Mapeamento e análise empírica de *misconceptions* comuns em avaliações de introdução à programação

Ada Araújo
Universidade Federal do Amazonas
Manaus, Amazonas
asfa@icomp.ufam.edu.br

Daniel L. Z. Filho
Universidade Federal do Amazonas
Manaus, Amazonas
dlzf@icomp.ufam.edu.br

Elaine Harada T. Oliveira
Universidade Federal do Amazonas
Manaus, Amazonas
elaine@icomp.ufam.edu.br

Leandro S. Galvão de Carvalho
Universidade Federal do Amazonas
Manaus, Amazonas
galvao@icomp.ufam.edu.br

Filipe Dwan Pereira
Universidade Federal de Roraima
Boa Vista, Roraima
filipedwan@gmail.com

David B. F. Oliveira
Universidade Federal do Amazonas
Manaus, Amazonas
david@icomp.ufam.edu.br

RESUMO

Misconceptions são erros de compreensão que inviabilizam o entendimento de conceitos acadêmicos por parte dos alunos, não permitindo que eles apliquem tais conceitos da forma esperada. No contexto de aprendizagem de programação, os *misconceptions* envolvem erros de sintaxe, falta de entendimento da semântica e outras dificuldades enfrentadas pelos alunos enquanto estão codificando. Para que professores de programação possam ministrar aulas de qualidade é importante que eles saibam quais são os *misconceptions* comuns, pois com essa informação os docentes podem adaptar sua metodologia com aulas que possam potencialmente minimizar a ocorrência desses erros, atuando assim de forma preventiva. Entretanto, a literatura aponta uma escassez de estudos empíricos que mapeiem e analisem os *misconceptions* de modo holístico. Nesse sentido, este trabalho apresenta um mapeamento e análise dos *misconceptions* cometidos por estudantes ao desenvolverem soluções para problemas de codificação nas avaliações parciais de uma disciplina introdutória de programação, ofertada para cursos de ensino superior de ciências exatas e engenharias. Além disso, analisou-se a associação dos *misconceptions* com tópicos (módulos) ministrados em um curso de introdução à programação ofertado para turmas de ciências exatas e engenharia presenciais de uma instituição pública de ensino superior. Em termos de número, foram analisados 1.068 códigos desenvolvidos por 81 alunos.

CCS CONCEPTS

• **Applied computing** → **Computer-assisted instruction**.

PALAVRAS-CHAVE

erros de compreensão, ensino de programação, misconceptions, análise dirigida aos dados, juiz online

EduComp'21, Abril 27–30, 2021, Jataí, Goiás, Brasil (On-line)

1 INTRODUÇÃO

Existe uma demanda crescente por profissionais de computação para impulsionar a inovação e o progresso econômico [17, 33]. Esforços para melhorar o ensino de programação estão em andamento [8, 11, 14, 16, 19, 20, 25, 26, 28, 29, 32, 39, 43], e professores da área são desafiados em cursos introdutórios de programação a ajudar os alunos a desenvolverem o pensamento computacional e a habilidade de resolver problemas por meio de soluções algorítmicas [14, 19, 21, 27, 31, 35, 41, 43]. Identificar e abordar os equívocos de compreensão dos alunos é uma parte fundamental da competência de um professor de programação [1, 7, 14–16, 42]. No entanto, pesquisas relevantes sobre esse tópico não são tão desenvolvidas no campo do ensino de ciência da computação como no ensino de matemática e outras ciências [30, 31].

É crescente o interesse de pesquisadores em usar coleções de dados de programação para entender como os alunos aprendem, quais as dificuldades que enfrentam e o que é possível fazer para melhorar o ensino da ciência da computação [6, 13, 18, 20, 22, 24, 25, 27–29, 32, 43]. Esse crescimento está ocorrendo ao mesmo tempo em que aumenta o campo de estudos com o uso de análise dirigida a dados [3, 22, 26, 32]. No entanto, a maioria dos estudos examinaram principalmente a saída do código, e deixaram de investigar o comportamento do aluno no ambiente e o código em si [5, 10, 22, 24]. É possível que, ao ignorar esses fatores, esteja-se perdendo uma grande quantidade de informações úteis [3, 27, 34].

Os alunos de disciplinas introdutórias de programação exibem vários entendimentos errôneos de conceitos e outras dificuldades nos conhecimentos sintático, conceitual e estratégico [18, 27, 38]. Tais dificuldades experimentadas pelos alunos estão relacionadas a muitos fatores, incluindo falta de familiaridade com a sintaxe, linguagem natural, conhecimento matemático, modelos mentais imprecisos, falta de estratégias, ambientes de programação e conhecimento e instrução dos professores [27, 35]. Nos trabalhos de pesquisa em educação escritos em língua inglesa, esses erros de compreensão são chamados de *misconceptions* [30, 31, 38]. Entretanto, diante da falta de um termo preciso para tradução, optou-se por manter o termo em inglês no presente artigo.

Perceba que esses erros nem sempre são nitidamente identificáveis e podem ser cometidos por vários motivos, como falta de atenção, falta de preparo técnico do aluno, falta de conhecimento

da linguagem, falta de engajamento, etc. Por exemplo, se o aluno esquecer a indentação em comandos aninhados na linguagem *Python*, é possível que seja um caso de falta de atenção. No entanto, esse mesmo erro de indentação pode acontecer porque o aluno não sabe indentar o código, ou seja, um erro por falta de conhecimento técnico do linguagem. Esses dois exemplos não podem ser caracterizados como "erros de compreensão", que seria uma tradução mais fiel ao termo *misconceptions*.

Embora nomes diferentes sejam usados para representar as dificuldades dos alunos em aprender a programar, há um consenso geral de que é importante ajudar os alunos a desenvolver sua compreensão conceitual e superar desafios [4, 14, 15, 23, 27, 28, 30, 35]. Para isso, é vital para o ensino das disciplinas introdutórias de programação saber quais são os *misconceptions* comuns [1, 14–16] para dar suporte aos professores, já que com essa informação os docentes podem adaptar sua metodologia com aulas que possam potencialmente minimizar a ocorrência desses erros de compreensão, atuando assim de forma preventiva. Além disso, uma análise dos erros comuns pode propiciar uma intervenção precoce manual ou automática, aplicando-se abordagens e ferramentas instrucionais eficazes para dirimir potenciais dificuldades enfrentadas pelos alunos [22, 23, 35, 36, 40]. Diante dos potenciais benefícios, em um estudo recente, Qian et al. [30] apontam que existe uma escassez de instrumentos de medição concretos e estudos empíricos que mapeiem e avaliem *misconceptions* comuns.

