

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DO RIO GRANDE
PROGRAMA DE PÓS-GRADUAÇÃO EM MODELAGEM COMPUTACIONAL

**MODELO PARA AVALIAÇÃO DE CÓDIGOS FONTE ATRAVÉS
DE ANÁLISE ESTÁTICA**

por

Ricardo Lemos de Souza

Dissertação para obtenção do Título de
Mestre em Modelagem Computacional

Rio Grande, Agosto, 2021

Ricardo Lemos de Souza

**“MODELO PARA AVALIAÇÃO DE CÓDIGOS FONTE ATRAVÉS DE ANÁLISE
ESTÁTICA”**

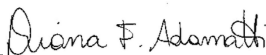
Dissertação apresentada ao Programa de Pós-Graduação em Modelagem Computacional da Universidade Federal do Rio Grande - FURG, como requisito parcial para obtenção do Grau de Mestre. Área de concentração: Modelagem Computacional.

Aprovado em: 26/08/2021

BANCA EXAMINADORA



Prof. Dra. Silvia Silva da Costa Botelho (Orientadora – FURG)



Prof. Dra. Diana Francisca Adamatti (FURG)



Prof. Dr. Thiago Thompsen Primo (UFPel)

Rio Grande - RS
2021

Ficha Catalográfica

S729m Souza, Ricardo Lemos de.

Modelo para avaliação de códigos fonte através de análise
estática / Ricardo Lemos de Souza. – 2021.
83 f.

Dissertação (mestrado) – Universidade Federal do Rio Grande –
FURG, Programa de Pós-Graduação em Modelagem Computacional,
Rio Grande/RS, 2021.

Orientadora: Dra. Sílvia Silva da Costa Botelho.

1. Avaliação de Códigos Fonte 2. TF-IDF 3. Pensamento
Computacional 4. Análise Estática I. Botelho, Sílvia Silva da Costa
II. Título.

CDU 004.4'415

Catálogo na Fonte: Bibliotecário José Paulo dos Santos CRB 10/2344

Em memória de minha mãe, Glória Maria, professora que me fez tanta falta nos ensinamentos da vida.

AGRADECIMENTOS

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Código de Financiamento 001.

À Universidade Federal do Rio Grande e aos seus professores que participaram e contribuíram na minha jornada.

Agradeço à todos os membros do grupo de pesquisa Sapiens, que muito contribuíram para o desenvolvimento do presente trabalho.

Agradeço à minha orientadora, professora Doutora Sílvia Silva da Costa Botelho, por ter me acolhido e agraciado com seu imenso saber e experiência, e que será sempre um exemplo de excelência em pesquisa.

À minha família, meu pai João e meu irmão Leonardo que sempre estiveram ao meu lado quando mais precisei.

À Fabiana, meu porto seguro nos últimos anos e, que Deus permita, nos que virão.

RESUMO

No mundo atual, observa-se a presença cada vez maior de dispositivos com alguma capacidade computacional embarcada sendo utilizados nos mais diferentes aspectos do cotidiano das pessoas, consequentemente o mundo contemporâneo também precisa de mais profissionais capazes de elaborar e manter softwares. A crescente demanda de profissionais programadores acarreta também no aumento de cursos voltados ao ensino de programação. Em um contexto educacional, o processo de avaliação é fundamental para adequação do processo de ensino, e dada a diversidade possível de respostas a problemas de programação, a avaliação é uma das atividades que demandam maior esforço por parte dos educadores. O presente trabalho propõe um modelo baseado em frequência de termos e frequência inversa de documentos para avaliação automática de códigos fonte, a fim de identificar e estimar habilidades do Pensamento Computacional. Para validação do modelo, elaborou-se um experimento no qual comparou-se a avaliação de especialistas com os dados obtidos a partir da aplicação do modelo em uma base de dados, composta por códigos fonte contendo soluções de problemas de programação. Os resultados obtidos demonstram que o modelo é capaz de identificar e estimar competências do Pensamento Computacional a partir do processamento de códigos fonte pertencentes à um conjunto de dados. Espera-se que as informações extraídas a partir dessa metodologia possam ser utilizadas para fins educacionais, de modo a contribuir para o desenvolvimento de metodologias educacionais voltadas ao acompanhamento do desenvolvimento de habilidades e competências de estudantes em programação de computadores.

