CBIE 2023
CONGRESSO BRASILEIRO DE INFORMÁTICA NA EDUCAÇÃO
Uma escola para o futuro: Tecnologia e conectividade a serviço da educação

Investigando o Uso de Testes para Apoiar a Resolução de Problemas de Programação

André Almeida[1], Eliane Araújo[1], Jorge Figueiredo[1]

[1]Universidade Federal de Campina Grande, Campina Grande, PB - Brasil

andrealmeida@copin.ufcg.edu.br, {eliane,abrantes}@computacao.ufcg.edu.br

Abstract. *Introductory programming courses use exercises to build skills and assess student performance. However, before developing the solution, students need to understand the specifications. In this work, we evaluate a test-based strategy to clarify the specification of problems and improve their resolution. We created the 'Oracle', which allows students to interact with reference solutions and compare it to other problem solving methods. In the empirical study carried out, we observed a significant improvement in student performance, reducing the time to find the correct solution in 65% of cases and the number of submissions to the system in 68% of cases, until the correct solution.*

Resumo. *Os cursos introdutórios de programação utilizam exercícios para desenvolver habilidades e avaliar o desempenho dos alunos. Entretanto, antes de desenvolver a solução, os alunos precisam compreender as especificações. Neste trabalho, avaliamos uma estratégia baseada em testes para esclarecer a especificação dos problemas e melhorar sua resolução. Criamos o 'Oráculo', que permite que os alunos interajam com soluções de referência e o comparamos com outros métodos de resolução de problemas. No estudo empírico realizado, observamos uma melhora significativa no desempenho dos alunos, reduzindo o tempo para encontrar a solução correta em 65% dos casos e o número de submissões ao sistema em 68% dos casos, até a solução correta.*

1. Introdução

No ensino de programação para iniciantes, a habilidade de refletir sobre problemas e desenvolver programas para alcançar soluções é tida como um dos requisitos principais para o sucesso [McCracken et al. 2001]. Os cursos introdutórios enfatizam o desenvolvimento desta habilidade através da resolução de diversos exercícios relacionados aos conceitos apresentados, os quais atuam também como forma de mensurar o desempenho dos alunos. No entanto, antes mesmo de produzir o programa, se faz necessário que os alunos compreendam a especificação do problema, ou seja, entendam do que se trata e quais são os requisitos necessários para desenvolver a solução.

De maneira geral, resolver problemas de programação envolve quatro etapas, com algumas variações, mas baseadas no *framework* de Polya [Polya and Conway 2004]. Primeiro, o programador precisa entender o problema (1) e desenvolver um plano para encontrar a solução deste (2). Depois, o programador precisa traduzir a solução encontrada para uma linguagem que o computador seja capaz de entender e executar (3); e por fim, avaliar sua solução por meio de testes e refatorá-la caso seja necessário (4). Portanto, percebe-se que o esforço de entender o problema de forma correta é algo crucial para o

desenvolvimento eficaz de programas. De modo ortogonal, é necessário também garantir a qualidade do software criado ao longo deste processo [Almeida et al. 2020].

Em estudos sobre dificuldades enfrentadas por programadores iniciantes, um argumento unânime é o de que aprender a programar não é uma tarefa fácil. Os alunos precisam lidar com vários obstáculos devido a complexidade do assunto, uma vez que é necessário entender corretamente conceitos que não estão diretamente associados ao seu cotidiano [Lahtinen et al. 2005, Garner et al. 2005]; e compreender a sintaxe das linguagens de programação [Ala-Mutka 2004, Kadar et al. 2021], por exemplo. Programar é um processo que se inicia desde a construção dos modelos mentais à respeito do programa, ou mais especificamente da representação mental de um problema.

Diante do exposto, esta pesquisa enfatiza a primeira etapa do processo de programação, quando os alunos devem tentar identificar o que é "o desconhecido". Através de uma estratégia baseada em testes, os alunos devem tentar reformular o problema em diferentes perspectivas. Quais são as entradas possíveis e suas respectivas saídas? Em outras palavras, os alunos devem ser capazes de formular novos pares de exemplos de entrada / saída (ou testes) para executar sobre uma solução de referência, antes de codificar a solução. Esta possibilidade foi viabilizada através do desenvolvimento de uma ferramenta (definida como "oráculo") e avaliada através de um estudo comparativo com outras estratégias de resolução de problemas. Para guiar este processo de avaliação, definimos as seguintes questões de pesquisa e as respectivas hipóteses.