Nesse sentido, o presente trabalho apresenta um mapeamento e uma análise dos *misconceptions* comuns cometidos por estudantes e sua associação aos assuntos ministrados em um curso de introdução à programação. Os *misconceptions* foram cometidos quando alunos desenvolviam soluções para problemas de programação nos exames de avaliação da disciplina Introdução à Programação de Computadores (IPC), ofertada para cursos presenciais de graduação da Universidade Federal do Amazonas. Para isso, foram analisados códigos, utilizados em avaliações, que foram submetidos à correção automática de um juiz online. Para guiar a análise, foram levantadas as seguintes questões de pesquisa:

- Q1: Quais e quantos *misconceptions* estão sendo cometidos em cada módulo da disciplina?
- Q2: Existe alguma relação entre os *misconceptions* e o enunciado da questão?

O restante deste trabalho está organizado como segue. A Seção 2 apresenta os trabalhos relacionados. Em seguida, descreve-se o ambiente educacional na Seção 3. A Seção 4 descreve a metodologia aplicada para analisar os exercícios de codificação e identificar os *misconceptions*. Os resultados iniciais são descritos na Seção 5 e as conclusões e perspectivas futuras do trabalho são apresentados na Seção 6.

2 TRABALHOS RELACIONADOS

Um dos pontos primordiais para a capacitação de professores é o desenvolvimento do Conhecimento do Conteúdo Pedagógico[1] (CCP) [30]. O CCP é o conhecimento necessário para fazer com que um dado assunto seja compreensível para os alunos [37]. Carlsen [2] explica que um dos componentes essenciais do CCP é o entendimento dos *misconceptions* no processo de aprendizagem dos estudantes.

Qian and Lehman [31] realizaram uma revisão da literatura relacionada aos tipos de *misconceptions* que os alunos de disciplinas de programação enfrentam. Os autores destacaram 8 fatores que contribuem significativamente para a ocorrência de *misconceptions*: falta de familiaridade com a sintaxe, linguagem natural, conhecimento matemático, modelos mentais imprecisos, falta de estratégias, ambientes de programação e conhecimento e instrução dos professores. Em uma extensão do trabalho anterior [31], Qian et al. [30] explicam que existe um entendimento limitado sobre o conhecimento que os professores deveriam ter a respeito dos *misconceptions* mais comuns que os estudantes enfrentam. Ademais, Qian et al. [30] apontam que apesar de já existirem trabalhos sobre *misconceptions* na área de pesquisa de educação em computação, existe também uma escassez de instrumentos de medição concretos e estudos empíricos que mapeiem e avaliem *misconceptions* comuns.

O trabalho de Ettles et al. [9] apresenta uma análise dos erros de lógica comuns cometidos por alunos de introdução à programação. Os erros eram detectados quando o código retornava uma saída diferente da desejada. Para tanto, analisou-se em qual caso de teste o código falhou a fim de ter um direcionamento de onde poderia estar o erro para então classificá-lo. Assim, ocorreu uma análise manual das saídas dos códigos com base em um vetor de casos de teste gerado pelo desenvolvedor. Como resultado, os autores listaram os erros de lógica mais comuns como ponto de partida para projetar intervenções de ensino específicas para resolvê-los. Além disso, Wiegand et al. [40] propuseram um algoritmo para identificar os conceitos de programação informativamente fáceis/difíceis dentre os tópicos de disciplinas introdutórias à programação e apresentaram os resultados em uma tabela dividida em 12 tópicos diferentes que são comumente cobertos em um curso introdutório de programação típico.

Já Gama et al. [12] apresentam um extenso estudo sobre *misconceptions* cometidos por alunos de introdução à programação em Python. Eles elaboraram um inventário com 28 hipóteses de possíveis *misconceptions*, agrupados em categorias. Por fim, os autores revelam que mais estudos sobre análises de *misconceptions* precisam ser conduzidos em outros cenários educacionais para que haja um avanço nesse campo de pesquisa.

Diante disso, este estudo tem como principal contribuição a realização de um estudo empírico de *misconceptions* que é realizado através de uma análise holística dos códigos submetidos por alunos em avaliações parciais ao longo de um curso de introdução à programação. Dessa forma, esse trabalho atende a uma chamada da literatura de educação em computação por trabalhos que mapeiem e analisem *misconceptions* comuns com o intuito de aprimorar o ensino de programação através da capacitação dos professores com um dos componentes essenciais do CCP que é o entendimento das *misconceptions* no processo de aprendizagem dos alunos. Para tanto, optou-se por realizar uma extensão do trabalho de Gama et al. em razão da similaridade do cenário educacional do trabalho deles com o nosso.

[1] Em inglês, *Pedagogical Content Knowledge* (PDK) [37]

3 CONTEXTUALIZAÇÃO

O presente trabalho foi desenvolvido a partir da análise dos *misconceptions* cometidos por alunos de Introdução à Programação de Computadores (IPC) da Universidade Federal do Amazonas. Nessa universidade, a ementa da disciplina de IPC é dividida em 07 tópicos ou módulos, com os seguintes conteúdos:

M1: variáveis e programação sequencial
M2: estruturas condicionais
M3: estruturas condicionais aninhadas
M4: repetição por condição
M5: vetores e strings
M6: repetição por contagem
M7: matrizes

Cada módulo está associado a uma lista de exercícios e a uma avaliação parcial. Nas listas de exercícios, os estudantes podem resolver questões de escrita de código em qualquer local e horário, dentro do prazo de duas semanas, em geral. Também são liberados para consultar colegas, o tutor de sua turma ou a bibliografia. Já as avaliações são realizadas em sala de aula (laboratório de informática), sem consulta e com tempo máximo de duração de uma hora (turmas maiores que a capacidade da sala) ou de duas horas (turmas até a capacidade do espaço). Cada avaliação pode conter duas ou três questões, variando de acordo com o tamanho da turma.

Todos os trabalhos desenvolvidos na disciplina (listas e avaliações) são realizados, entregues e corrigidos pelo juiz online *CodeBench* [2], que é um ambiente de correção automática de códigos desenvolvido por pesquisadores da Universidade. Esse ambiente é utilizado por todas as turmas de IPC da instituição, por isso, foi utilizado como uma plataforma para a análise das submissões das avaliações dos alunos. Dessa forma, o sistema guarda todas as interações do usuário com o sistema: comandos de teclado, cliques, códigos submetidos, entre outras.