Palavras-chaves: Avaliação de códigos fonte, TF-IDF, Pensamento Computacional, Análise Estática.

ABSTRACT

In today's world, there is an increasing presence of devices with some embedded computational capacity being used in the most different aspects of people's daily lives, consequently the contemporary world also needs more professionals capable of developing and maintaining software. The growing demand for professional programmers also leads to an increase in courses aimed at teaching programming. In an educational context, the assessment process is fundamental for the adequacy of the teaching process, and given the possible diversity of responses to programming problems, assessment is one of the activities that demand a great effort on the part of educators. The present work proposes a model based on term frequency and inverse document frequency for the evaluation of source codes, in order to identify and estimate Computational Thinking skills. To validate the proposed model, an experiment was developed in which experts evaluations were compared with the data obtained from the application of the model in a database, composed of source codes containing solutions to programming problems. The results obtained demonstrate that the proposed model is able to identify and estimate Computational Thinking competences from by processing a set of source codes belonging to a data set. It is expected that the information extracted from this methodology can be used for educational purposes, in order to contribute to the development of educational methodologies aimed at monitoring the development of skills and competences of students in computer programming.

Palavras-chaves: Source Code Assessment, TF-IDF, Computational Thinking, Static Analysis.

ÍNDICE

1	Introdução	13
1.1	Objetivos	16
1.2	Organização do trabalho	17
2	Embasamento Teórico	18
2.1	Avaliação de códigos fonte	18
2.1.1	Avaliação automática de códigos fonte	18
2.1.2	Avaliação e <i>feedback</i>	22
2.2	Relevância e TF-IDF	24
2.2.1	TF-IDF na Avaliação de Códigos Fonte	26
2.3	Transformação do código fonte em uma representação numérica	27
2.4	Pensamento Computacional	31
3	Um modelo para avaliação de códigos fonte	38
3.1	Competências e suas representações no código fonte	38
3.1.1	Conceito - Coleção de dados	39
3.1.2	Conceito - Análise de dados	39
3.1.3	Conceito - Representação de dados	41
3.1.4	Conceito - Decomposição do problema	42
3.1.5	Conceito - Abstração	42
3.1.6	Conceito - Simulação	45
3.1.7	Conceitos não relacionados nesse trabalho	46
3.2	Matrizes de representação	47
3.3	Rubrica de avaliação	49
3.3.1	Proporção Theta	49
3.3.2	Habilidade Theta	51
4	Experimento de Validação	53
4.1	Definição e coleta dos dados	53
4.2	Avaliação humana	54
4.2.1	Critérios de avaliação	54
4.2.2	Resultados gerais da avaliação humana	55
4.3	Avaliação do modelo	62
4.3.1	Resultados do modelo	62
5	Considerações Finais	76
	Lista de publicações	78
	Referências	79

LISTA DE FIGURAS

2.1	Taxonomia das abordagens de AA	19
2.2	Processo genérico de análise estática	20
2.3	Exemplo de expressões	29
2.4	Transformação léxica e semântica em representação numérica	30
4.1	Variação das notas entre os avaliadores	58
4.2	Variação das notas entre os avaliadores no problema 1	59
4.3	Variação das notas entre os avaliadores na competência <i>Análise de dados</i> .	60
4.4	Variação das notas entre os avaliadores - <i>Abstração</i>	60
4.5	Dispersão com Jitter - <i>Análise de Dados</i> - Problema 5	61
4.6	Gráfico Box-Plot Habilidade Theta Problema 1	63
4.7	Comparação das benevolências	69
4.8	Box-plot Modelo (a) e do Avaliador 1 (b) do problema 1	70
4.9	Box-plot Modelo - problema 6 e médias por nota dos avaliadores	71
4.10	Box-plot Aceitos x Rejeitados	73
4.11	Comparação entre submissões corretas e incorretas	74
4.12	Sequência de submissões	75