- QP1: Os alunos que tentam compreender as especificações dos problemas utilizando o oráculo, criando e executando testes, produzem programas mais funcionalmente corretos?
  - H1.0: Os alunos que resolvem as questões utilizando o oráculo como apoio apresentam taxa de corretude funcional menor em suas soluções.
  - H1.1: Os alunos que resolvem as questões utilizando o oráculo como apoio apresentam taxa de corretude funcional maior em suas soluções.
- QP2: Os alunos que tentam compreender as especificações dos problemas utilizando o oráculo, criando e executando testes, convergem para a solução correta mais rapidamente?
  - H2.0: Os alunos que resolvem as questões utilizando o oráculo como apoio apresentam tempo de resolução e número de submissões maiores.
  - H2.1: Os alunos que resolvem as questões utilizando o oráculo como apoio apresentam tempo de resolução e número de submissões menores.

As principais contribuições deste trabalho são o Oráculo, como ferramenta para que o aluno interaja através de entradas e saídas para verificar seu entendimento dos enunciados de programação; e seu processo avaliativo, através de variáveis associadas ao desempenho do aluno.

2. Trabalhos Relacionados

Neto et al. [Neto et al. 2013] apresentam o método de ensino de Programação e Teste Orientado a Problemas (POPT - *Problem-Oriented Programming and Testing*) para cursos de Introdução à Programação, com o objetivo de melhorar as habilidades dos alunos novatos em testes e em lidar com problemas mal definidos (aqueles que não apresentam um objetivo claro, dicas de resolução ou a saída esperada). Suportado por uma ferramenta

chamada *TestBoot*, que utiliza planilha eletrônica para permitir que alunos iniciantes definam casos de teste de entrada-saída simples no estilo de tabela.

Basu et al. [Basu et al. 2015] apresentam um método aplicado no contexto de MOOCs (*Massive Open Online Courses*). Os autores propõem uma ferramenta, o Sistema OK, que realiza a avaliação automática de exercícios de programação com base em suítes de testes criadas para verificar as soluções fornecidas nos mais diversos cenários. No entanto, além desta proposta, os autores incluem um recurso para que os alunos busquem validar o entendimento dos exercícios. Através de um processo interativo, o aluno precisa lidar com algumas questões de respostas curtas sobre o potencial comportamento que o código deve esboçar; e cada uma dessas questões corresponde a um caso de teste "bloqueado". Respondendo corretamente a uma dessas questões, o aluno pode desbloquear o caso de teste e aplicar em sua solução antes de realizar sua submissão.

A estratégia de Denny et al. [Denny et al. 2019] apresenta forte embasamento no *Test Driven Development* [Beck 2003] e na inclusão do método no processo de desenvolvimento dos alunos, pois um dos benefícios do *Test Driven Development* é impor ao programador a pensar em casos de entrada especiais que podem ocorrer. No entanto, solicitar que os alunos desenvolvam os casos de teste primeiro também pode ser usado para ajudá-los a entender melhor o problema determinado. Uma das ferramentas de avaliação automatizada mais populares, o WebCAT [Edwards and Perez-Quinones 2008], foi criada justamente com o intuito de integrar o TDD no contexto dos alunos.

Wrenn e Krishnamurthi [Wrenn and Krishnamurthi 2019] propõem uma ferramenta, o *Examplar*, que fornece aos alunos um *feedback* instantâneo sobre se eles exploraram correta e completamente um problema, independentemente de seu progresso de implementação. Nesta abordagem os exemplos de entrada-saída, como casos de teste, podem ser articulados como afirmações do comportamento de entrada-saída das funções, ou seja, os alunos podem fornecer tanto a entrada quanto a saída que esperam do(s) problema(s) em questão. Com esta estratégia, os autores objetivam fornecer uma maneira de verificar a compreensão dos problemas e prevenir a implementação sem o conhecimento suficiente das particularidades dos mesmos.

A análise do estado da arte em relação aos trabalhos que denotam aproximação com a estratégia investigada por este trabalho revela que a inclusão de testes na etapa de compreensão do problema é um objeto de estudo com diversas variações. De recursos mais simples como planilhas, até ferramentas de automatização de *feedback*, a atividade de escrever testes capazes de capturar comportamentos que devem ser representados em código se mostra deveras benéfica quando comparada apenas a aplicação de testes de caixa preta que pouco agregam na construção do modelo mental da solução programática do problema.

