineering*, 27(5):115.

Mahajan, R., Gupta, P., and Singh, T. (2019). Massive open online courses: concept and implications. *Indian pediatrics*, 56(6):489–495.

Marcilio, D., Bonifácio, R., Monteiro, E., Canedo, E., Luz, W., and Pinto, G. (2019). Are static analysis violations really fixed? a closer look at realistic usage of sonarqube. In *2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC)*, pages 209–219.

Pereira, R., Henriques, P. R., and Vieira, M. (2018). The effects of code smells on software maintainability: A replication study. *Journal of Software: Evolution and Process*, 30(1):e1941.

Qaralleh, E. A. and Darabkh, K. A. (2015). A new method for teaching microprocessors course using emulation. *Computer Applications in Engineering Education*, 23(3):455–463.

Sharma, K. and Chhabra, J. K. (2024). An empirical evaluation of design smells and code smells over multiple versions of software evolution. In *15th International Conference on Computing, Communications, and Cyber-Security*, pages 961–973. Springer Nature Singapore.

Vassallo, C., Panichella, S., Palomba, F., Proksch, S., Zaidman, A., and Gall, H. C. (2018). Context is king: The developer perspective on the usage of static analysis tools. In *25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 38–49.

Vegi, L. F. d. M. and Valente, M. T. (2022). Code smells in elixir: early results from a grey literature review. In *30th IEEE/ACM International Conference on Program Comprehension (ICPC)*, pages 580–584. Association for Computing Machinery.