LISTA DE TABELAS

1	Exemplo de tokens	28
2	Framework para avaliação das competências do Pensamento Computacional	35
3	Conceitos centrais do Pensamento Computacional e competências	36
4	Estruturas de entrada de dados	39
5	Tabela operadores aritméticos	40
6	Operadores lógicos e relacionais	43
7	Funções de saída stdio.h	46
8	Habilidades e tokens extraídos	47
9	Distribuição das submissões por problema	53
10	Instruções do instrumento de avaliação	55
11	Correlações das avaliações entre os avaliadores	56
12	Teste de normalidade dos dados do avaliador 1	57
13	Teste de normalidade Habilidade Theta	64
14	Correlações Modelo x Avaliador 1	65
15	Correlações Modelo x Avaliador 2	66
16	Benevolência por habilidade	72

LISTA DE SÍMBOLOS

tf	representa a <i>term-frequency</i>
idf	representa a <i>inverse-document-frequency</i>
N	corresponde ao número total de códigos fonte em um conjunto
std	desvio padrão
m	média da frequência
θI	Proporção Theta I
θD	Proporção Theta D
$\min f$	frequência mínima
$\max f$	frequência máxima
$H\theta$	Habilidade Theta

Sub índices

d	documento ou código fonte
t	termo ou token
D	conjunto de códigos fonte

LISTA DE ABREVIATURAS

AA	avaliação automática
AD	Análise Dinâmica
AE	Análise Estática
API	<i>Aplication Programming Interface</i>
AST	<i>Abstract Syntax Tree</i>
AV1	Avaliador Humano 1
AV2	Avaliador Humano 2
BOW	<i>Bag of Words</i>
CF	Código Fonte
IDF	<i>Inverse Term Frequency</i>
K-12	Curriculo fundamental em alguns países de língua inglesa
LSA	<i>Latent Semantic Analysis</i>
PC	Pensamento Computacional
SBC	Sociedade Brasileira de Computação
SVD	<i>Single Value Decomposition</i>
TF	<i>Term Frequency</i>

1 INTRODUÇÃO

A presença da computação cresce nos mais diferentes aspectos da contemporaneidade. Seja em computadores, telefones inteligentes, veículos ou eletrodomésticos capazes de conectar-se à internet, é cada vez maior o número de objetos que possuem algum tipo de software embarcado (BERMEJO et al., 2016; SU et al., 2016). A maior presença da computação em todos os setores acarreta também em um aumento na demanda de profissionais capazes de desenvolver e manter sistemas computacionais.

Cursos introdutórios de programação possuem reconhecidamente um alto número de taxas de reprovação e evasão (RAIGOZA, 2017; ROBINS, 2010). O aprendizado de programação de computadores engloba o desenvolvimento de um conjunto de habilidades como lógica, matemática, assim como outras não diretamente relacionados com as ciências da computação (CABO, 2019; ROBINS, 2010). Os novatos em programação deparam-se não apenas com o desafio de aprender uma linguagem de programação, mas também em como aplicar seus conhecimentos prévios em uma solução algorítmica (CABO, 2019). A dificuldade de relacionar diferentes habilidades para esse fim é apresentada como uma das principais justificativas para os níveis de reprovação e evasão (CABO, 2019; GOMES et al., 2008).

Em cursos presenciais, exercícios de implementação de algoritmos em código fonte são amplamente utilizados nas disciplinas de programação. Com o aumento do número de estudantes, o número de códigos resultado das atividades realizadas pelos alunos pode gerar uma demanda de trabalho elevada para os educadores, em consequência disso, é comum a busca por um modelo de avaliação automática de exercícios. Para atender à crescente demanda por programadores, também observa-se a propagação de novos ambientes de ensino de programação, como cursos em plataformas online, as quais constantemente procuram desenvolver metodologias para proporcionar um melhor ensino e aprendizagem, bem como metodologias mais precisas de avaliação (ALA-MUTKA, 2005; SOUZA et al., 2016).