O trabalho de Wrenn e Krishnamurthi [Wrenn and Krishnamurthi 2019], dentre os que foram apreciados, é o que mais se aproxima de nossa estratégia. No *Examplar*, os estudantes trabalham com a ideia de asserções para introduzir os exemplos de entrada e saída. No Oráculo, abordagem que propomos, o aluno se aproxima mais do contexto formal de testes, refletindo sobre os enunciados dos problemas e desenvolvendo soluções baseadas nos testes de entrada e saída que executa enquanto desenvolve suas soluções. Além disso, também se torna pertinente um estudo comparativo com as abordagens mais

cotidianas de resolução de problemas de programação, sendo elas o TDD e a forma tradicional de resolução de exercícios de programação (a qual nos referimos por “Usual”).

3. Metodologia

Com o objetivo de avaliar a estratégia que é o foco deste trabalho, idealizamos a comparação deste com outros métodos de resolução de exercícios de programação. Sendo assim, definimos as seguintes etapas para a realização do estudo.

3.1. Design do estudo

Definimos como público alvo os alunos de disciplinas introdutórias de programação da UFCG, das turmas de Engenharia Elétrica e Ciência da Computação. Para nivelar a turma em termos de conteúdo apresentado durante as aulas, definimos como conhecimento mínimo as estruturas de repetição e os vetores unidimensionais em Python.

Para cada exercício proposto, uma estratégia de resolução seria aplicada: (1) **usual**, na qual geralmente o aluno passa para a codificação após ler o enunciado; (2) **TDD**, no qual o aluno escreve um caso de teste, para em seguida produzir o código que pode ser validado pelo teste desenvolvido; e (3) **interação com o oráculo**, no qual o aluno deve criar testes de entrada/saída e executá-los sobre uma solução de referência, antes de codificar (ver Figura 1).

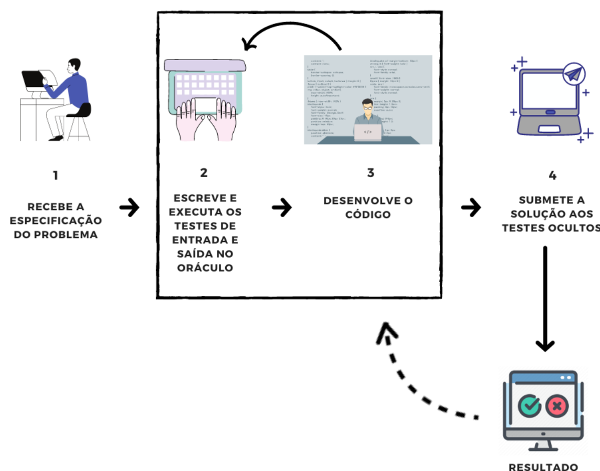

Figura 1. Processo de resolução com a inclusão do oráculo.

Vislumbramos a realização do estudo de maneira presencial e em dois dias diferentes devido a limitações de tempo durante as disciplinas dos cursos.

Para avaliar as questões de pesquisa propostas nesse estudo observamos as seguintes métricas, as quais são comuns em ambientes de submissão de atividades de programação.

- Número de submissões (**NSub**): com o intuito de observar se a criação/execução de testes de entrada e saída antes da fase de codificação (oráculo) resulta em um número menor de tentativas de submissão, com a resposta se aproximando da solução de referência em menos tentativas. O objetivo é capturar a contagem até a última submissão feita e tida como correta.

- Tempo (**T**): tempo gasto, em minutos, entre o aluno recebendo uma especificação e o aluno submetendo a versão final do programa ao sistema.
- Taxa de corretude funcional (**TC**): para observar o quão funcionalmente corretas estão as respostas fornecidas pelos alunos. Consiste em realizar a contagem de testes que foram aceitos (TA) e realizar a divisão pelo total de testes (TT) cadastrados para a solução. Exemplo: [5(TA) / 8(TT)] = 62,5%.

3.2. Seleção dos exercícios

A seleção dos exercícios de programação para serem aplicados nesse estudo levou em consideração premissas para garantir que um maior número de alunos pudessem participar de maneira efetiva, como por exemplo, exercícios que estivessem dentro do conteúdo já abordado na disciplina; que tivessem o mesmo nível de dificuldade; e que pudessem ser executados em até trinta minutos.

Tendo estas preocupações em mente, utilizamos como uma das referência os exercícios do *URI Online Judge - beecrowd*[1], uma vez que estão organizados por categorias e identificados por nível de dificuldade. Selecionamos quatro exercícios, dois deles de origem da plataforma - 'Idade em dias' e 'Soma de ímpares consecutivos' - e os outros dois propostos em [Paiva et al. 2021] - 'Calculando desconto' e 'Calculando IMC'.