4 METODOLOGIA

A metodologia apresentada a seguir foi utilizada para realizar a análise empírica dos códigos e identificação dos *misconceptions* relativos às 7 avaliações parciais, codificados em Python, dos alunos de duas turmas de IPC.

4.1 Coleta de dados

Optou-se por avaliar apenas duas turmas: a primeira é do primeiro semestre de 2018 (turma 2018/1) do curso de Engenharia Mecânica, com 39 alunos que concluíram a disciplina; e a segunda turma aconteceu no primeiro semestre de 2019 (turma 2019/1) do curso de Engenharia de Materiais, com 42 alunos concludentes. A escolha dessas duas turmas foi baseada no fato de que o primeiro semestre de 2019 obteve a menor taxa de aprovação (cerca de 30%) na disciplina de uma série histórica de 20 semestres, ou seja, em 10 anos. Desta forma, foram consideradas uma turma com uma alta taxa de reprovação e outra com desempenho melhor.

Os dados foram coletados a partir dos códigos submetidos pelos alunos no juiz online nas questões das avaliações parciais de IPC. Na correção, o juiz online testava o código com 3 casos de teste. Foram feitas 7 avaliações parciais que foram aplicadas nas 2 turmas. No entanto, na turma 2018/1 cada avaliação continha 3 questões, já na turma 2019/1 os três primeiros trabalhos continham 2 questões e o restante 3. Assim, obtivemos o total de 3.159 códigos para realizar a análise.

O foco da disciplina de IPC e deste trabalho é em alunos de cursos de graduação que não são da área de Computação. Como alguns alunos do Instituto de Computação da Universidade solicitam IPC como disciplina optativa e existem alunos matriculados que nunca utilizaram o sistema, foi necessário filtrá-los. Desta forma, para que os códigos de um aluno fossem analisados, esse deveria estar matriculado em um curso que não fosse do Instituto de Computação e ter feito no mínimo uma avaliação parcial, ainda que tenha obtido nota zero. Por fim, após todos os filtros, a turma 2018/1 contabilizou 819 códigos e a 2019/1, 756 códigos.

4.2 Categorização de código

A análise de cada código foi realizada por meio de leitura e interpretação, podendo ser dividida em duas partes:

(1) identificação do erro
(2) categorização do erro

A primeira parte consistiu em verificar se o aluno acertou ou errou a questão. Caso o aluno tenha errado, foi verificado em qual caso de teste o código falhou. Caso tenha retornado um erro sintático, esse foi relacionado a algum *misconception*, caso contrário foi realizada a leitura do código e interpretação do erro para relacionar a um *misconception*. A segunda parte da análise foi a categorização do erro, em que procurou-se relacionar os *misconceptions* encontrados nos códigos com as categorias apresentadas, conforme será apresentado na próxima subseção. Em alguns casos, os erros não eram compatíveis com nenhum dos listados, por isso, para obter um resultado mais abrangente optou-se por criar uma categoria adicional detalhada na próxima seção.

4.3 Relação Estendida de *Misconceptions*

Conforme mencionado na seção 2, Gama et al. [12] relacionam uma lista de *misconceptions* cometidos por alunos na linguagem Python. Já que o escopo deste trabalho é semelhante, optou-se por utilizar uma adaptação desta relação. A ementa de IPC adotada na Universidade onde este estudo foi conduzido não abrange todos os assuntos abordados por Gama et al. [12], como por exemplo recursão. Por isso, foi necessário primeiramente realizar um recorte da relação inicial de *misconceptions*.

Após isso, observou-se que não foi possível categorizar alguns *misconceptions* com base na relação obtida por [12]. Dessa forma, surgiu a necessidade de criar uma categoria chamada de *Adicional*, representada pela letra **A**. Essa categoria adicional atua como uma extensão do recorte realizado neste estudo. A seguir, as categorias que restaram após a extensão do recorte e a Tabela 1 contém uma breve descrição de cada *misconception* individualmente:

- PA: Parâmetro de funções e escopo
- PB: Variáveis, identificadores e escopo
- PD: Iteração
- PH: Uso e implementação de classe e objetos
- X: Específicos do Python
- A: Adicional

[2] http://http://codebench.icomp.ufam.edu.br/

Tabela 1: Descrição dos *misconceptions* identificados por Gama et al. [12]

ID	Descrição
PA.3	Supõe-se que os parâmetros de função possam ser acessados em qualquer lugar do programa, independentemente do escopo
PB.1	Atribuição fora do escopo, considerando ou classificando variáveis locais como se fossem variáveis globais
PD.7	O laço `for` é declarado sem uma instância de intervalo (por exemplo, iterando sobre elementos de uma lista), mas contém código que usa o iterador como um subscrito em notação de colchete
PH.1	Tentativa de chamar um método existente de uma classe diferente daquela em que é implementado
PH.4	Um atributo de uma determinada classe (e, portanto, não incorporada) é chamado, sem ter sido importado anteriormente ou sem ser chamado de uma classe ou instância da mesma
PH.5	Tentativa de definir um valor para o identificador de um método
X.1	O laço `for` é considerado uma função, e a variável de iteração deveria ser considerada local para o escopo da suposta função
X.2	O laço `for` é chamado para um objeto `range`, mas o laço foi construído como se tivesse definido sobre os membros de um iterable, como uma lista
X.4	Tentativa de invocar um método que retorna uma nova instância de uma classe a qual não foi definida
X.5	Tentativa de chamar um método que retorna uma nova instância desta classe e não atribuir essa instância a uma variável
X.6	O laço `for` é chamado para iterar sobre membros de um iterable, como uma lista, mas o código do laço é construído como se fossem definidos em um objeto range
X.7	O método é chamado sem ser importado ou, sem ter referenciando um módulo com ponto de notação se o próprio módulo foi importado
X.9	Tentativa de atribuir um valor a um método, em vez de passar esse valor como parâmetro
X.8	Tentativa de alterar o objeto str pelo índice
X.10	Uso incorreto da notação de partição
X.11	Tentativa de chamar laço `for` ou `while` e não colocar um objeto de iteração
X.12	Tentar concatenar objetos `str` sem usar o operador +
X.13	Agindo no pressuposto de que um valor de retorno atribuído anteriormente de uma função será automaticamente atualizada quando os parâmetros fornecidos para a função são alteradas
X.14	Comparando dois objetos referenciados por suas referências quando compara métodos que não foram definidos para essa classe
A.1	Não utilizar corretamente operadores relacionais (>=, <=, **!**= e ==), afetando a condição ou resultando erro de sintaxe
A.2	Utilizar os operadores lógicos **and** e **or** incorretamente, afetando a condição ou resultando erro de sintaxe
A.3	Utilizar indentação errada que afeta o escopo de funções
A.4	Não utilizar os operadores (**, %, +=) corretamente, afetando a lógica da equação matemática
A.5	Não declarar ou utilizar incorretamente variáveis (utilização de variável não declarada ou não utilização das declaradas)
A.6	Não utilizar/fechar parênteses, chaves, aspas, vírgula, pontuação depois de funções
A.7	Utilizar `input` incompatível com o tipo de entrada solicitada pelo problema
A.8	Usar os comandos **else** ou **elif** sem um **if** anterior correspondente