Cursos e plataformas Web oferecem ao educador a oportunidade agregar diferentes serviços, tais como: administração, organização e atribuição de atividades à alunos. Muitas instituições de ensino desenvolveram seus próprios sistemas de gestão de atividades para o ensino de programação, outros fazem uso de plataformas desenvolvidas por terceiros, como os *Online Judges* (SOUZA et al., 2016). *Online Judges* são plataformas que possuem um banco de exercícios de programação. Nessas plataformas questões são disponibilizadas para escolha e submissão de código fonte com a solução para elas. Uma vez submetida, a solução é julgada, isto é, um sistema de avaliação automática produz e informa o resultado de sua avaliação. Como exemplo de *Online Judges* podemos citar as plataforma URI Online Judge (BEZ et al., 2014) e Codeforces (MIRZAYANOV, 2015).

Métodos de avaliação automática são comuns em questões objetivas, entretanto o universo de respostas possíveis em um exercício de programação assemelha-se à questões dissertativas, ou seja, mesmo que dois códigos fonte sejam diferentes eles podem chegar ao mesmo resultado. Os métodos de avaliação mais utilizados em programação estão englobados em duas classificações:

dinâmica e estática (ALA-MUTKA, 2005; ULLAH et al., 2018b).

A maior parte dos sistemas de avaliação faz uso de análise dinâmica de códigos fonte (AD). Esse método de avaliação requer a execução do código fonte e é frequentemente utilizado como uma avaliação binária, determinando se a resposta está correta ou não, de forma que dado um determinado conjunto de entradas, o conjunto de saídas geradas pelo código fonte é comparado com um conjunto previamente determinado (ULLAH et al., 2018b). Pode-se ainda identificar ramificações da avaliação dinâmica como o modelo de avaliação de funcionalidades, que consiste em executar o código fonte com diversos casos de teste, verificando-se a saída gerada, comparando-a a saída esperada.

Como outro exemplo de ramificação da avaliação dinâmica tem-se o modelo de avaliação de eficiência, que tipicamente executa o código fonte e verifica o comportamento do programa durante sua execução, ou seja, verifica-se dados de execução como tempo de execução, quantidade de memória utilizada, etc. (ALA-MUTKA, 2005). Outras classificações da avaliação dinâmica estão presentes na literatura, em todas observa-se a presença da sua característica identitária, a execução do código fonte.

A Análise Estática (AE) de códigos fonte, não requer a execução do código avaliado, utiliza-se da quantificação de elementos presentes no código fonte como a quantidade de identificadores e operadores, e é capaz de avaliar inclusive soluções que não puderam ser compiladas e executadas devido à algum erro (ALA-MUTKA, 2005). Assim como na análise dinâmica, na análise estática também é possível verificar ramificações, como o modelo de avaliação de estilo, no qual o sistema verifica se a sintaxe do código fonte está correta, assim como outras características como legibilidade, indentação, documentação, etc.. A maioria dos compiladores ou interpretadores fazem algum tipo de avaliação de estilo.

Outra ramificação da avaliação estática são os sistemas de avaliação de erros de programação, ainda que erros funcionais são mais facilmente verificados com a avaliação dinâmica, em alguns casos, como trechos de códigos suspeitos ou erros que frequentemente se repetem em um padrão, também podem ser avaliados estaticamente (ALA-MUTKA, 2005). Outras ramificações podem ser observadas na literatura, todas elas compartilham da sua característica identitária de avaliar o texto do código fonte em si, sem sua execução. Como um recurso educacional para avaliação, análise estática oferece uma oportunidade de coletar dados sobre como os estudantes aplicam suas habilidades e competências, e em um contexto histórico, possibilitam um mapeamento do seu desenvolvimento, não apenas pelas respostas corretas, mas também por respostas incorretas consideradas não executáveis, como aquelas que contém erros de compilação (ULLAH et al., 2018b).