Vale evidenciar que selecionamos quatro exercícios para aplicar três estratégias seguindo esta organização: 2 exercícios para aplicação utilizando o oráculo, 1 exercício para o método usual e 1 exercício para o método baseado no TDD. Desta forma objetivamos realizar comparações entre pares de estratégias: Usual *vs.* Oráculo e TDD *vs.* Oráculo.

3.3. Definição das ferramentas

Com os exercícios selecionados, partimos para a definição e preparação dos ambientes que os alunos deveriam utilizar para aplicar os procedimentos propostos pelo estudo. Analisamos plataformas de submissão de exercícios de programação em busca daquela que apresentasse maior flexibilidade de uso e uma interface simples que não prejudicasse o fluxo de trabalho dos alunos, optando ao final pelo *MIMIR Classroom* [2].

No que se refere a abordagem alvo desta pesquisa, o oráculo, desenvolvemos uma prova de conceito capaz de realizar o comportamento esperado pelo oráculo: munida das soluções de referência dos exercícios, a ferramenta oferece uma interface que permite que o usuário forneça uma entrada para determinado problema e a saída que ele imagina que o programa irá gerar. A Figura 2 apresenta o menu de acesso aos exercícios cadastrados no Oráculo[3] (à esquerda) e o menu de interação (à direita).

3.4. Elaboração dos questionários

Como última etapa da metodologia, partimos para a elaboração dos questionários para registrar dados referentes à identificação e ao *feedback* dos participantes. Com o questionário[4] aplicado antes do estudo, objetivamos coletar informações dos alunos que participaram com relação às suas aptidões frente a resolução dos problemas de programação.

[1] https://www.beecrowd.com.br/judge/en/login
[2] https://www.mimirhq.com/
[3] Disponível em: https://github.com/almdanddre/oraculo
[4] Disponível em: https://forms.gle/nTDtwHrTTt47m9Bw6

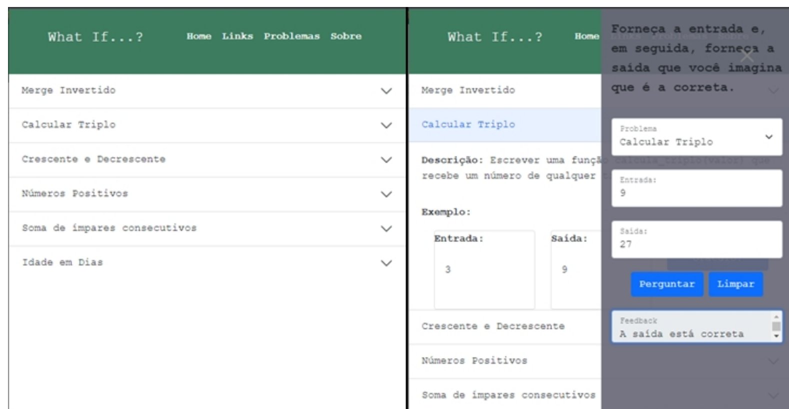

Figura 2. Telas de interação do oráculo.

Após a aplicação do estudo, com o segundo questionário[5], pretendemos verificar a opinião dos alunos sobre a efetividade do suporte oferecido pela estratégia alvo do trabalho: a interação com o oráculo através de pares de testes de entrada e saída.

4. Resultados e Discussão

Os participantes do estudo, totalizando 45 alunos, foram convidados a uma atividade extra-classe. Inicialmente houve uma apresentação de cerca de vinte minutos explicando como seria realizado o estudo e quais plataformas os alunos precisariam acessar para realizar o planejado. Em ambas as turmas o processo de resolução dos exercícios foi o mesmo: (1) na primeira sessão, os alunos tiveram que resolver o primeiro exercício da forma usual e o segundo exercício com auxílio do oráculo; (2) Na segunda sessão, os alunos tiveram que resolver o primeiro exercício através da estratégia TDD e o segundo exercício com auxílio do oráculo.

Como mencionado, com base em algumas métricas, objetivamos analisar o desempenho dos alunos realizando o contraste entre diferentes estratégias de resolução de exercícios de programação, Iniciamos a análise quantitativa com a métrica tempo de resolução, seguida pelo número de submissões e, por último, a taxa de corretude funcional das respostas.

4.1. Tempo de resolução (T)

Neste ponto analisamos o tempo total de resolução de cada um dos quatro exercícios propostos com o intuito de investigar, principalmente, se alunos que utilizaram o oráculo convergem mais rapidamente para as respectivas soluções corretas.