Os novos *misconceptions* foram adicionados de acordo com a necessidade. Ou seja, se um erro novo foi encontrado, um novo *misconception* foi criado para aquele contexto. Os Adicionais englobam erros sintáticos e lógicos considerados simples, como por exemplo utilizar `else` e `elif` sem um `if` antes. A criação destes *misconceptions* foi importante, pois, nas seções seguintes, é demonstrado o quanto os erros adicionais impactaram nos resultados.

5 RESULTADOS

Nesta seção são apresentados os *misconceptions* das questões de cada aluno, dispostos em uma tabela (Tabela 2) e posteriormente agrupados para uma análise quantitativa (Figura 2).

5.1 Caracterização da Análise

Neste trabalho, consideramos as seguintes definições de *acerto* e *desistência*:

- um **acerto** é caracterizado quando o código passa sem erros pelos casos de teste cadastrados para uma dada questão no juiz online.
- uma **desistência** ocorre quando se observa um dos seguintes casos:
  - o código não passou nos casos de teste e também não foram identificadas ocorrências de *misconceptions*.
  - o código está incompleto, indicando que o estudante não terminou de conceber a resolução da questão. A Figura 1 exibe um exemplo deste segundo caso de desistência. Note que o estudante só digitou uma linha de código.

Dessa forma, reuniu-se as questões de cada aluno, seus acertos, desistências e *misconceptions* cometidos. A partir dessas informações, foram extraídas duas listas: uma com os diferentes *misconceptions* ocorridos por módulo e outra com os *misconceptions* mais cometidos por questão.

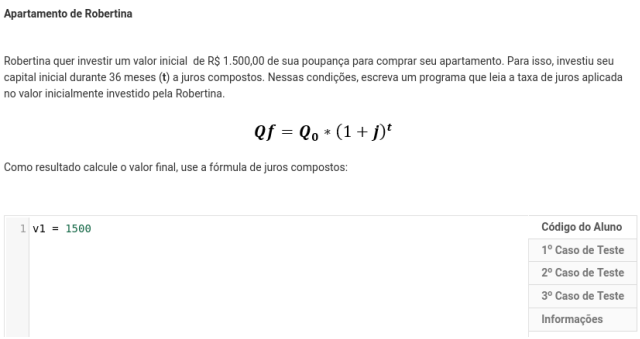

Figura 1: Exemplo de código em que houve desistência do aluno

Tabela 2: Diferentes *misconceptions* por módulo e por ano.

	2018		2019	
Mód.	**#**	***Misconceptions***	**#**	***Misconceptions***
M1	3	A.3, A.4, A.5	9	A.4, A.5, A.6, A.7, X.5, X.7, X.9, PH.5, PH.4
M2	5	A.1, A.3, A.5, A.4, A.6	6	A.1, A.3, A.4, A.5, A.6, A.7
M3	7	A.1, A.3, A.5, A.6, X.5, X.12, X.14	8	A.1, A.2, A.3, A.4, A.5, A.6, A.7, A.8
M4	10	A.1, A.2, A.3, A.4, A.5, A.7, PA.3, PB.1, X.11, X.13	10	A.1, A.2, A.3, A.4, A.5, A.6, A.7, A.8, X.11, X.13
M5	12	A.1, A.3, A.4, A.5, PD.7, PH.1, X.1, X.8, X.5, X.10, X.13	10	A.1, A.2, A.3, A.4, A.5, A.6, A.7, X.4, X.10, X.11
M6	11	A.1, A.3, A.4, A.5, PD.7, PH.4, X.1, X.2, X.6, X.11, X.14	17	A.1, A.3, A.4, A.5, A.6, A.7, A.8, PB.1, PD.7, PH.4, X.1, X.2, X.4, X.10, X.11, X.13, X.14
M7	10	A.1, A.2, A.4, A.5, A.6, PH.4, X.5, X.10, X.11, X.14	9	A.5, A.6, A.7, X.2, X.4, X.6, X.7, X.10, X.11

5.2 Quais e quantos *misconceptions* foram cometidos em cada módulo da disciplina?

Ao realizar o agrupamento dos resultados em relação a cada avaliação parcial individual dos estudantes, obteve-se a quantidade de *misconceptions* cometidos em cada módulo, apresentada na Tabela 2.

Na primeira coluna tem-se os módulos da disciplina, na segunda tem-se a quantidade de *misconceptions* diferentes que ocorreram em 2018. Pode-se observar que a avaliação parcial 5, no módulo Vetores e Strings, obteve uma ocorrência de 12 *misconceptions* diferentes. Na coluna "*Misconceptions* cometidos em 2018", é apresentada a lista com a identificação de cada *misconception* ocorrido. Nas duas próximas colunas, tem-se os dados das duas colunas anteriores, mas para a turma de 2019.

No gráfico da Figura 2 é possível verificar que, em 2019, houve uma ocorrência maior de diferentes *misconceptions*. O maior valor foi de 17 *misconceptions* no módulo 6, enquanto, em 2018, foi de 12 no módulo 5.

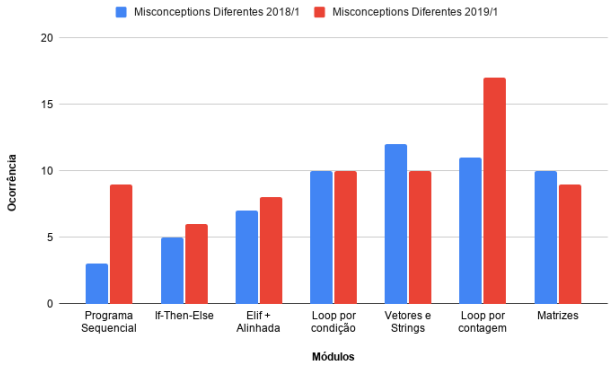

Figura 2: Gráfico de ocorrência de diferentes *misconceptions* das turmas 2018/1 e 2019/1

O próximo passo foi observar as categorias dos *misconceptions* ocorridos, de forma a visualizar melhor em quais conceitos os alunos cometeram um *misconception*. A Tabela 3 contém uma relação dessas categorias com base nos dados da Tabela 2.

Com isso, observou-se que a categoria Adicionais (A) se destaca por aparecer em todos os módulos, seguida da categoria de Específicos do Python (X) aparecendo em 5 módulos de cada turma. A partir dessa informação, pode-se ter uma noção maior das dificuldades que os alunos podem estar passando em um determinado conceito/assunto.

A Tabela 4 trata da incidência de um determinado *misconception* em um módulo. Os dados representam uma incidência maior que um no formato $N(M)$, onde N é a quantidade de ocorrências de um *misconception* M.

Pode-se interpretar que, por exemplo, na primeira avaliação parcial, sobre estrutura sequencial, na turma de 2018, tem-se 2 ocorrências do *misconception* A.5 relacionadas à utilização de variáveis, enquanto na turma de 2019, no mesmo módulo, foram 9 ocorrências do mesmo *misconception*.

5.3 Existe alguma relação entre os *misconceptions* e o enunciado da questão?

Inicialmente, buscou-se saber qual o módulo com maior quantidade de *misconceptions* diferentes na turma de 2019, a turma com maior taxa de reprovação. Assim, pode-se dizer que em uma questão ocorreu uma quantidade X de *misconceptions* diferentes. Nesse caso, de acordo com os dados analisados, o módulo com o maior número de *misconFceptions* diferentes foi o relacionado a laço de contagem, na avaliação parcial 6.

Com essa informação em mãos, analisou-se o enunciado das questões desse módulo para comparar com os enunciados do módulo

Tabela 3: Categorias dos *misconceptions* por módulo.

	Categorias dos *misconceptions*	
Mód.	**2018**	**2019**
M1	Adicionais	Adicionais, Específicos do Python, Implementação de Classes e Objetos
M2	Adicionais	Adicionais
M3	Adicionais, Específicos do Python	Adicionais
M4	Adicionais, Específicos do Python, Uso de Funções com Parâmetro e Escopo, Variáveis, Identificadores e Escopo	Adicionais, Específicos do Python
M5	Adicionais, Específicos do Python, Iterações - Laços, Implementação de Classes e Objetos	Adicionais, Específicos do Python
M6	Adicionais, Iterações - Laços, Específicos do Python, Implementação de Classes e Objetos	Adicionais, Específicos do Python, Variáveis, Identificadores e Escopo, Iterações - Laços, Implementação de Classes e Objetos
M7	Adicionais, Específicos do Python, Implementação de Classes e Objetos	Adicionais, Específicos do Python

correspondente, mas da turma de 2018/1. Ainda que as questões selecionadas variem de ano para ano e de aluno para aluno, o assunto mantém-se o mesmo, por isso, foi escolhido um aluno aleatório de cada ano que os enunciados das questões fossem analisados.

Abaixo, são apresentadas as primeiras questões das avaliações parciais do módulo 6 das turmas de 2018 e 2019 (Figuras 3 a 4):

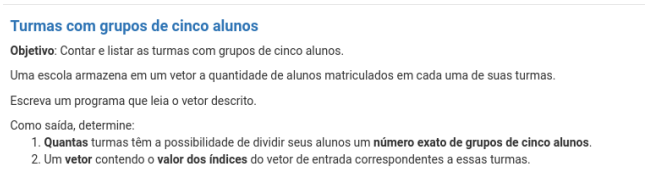

Figura 3: Questão 1 - Avaliação 6 na turma 2019/1

Ao realizar uma leitura atenta das questões, percebe-se que todas seguem o mesmo padrão quanto ao formato, definindo um objetivo, o dado que será fornecido como entrada e saída do programa. Caso os enunciados não apresentassem semelhança, nesta análise surgiriam outras problemáticas para investigar, como por exemplo, de qual forma a divergência dos enunciados interferem na análise dos *misconceptions*. Uma vez que os enunciados mantêm o mesmo

Tabela 4: Relação de incidência de *misconceptions* por módulo

	Misconceptions	
Mód.	**2018**	**2019**
M1	2(A.5)	9(A.5), 9(A.4), 4(A.7), 3(A.6), 3(X.7), 2(X.9)
M2	4(A.4), 3(A.3)	15(A.5), 9(A.1), 8(A.6), 8(A.4), 2(A.3), 2(A.7)
M3	4(A.1), 3(A.3), 2(A.5)	16(A.1), 7(A.5), 6(A.6), 5(A.2), 4(A.3), 3(A.4), 3(A.8), 2(A.7)
M4	8(A.3), 7(A.5), 6(X.11), 6(A.1), 4(A.4), 3(X.13), 3(PA.3), 3(A.2)	16(A.5), 12(A.4), 9(X.11), 7(A.3), 6(A.1), 4(A.6), 3(A.8), 3(X.13), 2(A.2)
M5	4(X.10), 3(A.5), 3(A.4), 3(A.3), 3(A.1), 2(X.8), 2(X.5)	15(A.5), 11(A.4), 5(A.6), 5(A.7), 4(A.1), 3(X.10), 3(A.3), 2(X.11)
M6	8(A.5), 4(A.1), 4(PD.7), 2(X.6), 2(A.4), 3(A.3)	13(A.5), 5(A.4), 4(A.6), 2(X.11), 2(X.10), 2(A.7), 2(X.4), 2(A.1)
M7	7(X.10), 3(X.11), 2(A.6), 2(X.5), 2(A.1), 2(A.5)	7(A.5), 3(X.7), 2(X.11)

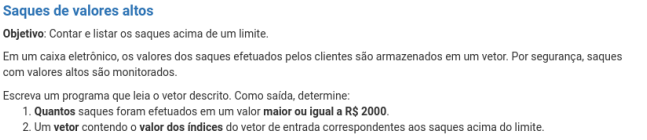

Figura 4: Questão 1 - Avaliação 6 na turma 2018/1

padrão, foi possível comparar os diferentes *misconceptions* obtidos no módulo 6 para a turma de 2018 e 2019, apresentado nas Figuras 5 e 6.