No gráfico de boxplots e violinos (Figura 3) são apresentados os resultados. Os violinos representam a concentração (ou densidade) das informações. Basicamente, as curvas mais largas indicam uma maior frequência de pontos. Por convenção, como os alunos utilizaram o oráculo em dois momentos, nomeamos o método de “Oráculo A” e “Oráculo B”. É possível observar que, tomando como *baseline* o maior tempo nas estratégias usual e TDD, o tempo de resolução utilizando o oráculo não ultrapassa este limite. Isto revela que o oráculo não chega a influenciar negativamente no processo de

[5]Disponível em: https://forms.gle/1LCVcTNTZrmE8zwj6

resolução dos exercícios, mesmo sendo uma ferramenta extra que os alunos precisam utilizar para alcançar suas soluções.

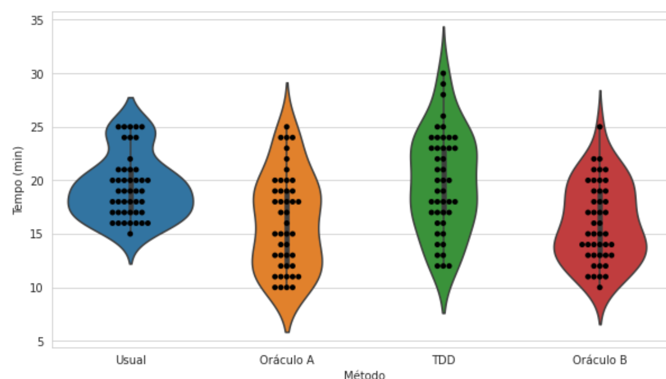

Figura 3. Distribuição do tempo para cada método aplicado.

A fim de responder às questões de pesquisa, realizamos o teste de Wilcoxon para amostras pareadas. Nesse teste de hipótese, o cenário (1) é o Usual vs. Oráculo A e o (2) é o TDD vs. Oráculo B. Considerando um valor-p = 0,05, obtivemos um p-valor = 0,0009 para o caso (1) e p-valor = 0,003 para o caso (2). Como o p-valor obtido em ambos os casos foi menor que o estabelecido, podemos rejeitar a hipótese nula e considerar que existe uma diferença significativa na variável tempo de resolução, favorecendo o oráculo. Formalmente, ou seja, os alunos que resolvem as questões utilizando o oráculo como apoio apresentam **T** menor que na estratégia usual e na estratégia TDD.

Na comparação Usual vs. Oráculo A percebemos que 69% dos alunos apresentam tempo de resolução menor se utilizando o oráculo, enquanto que 60% também apresentam tempo menor para esta mesma estratégia na comparação TDD vs. Oráculo B. Em alguns casos, alunos gastaram quase que metade do tempo que levaram para resolver o primeiro exercício; assim como houve alunos que passaram mais tempo para alcançar a solução quando utilizando o oráculo.

4.2. Número de submissões (NSub)

Ao analisar essa variável, objetivamos identificar com esta variável se os alunos que utilizaram o oráculo convergem mais rapidamente para a solução aceita em todos os casos de teste. Consideramos apenas o número de vezes que submeteram à plataforma até a última submissão.

Na Figura 4 podemos fazer uma análise sobre a métrica. Observamos que o espectro de número de submissões nas abordagens que não utilizam o oráculo é razoavelmente maior que nas demais, além da média também apresentar o mesmo comportamento.

Assim como com o tempo de resolução, para realizar o teste de Wilcoxon para amostras pareadas, precisamos realizar comparações dois a dois. Sendo assim, realizamos o teste de hipótese para os cenários (1) Usual vs. Oráculo A e (2) TDD vs. Oráculo B. Considerando um valor-p = 0,05, obtivemos um p-valor = 0,001 para o caso (1) e p-valor = 0,01 para o caso (2). Como o p-valor obtido em ambos os casos foi menor que o estabelecido, podemos rejeitar a hipótese nula e considerar que existe uma diferença significativa na variável número de submissões, favorecendo o oráculo. Formalmente,