A diferença entre os semestres analisados é em relação à quantidade de *misconceptions* diferentes, onde em 2019 tem-se 24 e em 2018, 18. Em 2019, ocorreram além dos *misconceptions* cometidos em 2018, os seguintes: A.7, A.8, PB.1, X.4, X.10 e X.13. Além disso, não ocorreu a categoria Variáveis, identificadores e escopo em 2018 e a categoria Adicional expressou ocorrência maior na turma de 2019.

6 CONCLUSÃO

Após realizar a análise do agrupamento dos *misconceptions* em relação ao módulo em que cada avaliação parcial é aplicada e com base na questão de cada trabalho, é possível concluir que:

A turma de 2018 teve mais dificuldade na avaliação parcial 5, o módulo de vetores e strings, apresentando *misconceptions* das categorias: adicionais, específicos do Python, iterações - laços, implementação de classes e objetos. Neste módulo, o *misconception*

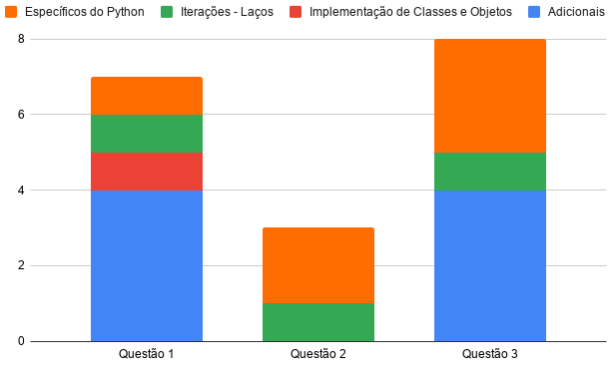

Figura 5: Categorias de *misconceptions* cometidos em cada questão na avaliação parcial 6 - 2018/1

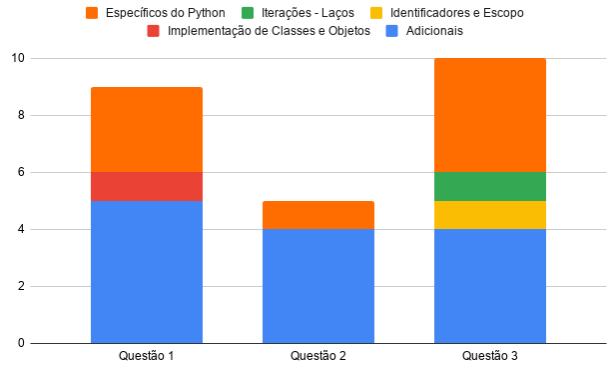

Figura 6: Categorias de *misconceptions* cometidos em cada questão na avaliação parcial 6 - 2019/1

que teve maior ocorrência foi o X.10, e de todos os módulos os *misconceptions* que mais ocorreram foram A.3 e A.5, cada um com 8 ocorrências nos módulos 4 e 6.

Já a turma de 2019 teve maior dificuldade na avaliação parcial no módulo laço por contagem, apresentando *misconceptions* das categorias: adicionais, específicos do Python, variáveis, identificadores e escopo, iterações - laços, implementação de classes e objetos. Neste, o *misconception* mais cometido foi o A.5, e de todos os módulos os *misconceptions* que mais ocorreram foram A.1 e A.5, cada um com 16 ocorrências nos módulos 3 e 4.

Inicialmente buscou-se saber qual o módulo com maior quantidade de *misconceptions* diferentes na turma de 2019, a que teve maior taxa de reprovação, não importando a quantidade total de ocorrências mas a variação de tipos de erros. De acordo com os dados analisados o módulo com mais *misconceptions* diferentes foi o laço por contagem, na avaliação parcial 6.

Além disso, percebeu-se que mesmo após o recorte da relação de *misconceptions* proposta por Gama et al. [12], nem todos foram utilizados, ou identificados. A Tabela 5 apresenta um comparativo entre os *misconceptions* encontrados e os que foram dispostos para realizar a identificação. Além disso, houve a adição de 8 *misconceptions*

Tabela 5: Relação de *misconceptions* do recorte e lista dos identificados durante a análise

Misconceptions	**Cometidos**
PA.3	PA.3
PB.1	PB.1
PD.7	PD.7
PH.1	PH.1
PH.4	PH.4
PH.5	PH.5
X.1	X.1
X.2	X.2
X.4	X.5
X.5	X.6
X.6	X.7
X.7	X.9
X.8	X.10
X.9	X.11
X.10	X.12
X.11	X.13
X.12	X.14
X.13	—
X.14	—

categorizados como Adicionais, não apresentados na Tabela 5. Com isso, foi possível obter uma lista de 25 *misconceptions* utilizados como base para análise dos códigos dos alunos.

Com a apresentação dos resultados, foi perceptível a grande incidência da categoria Adicional deixando os resultados mais abrangentes em relação ao tipo de erro que o aluno está cometendo.

Em relação ao enunciado das questões, não foi possível chegar a um parecer conclusivo por analisar-se apenas um único módulo e a avaliação parcial de apenas um aluno de cada turma. Mas ainda assim, pode-se obter a relação de *misconceptions* que foram cometidos em 2019 que não foram em 2018:

- **A.7** Tipo do comando `input` incompatível com o tipo de entrada solicitada pelo problema;
- **A.8** Usar os comandos **else** ou **elif** sem um **if** anterior correspondente.
- **PB.1** Atribuição fora do escopo, considerando ou classificando variáveis locais como se fossem variáveis globais;
- **X.4** Tentativa de invocar um método que retorna uma nova instância de uma classe a qual não foi definida;
- **X.10** Uso incorreto da notação de partição de vetores;
- **X.13** Agindo no pressuposto de que um valor de retorno atribuído anteriormente de uma função será automaticamente atualizado quando os parâmetros fornecidos para a função são alterados.

Dessa forma, com esta análise é possível realizar uma observação geral de quais assuntos os alunos de matérias introdutórias a programação de computadores têm mais dificuldade e que tipos de erros eles estão cometendo devido a essa dificuldade.

AGRADECIMENTOS

Esta pesquisa, realizada no âmbito do Projeto Samsung-UFAM de Ensino e Pesquisa (SUPER), nos termos do artigo 48 do Decreto nº 6.008/2006 (SUFRAMA), foi parcialmente financiada pela Samsung Eletrônica da Amazônia Ltda., nos termos da Lei Federal nº 8.387/1991, por meio dos convênios 001/2020 e 003/2019, firmados com a Universidade Federal do Amazonas e a FAEPI, Brasil. O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001.

REFERÊNCIAS

[1] Ricardo Caceffo, Pablo Frank-Bolton, Renan Souza, and Rodolfo Azevedo. 2019. Identifying and validating java misconceptions toward a cs1 concept inventory. In *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education*. 23–29.

[2] Williams Carlsen. 1999. Domains of teacher knowledge. In *Examining pedagogical content knowledge*. Springer, 133–144.

[3] Adam Carter, Christopher Hundhausen, and Daniel Olivares. 2019. Leveraging the IDE for learning analytics. *The Cambridge Handbook of Computing Education Research, Sally A. Fincher and Anthony V. Robins (Eds.). Cambridge University Press, Cambridge* (2019), 679–706.

[4] Yuliya Cherenkova, Daniel Zingaro, and Andrew Petersen. 2014. Identifying challenging CS1 concepts in a large problem dataset. In *Proceedings of the 45th ACM technical symposium on Computer science education*. 695–700.

[5] Hermino Barbosa de Freitas Júnior and Filipe Dwan Pereira. 2020. Recomendação de Problemas em Juízes Onlines Utilizando Técnicas de Processamento de Linguagem Natural e Análise Dirigida aos Dados. In *Anais dos Workshops do IX Congresso Brasileiro de Informática na Educação*. SBC, 94–94.

[6] Joseph de Oliveira, Felipe Salem, Elaine Harada Teixeira de Oliveira, David Braga Fernandes Oliveira, Leandro Silva Galvão de Carvalho, and Filipe Dwan Pereira. 2020. Os estudantes leem as mensagens de feedback estendido exibidas em juízes online?. In *Anais do XXXI Simpósio Brasileiro de Informática na Educação*. SBC, 1723–1732.

[7] Ingrid Lima dos Santos, David Braga Fernandes Oliveira, Leandro Silva Galvão de Carvalho, Filipe Dwan Pereira, and Elaine Harada Teixeira de Oliveira. 2020. Tempos de Transição em Estados de Corretude e Erro como Indicadores de Desempenho em Juízes Online. In *Anais do XXXI Simpósio Brasileiro de Informática na Educação*. SBC, 1283–1292.

[8] Filipe Dwan, Elaine Oliveira, and David Fernandes. 2017. Predição de zona de aprendizagem de alunos de introdução à programação em ambientes de correção automática de código. In *Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE)*, Vol. 28. 1507.

[9] Andrew Ettles, Andrew Luxton-Reilly, and Paul Denny. 2018. Common logic errors made by novice programmers. In *Proceedings of the 20th Australasian Computing Education Conference*. Association for Computing Machinery, New York, NY, USA, 83–89.

[10] Samuel Fonseca, Elaine Oliveira, Filipe Pereira, David Fernandes, and Leandro Silva Galvão de Carvalho. 2019. Adaptação de um método preditivo para inferir o desempenho de alunos de programação. In *Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE)*, Vol. 30. 1651.

[11] Samuel C Fonseca, Filipe Dwan Pereira, Elaine HT Oliveira, David BF Oliveira, Leandro SG Carvalho, and Alexandra I Cristea. 2020. Automatic subject-based contextualisation of programming assignment lists. *International Educational Data Mining Society* (2020).

[12] Guilherme Gama, Ricardo Caceffo, Renan Souza, Raysa Bennati, Tales Aparecida, Islene Garcia, and Rodolfo Azevedo. 2018. *An antipattern documentation about misconceptions related to an introductory programming course in Python*. Technical Report. Institute of Computing, University of Campinas, Tech. Rep. IC-18-19.

[13] Petri Ihantola, Arto Vihavainen, Alireza Ahadi, Matthew Butler, Jürgen Börstler, Stephen H Edwards, Essi Isohanni, Ari Korhonen, Andrew Petersen, Kelly Rivers, et al. 2015. Educational data mining and learning analytics in programming: Literature review and case studies. In *Proceedings of the 2015 ITiCSE on Working Group Reports*. Association for Computing Machinery, New York, NY, USA, 41–63.

[14] Fionnuala Johnson, Stephen McQuistin, and John O'Donnell. 2020. Analysis of Student Misconceptions using Python as an Introductory Programming Language. In *Proceedings of the 4th Conference on Computing Education Practice 2020*. 1–4.

[15] Lisa C Kaczmarczyk, Elizabeth R Petrick, J Philip East, and Geoffrey L Herman. 2010. Identifying student misconceptions of programming. In *Proceedings of the 41st ACM technical symposium on Computer science education*. 107–111.

[16] Cazembe Kennedy, Aubrey Lawson, Yvon Feaster, and Eileen Kraemer. 2020. Misconception-Based Peer Feedback: A Pedagogical Technique for Reducing Misconceptions. In *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education*. 166–172.

[17] Kai S Koong, Lai C Liu, and Xia Liu. 2020. A study of the demand for information technology professionals in selected internet job portals. *Journal of Information Systems Education* 13, 1 (2020), 4.

[18] Marcos Lima, Leandro Silva Galvão de Carvalho, Elaine Harada Teixeira de Oliveira, David Braga Fernandes Oliveira, and Filipe Dwan Pereira. 2020. Classificação de dificuldade de questões de programação com base em métricas de código. In *Anais do XXXI Simpósio Brasileiro de Informática na Educação*. SBC, 1323–1332.

[19] Andrew Luxton-Reilly, Ibrahim Albluwi, Brett A Becker, Michail Giannakos, Amruth N Kumar, Linda Ott, James Paterson, Michael James Scott, Judy Sheard, and Claudia Szabo. 2018. Introductory programming: a systematic literature review. In *Proceedings Companion of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education*. 55–106.

[20] Rodrigo Pessoa Medeiros, Geber Lisboa Ramalho, and Taciana Pontual Falcão. 2018. A systematic literature review on teaching and learning introductory programming in higher education. *IEEE Transactions on Education* 62, 2 (2018), 77–90.