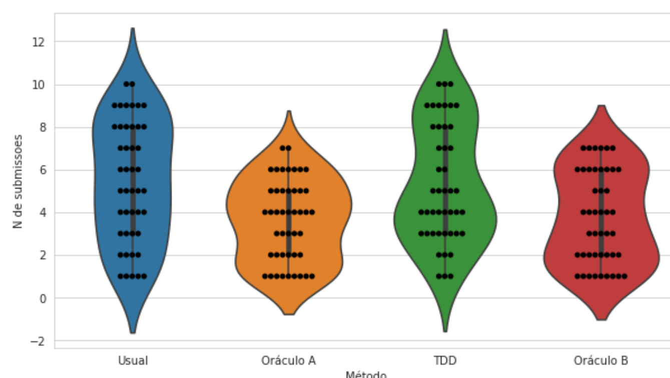

Figura 4. Distribuição do NSub para cada método aplicado.

ou seja, os alunos que resolvem as questões utilizando o oráculo como apoio apresentam **NSub** menor que na estratégia usual e na estratégia TDD.

4.3. Taxa de Corretude Funcional (TC)

Na análise desta métrica, percebemos que todos os alunos que participaram do estudo atingiram uma taxa de correção funcional de 100% em todos os exercícios. Sendo assim, não foi conduzido um teste de hipótese para verificar a conjectura referente a esta métrica, ficando para uma posterior avaliação considerando ajustes no cenário de aplicação, como no nível de dificuldade do exercício e no tempo dado para a resolução, fatores que podem ter impactado nesse resultado.

4.4. Análise Qualitativa

Outro ponto de vista que pode ser explorado é sobre como, individualmente, se comportou o número de submissões de cada aluno nas duas estratégias em comparação direta. Utilizamos o mesmo tipo de visualização que para a variável tempo de resolução, conforme ilustrado nas Figuras 5 e 6.

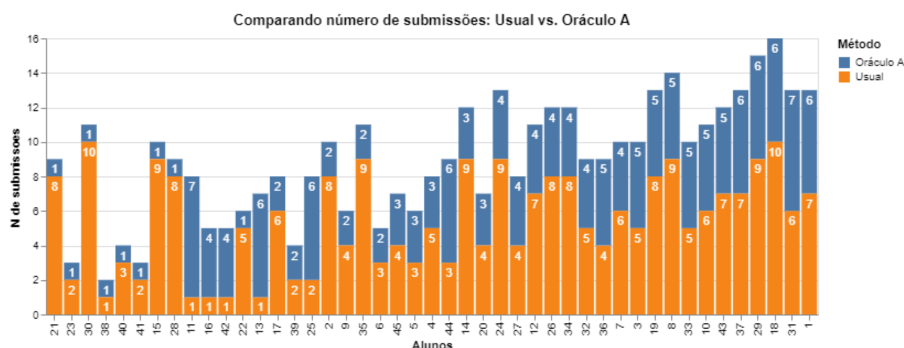

Figura 5. NSub, por aluno, para resolver cada questão proposta na primeira sessão do estudo.

Na comparação Usual vs. Oráculo A percebemos que 69% dos alunos apresentam NSub menor se utilizando o oráculo, enquanto que 67% também apresentam NSub menor para esta mesma estratégia na comparação TDD vs. Oráculo B. É perceptível que, apesar da análise conduzida para desconsiderar as submissões consecutivas sem alterações

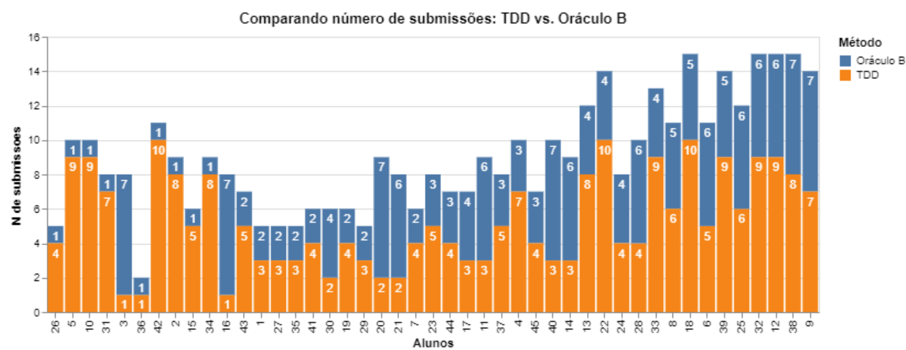

Figura 6. NSub, por aluno, para resolver cada questão proposta na segunda sessão do estudo.

visíveis no código, em alguns casos os alunos realizaram muito mais submissões quando utilizando o oráculo e é isto que nos leva a conduzir uma análise qualitativa com o intuito de entender as particularidades desses casos e sobre a representatividade dos mesmos.