[21] Filipe Pereira, Elaine Oliveira, David Fernandes, Leandro Silva Galvão de Carvalho, and Hermino Junior. 2019. Otimização e automação da predição precoce do desempenho de alunos que utilizam juízes online: uma abordagem com algoritmo genético. In *Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE)*, Vol. 30. 1451.

[22] Filipe Dwan Pereira, Linnik Maciel de Souza, Elaine Harada Teixeira de Oliveira, David Braga Fernandes de Oliveira, and Leandro Silva Galvão de Carvalho. 2020. Predição de desempenho em ambientes computacionais para turmas de programação: um Mapeamento Sistemático da Literatura. In *Anais do XXXI Simpósio Brasileiro de Informática na Educação*. SBC, 1673–1682.

[23] Filipe Dwan Pereira, Samuel C Fonseca, Elaine HT Oliveira, David BF Oliveira, Alexandra I Cristea, and Leandro SG Carvalho. 2020. Deep learning for early performance prediction of introductory programming students: a comparative and explanatory study. *Brazilian journal of computers in education*. 28 (2020), 723–749.

[24] Filipe D Pereira, Elaine Oliveira, Alexandra Cristea, David Fernandes, Luciano Silva, Gene Aguiar, Ahmed Alamri, and Mohammad Alshehri. 2019. Early dropout prediction for programming courses supported by online judges. In *International Conference on Artificial Intelligence in Education*. Springer, Springer International Publishing, Cham, 67–72.

[25] Filipe Dwan Pereira, Elaine HT Oliveira, David Fernandes, and Alexandra Cristea. 2019. Early performance prediction for CS1 course students using a combination of machine learning and an evolutionary algorithm. In *2019 IEEE 19th International Conference on Advanced Learning Technologies (ICALT)*, Vol. 2161. IEEE, 183–184.

[26] Filipe D Pereira, Elaine HT Oliveira, David BF Oliveira, Alexandra I Cristea, Leandro SG Carvalho, Samuel C Fonseca, Armando Toda, and Seiji Isotani. 2020. Using learning analytics in the Amazonas: understanding students' behaviour in introductory programming. *British Journal of Educational Technology* 51, 4 (jul 2020), 955–972.

[27] Filipe Dwan Pereira, Elaine H T Oliveira, and David F B Oliveira. 2018. *Uso de um método preditivo para inferir a zona de aprendizagem de alunos de programação em um ambiente de correção automática de código*. Mestrado em Informática. Universidade Federal do Amazonas, Manaus.

[28] Filipe Dwan Pereira, Francisco Pires, Samuel Fonseca, Elaine Oliveira, Leandro Carvalho, David Oliveira, and Alexandra Cristea. 2021. Towards a Human-AI hybrid system for categorising programming problems *(SIGCSE '21)*. Association for Computing Machinery, New York, NY, USA, 7. https://doi.org/10.1145/3408877.3432422

[29] Filipe D Pereira, Armando Toda, Elaine HT Oliveira, Alexandra I Cristea, Seiji Isotani, Dion Laranjeira, Adriano Almeida, and Jonas Mendonça. 2020. Can We Use Gamification to Predict Students' Performance? A Case Study Supported by an Online Judge. In *International Conference on Intelligent Tutoring Systems*. Springer, 259–269.

[30] Yizhou Qian, Susanne Hambrusch, Aman Yadav, Sarah Gretter, and Yue Li. 2020. Teachers' Perceptions of Student Misconceptions in Introductory Programming. *Journal of Educational Computing Research* 58, 2 (2020), 364–397.

[31] Yizhou Qian and James Lehman. 2017. Students' misconceptions and other difficulties in introductory programming: A literature review. *ACM Transactions on Computing Education (TOCE)* 18, 1 (2017), 1–24.

[32] Keith Quille and Susan Bergin. 2019. CS1: how will they do? How can we help? A decade of research and practice. *Computer Science Education* 29, 2-3 (2019), 254–282.

[33] Meenakshi Raman and Anjani Srikanth Koka. 2015. The ever-increasing demand for soft skills at workplace: A Study on IT professionals' perspectives. In *International Conference on Management and Information Systems*, Vol. 18. 4–8.

[34] Kelly Rivers, Erik Harpstead, and Kenneth R Koedinger. 2016. Learning curve analysis for programming: Which concepts do students struggle with?. In *ICER*. Association for Computing Machinery, New York, NY, USA, 143–151.

[35] A. V. Robins. 2019. Novice programmers and introductory programming. In *The Cambridge Handbook of Computing Education Research*. Cambridge University

Press, Cambridge, Chapter 12, 327–376.

[36] Kate Sanders and Lynda Thomas. 2007. Checklists for grading object-oriented CS1 programs: concepts and misconceptions. *ACM SIGCSE Bulletin* 39, 3 (2007), 166–170.

[37] Lee Shulman. 1987. Knowledge and teaching: Foundations of the new reform. *Harvard educational review* 57, 1 (1987), 1–23.

[38] Keith S Taber. 2013. *Modelling learners and learning in science education.* Springer, Netherlands.

[39] Szymon Wasik, Maciej Antczak, Jan Badura, Artur Laskowski, and Tomasz Sternal. 2018. A survey on online judge systems and their applications. *ACM Computing Surveys (CSUR)* 51, 1 (2018), 1–34.

[40] R Paul Wiegand, Anthony Bucci, Amruth N Kumar, Jennifer L Albert, and Alessio Gaspar. 2016. A data-driven analysis of informatively hard concepts in introductory programming. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*. Association for Computing Machinery, New York, NY, USA, 370–375.

[41] Aman Yadav, Sarah Gretter, Susanne Hambrusch, and Phil Sands. 2016. Expanding computer science education in schools: understanding teacher experiences and challenges. *Computer Science Education* 26, 4 (2016), 235–254.

[42] Žana Žanko, Monika Mladenović, and Ivica Boljat. 2019. Misconceptions about variables at the K-12 level. *Education and Information Technologies* 24, 2 (2019), 1251–1268.

[43] Daniel Lopes Zordan Filho, Elaine Harada Teixeira de Oliveira, Leandro Silva Galvão de Carvalho, Marcela Pessoa, Filipe Dwan Pereira, and David Braga Fernandes de Oliveira. 2020. Uma análise orientada a dados para avaliar o impacto da gamificação de um juiz on-line no desempenho de estudantes. In *Anais do XXXI Simpósio Brasileiro de Informática na Educação*. SBC, 491–500.