Para a Figura 5 temos que os alunos 11, 16, 42 e 25 apresentaram NSub muito maior na estratégia que utiliza o oráculo se comparada com a forma usual. Analisando de maneira geral, os alunos tiveram dificuldade no emprego dos operadores aritméticos e sobre como aplicar a função *print()* de acordo com a formatação pedida. O trecho de código da Figura 7 representa a forma como os alunos procederam para resolver o exercício e a dificuldade de organização dos operadores aritméticos para alcançar a formatação.

1ª versão
```
i = int(input())

y = i//365
m = (i%365)//12
d = ((i%365)%12)//30

print(y,'A',m,'M',d,'D')
```

2ª versão
```
i = int(input())

y = i//365
m = (i%365)//30
d = ((i%365)%12)//30

print(y,'A',m,'M',d,'D')
```

3ª versão
```
i = int(input())

y = i//365
m = (i%365)//30
d = ((i%365)%12)%30

print(y,'A',m,'M',d,'D')
```

4ª versão
```
i = int(input())

y = i//365
m = (i%365)//30
d = (i%365)%30

print(f"{y}A{m}M{d}D")
```

Figura 7. Dificuldade na aplicação dos operadores e saída especificada.

Para a Figura 6, temos que os alunos 3, 16, 20 e 40 apresentaram NSub muito maior na estratégia que utiliza o oráculo se comparada com a forma TDD. Apesar do oráculo, os alunos tiveram dificuldade no entendimento de que o exercício pedia para realizar a soma dos números ímpares consecutivos de acordo com um intervalo, mas que os valores a serem somados não podiam ser os que definiam o intervalo e sim os valores que ficam entre esse intervalo. Além disso, os alunos sentiram a dificuldade de compreender que os números dados como entrada poderiam estar na ordem crescente ou decrescente, e que essa variação necessitaria de verificação para organizar a execução do laço *for*. O trecho de código da Figura 8 representa a forma como os alunos procederam para resolver o exercício e a dificuldade de organização dos possíveis cenários para o exercício.

Sobre o exercício aplicado empregando o método usual, o ‘Calculando Desconto’, a principal dificuldade dos alunos foi calcular valor final do produto, uma vez que seria necessário subtrair o valor do desconto do valor inicial do mesmo. No exercício aplicando

Submissão Inicial

```
x = int(input())
y = int(input())
tally = 0
for rep in range(x,y):
  if rep % 2 != 0:
    tally += rep
print(tally)
```

Submissão Intermediária

```
x = int(input())
y = int(input())
tally = 0
if x > y:
  for rep in range(x,y):
    if rep % 2 != 0:
      tally += rep
if y > x:
  for rep in range(y,x):
    if rep % 2 != 0:
      tally += rep
print(tally)
```

Submissão Final

```
x = int(input())
y = int(input())
tally = 0
if x > y:
  for rep in range(x,y,-1):
    if rep % 2 != 0 and rep != x:
      tally += rep
if x < y:
  for rep in range(x,y):
    if rep % 2 != 0 and rep != x:
      tally += rep
print(tally)
```

Figura 8. Dificuldade na representação de mais de um cenário para o exercício.

o método TDD, o ‘Calculando IMC’, entre as principais dificuldades estão (1) os operadores de comparação a serem utilizados para considerar o intervalo correto do IMC, (2) confusão nos tipos das variáveis peso e altura durante a leitura e (3) atenção ao resultado do cálculo do IMC e a exibição da mensagem de situação situação - os *prints* deveriam ser iguais ao apresentado na especificação do problema.

5. Conclusão e Trabalhos Futuros

Este artigo analisa a efetividade de uma estratégia baseada em testes para melhorar o entendimento de enunciados de exercícios de programação. A hipótese é que os alunos que refletem sobre o problema, ampliando a compreensão sobre o seu enunciado, à medida que apresentam entradas e em seguida, raciocinam, constroem e propõem as suas respectivas saídas.

O Oráculo foi desenvolvido de tal forma que o “teste de entendimento” pudesse ser feito através de testes de entrada e saída. Com esta ferramenta objetivamos oferecer suporte no primeiro passo do processo de resolução de problemas e obtivemos resultados que retratam esta estratégia como sendo promissora, no que se refere a alcançar soluções corretas mais rapidamente, aspecto abordado pela QP2 do estudo. Além disso, os *feedbacks* fornecidos pelos alunos, através do questionário, foram positivos e construtivos de tal forma que guiam alguns dos trabalhos futuros.

Vislumbrando algumas melhorias no *design* do estudo, podemos traçar alguns trabalhos futuros com o intuito de tornar ainda mais evidente os benefícios da estratégia. Como a reexecução do estudo para analisar melhor a métrica **TC** e obter uma resposta mais conclusiva à QP1 e minimizar algumas das ameaças à validade; o aperfeiçoamento do *feedback* oferecido pelo oráculo, bem como outras formas de interação.

6. Ameaças à Validade

Cabe ressaltar algumas ameaças a validade com relação a aplicação do oráculo. No que tange a validade interna podemos apontar a execução do estudo em dias diferentes, o que permitiu que os alunos dispusessem de uma janela de tempo para fixar melhor os conteúdos do curso. Atrelado a isto está a heterogeneidade dos participantes, uma vez que o nível de conhecimento das turmas se mostrou misto. Para minimizar estas questões, nos certificamos em executar o estudo em um momento do curso em que todos os alunos já dispusessem do conhecimento prévio para aplicar em seus exercícios de programação.

A situação pandêmica causada pelo Coronavírus (COVID-19) dificultou o acesso presencial aos alunos e a própria execução do estudo, impactando na validade externa. Estudos envolvendo pessoas naturalmente tem o viés referente à particularidade dos indivíduos. Sendo assim, não podemos estabelecer qualquer relação de causalidade com a população geral dos aprendizes de programação introdutória.

Referências

Ala-Mutka, K. (2004). Problems in learning and teaching programming-a literature study for developing visualizations in the codewitz-minerva project. *Codewitz needs analysis*, 20.

Almeida, A., Araújo, E., and Figueiredo, J. (2020). Avaliando a construção do conhecimento em programação através da taxonomia solo. In *Anais do XXXI Simpósio Brasileiro de Informática na Educação*, pages 1813–1822. SBC.

Basu, S., Wu, A., Hou, B., and DeNero, J. (2015). Problems before solutions: Automated problem clarification at scale. In *Proceedings of the Second (2015) ACM Conference on Learning@ Scale*, pages 205–213.

Beck, K. (2003). *Test-driven development: by example*. Addison-Wesley Professional.

Denny, P., Prather, J., Becker, B. A., Albrecht, Z., Loksa, D., and Pettit, R. (2019). A closer look at metacognitive scaffolding: Solving test cases before programming. In *Proceedings of the 19th Koli Calling international conference on computing education research*, pages 1–10.

Edwards, S. H. and Perez-Quinones, M. A. (2008). Web-cat: automatically grading programming assignments. In *Proceedings of the 13th annual conference on Innovation and technology in computer science education*, pages 328–328.

Garner, S., Haden, P., and Robins, A. (2005). My program is correct but it doesn't run: a preliminary investigation of novice programmers' problems. In *Proceedings of the 7th Australasian conference on Computing education-Volume 42*, pages 173–180.

Kadar, R., Wahab, N. A., Othman, J., Shamsuddin, M., and Mahlan, S. B. (2021). A study of difficulties in teaching and learning programming: a systematic literature review. *Int. J. Acad. Res. Progress. Educ. Dev*, 10:591–605.

Lahtinen, E., Ala-Mutka, K., and Järvinen, H.-M. (2005). A study of the difficulties of novice programmers. *Acm sigcse bulletin*, 37(3):14–18.

McCracken, M., Almstrum, V., Diaz, D., Guzdial, M., Hagan, D., Kolikant, Y. B.-D., Laxer, C., Thomas, L., Utting, I., and Wilusz, T. (2001). A multi-national, multi-institutional study of assessment of programming skills of first-year cs students. In *Working Group Reports from ITiCSE on Innovation and Technology in Computer Science Education*, ITiCSE-WGR '01, page 125–180, New York, NY, USA. Association for Computing Machinery.

Neto, V. L., Coelho, R., Leite, L., Guerrero, D. S., and Mendonça, A. P. (2013). Popt: a problem-oriented programming and testing approach for novice students. In *2013 35th international conference on software engineering (ICSE)*, pages 1099–1108. IEEE.

Paiva, F., Souza, G., Nascimento, J., and Martins, R. (2021). Introdução a python com aplicações de sistemas operacionais.

Polya, G. and Conway, J. (2004). *How to Solve It: A New Aspect of Mathematical Method*. Penguin mathematics. Princeton University Press.

Wrenn, J. and Krishnamurthi, S. (2019). Executable examples for programming problem comprehension. In *Proceedings of the 2019 ACM Conference on International Computing Education Research*, pages 131–139.