Pode-se dizer que há ainda outros modelos de avaliação, como a híbrida e a semiautomática. A avaliação híbrida contempla aspectos tanto da avaliação dinâmica quanto da avaliação estática. Já a avaliação semiautomática faz uso da avaliação automática em alguns aspectos, mas não exclui a necessidade da avaliação humana para outros. Em um contexto educacional, é possível identificar-se muitos benefícios da avaliação automática, como rapidez, consistência, objetividade e disponibilidades. Contudo, nesse mesmo contexto educacional, os sistemas de avaliação

automática podem gerar um *feedback* que não satisfaça a subjetividade necessária para o acompanhamento do aluno (ALA-MUTKA, 2005).

Entende-se que a avaliação é um processo diagnóstico, que visa identificar elementos positivos e negativos nos processos de ensino e aprendizado, possibilitando a adequação do processo educacional (LUCKESI, 1986). Portanto considera-se importante a possibilidade de identificar possíveis lacunas no processo de aprendizado de estudantes de programação. Dessa forma, uma aferição de capacidade de forma binária, examinar apenas se o estudante é capaz de oferecer uma solução adequada ou não, não satisfaz a principal função do processo de avaliação. Faz-se necessário uma metodologia que contemple a identificação e avaliação das competências envolvidas na elaboração de códigos fonte para a resolução de problemas de programação, de modo a proporcionar meios de intervenções educacionais para o processo de ensino a fim de adequar o processo de aprendizagem às necessidades do aprendiz.

As competências em programação são frequentemente associadas aos conteúdos propostos no currículo dos cursos, tais como: o uso de seleções condicionais, laços de repetição e *arrays*. A avaliação tradicional nos cursos de computação frequentemente se utiliza então de modelos nos quais verifica-se se o aprendiz é capaz de executar com sucesso exercício ligados aos conteúdos apresentados, ou seja, verifica-se se o educando se enquadra no padrão de desenvolvimento esperado (CHRYSAFIADI; VIRVOU, 2013).

Existem abordagens educacionais que visam desenvolver, acompanhar e avaliar habilidades e competências de estudantes, como o Pensamento Computacional, o qual é utilizado não só em disciplinas de programação, mas também em outras áreas de conhecimento. O Pensamento Computacional refere-se a um processo de pensamento para a formulação ou entendimento de um problema e na elaboração de uma expressão de sua solução de forma que ela possa ser interpretada por um computador (WING, 2006). A partir dessa definição, percebe-se que as competências do Pensamento Computacional propostas por Wing (2006), *Abstração*, *Automação* e *Análise*, podem ter uma relação estreita com o desenvolvimento de algoritmos para solução de exercícios de programação, como no trabalho de Alves (2013). As competências do Pensamento Computacional também podem ser relacionadas com estruturas presentes em códigos fonte, como no trabalho de Barr e Stephenson (2011).

Um dos desafios na avaliação de competências é identificá-las a partir da análise de códigos fonte. Pode-se definir os códigos fonte como uma versão de software escritos em uma linguagem de texto plano, legível por um ser humano (LINUX, 2005). Portanto, as linguagens utilizadas para elaboração de códigos fonte, independente do tipo, possuem uma gramática com sintaxe própria, específica, semelhante à linguagem natural (LOUDEN; LAMBERT, 2011).

Métodos de recuperação de informações em linguagem natural são um campo de estudo das ciências da computação, com aplicação inclusive em outras áreas de conhecimento. Muitos sistemas de recuperação de informações utilizam métodos estatísticos para determinar a relevância de termos presentes nos textos analisados e, um método proeminente para este fim é o produto da Frequência de Termos com a Frequência Inversa do Documento (*TF-IDF: Term Frequency - Inverse Document Frequency*). Dado a proximidade das características dos textos produzidos em

linguagem natural e dos códigos fonte, o uso de métodos de recuperação de informações advindos do processamento de linguagem natural é uma alternativa viável à análise e recuperação de informações presentes em códigos fonte produzidos em linguagens de programação.

Sabe-se que a TF-IDF e outras metodologias de recuperação de informações já foram utilizadas para análise de códigos fonte, como no trabalho de Yang et al. (2016), no qual os autores determinam requisitos estabelecidos para o desenvolvimento de uma aplicação a partir da análise de seu código fonte, e no trabalho de Lucia et al. (2012), no qual os autores indicam a provável aplicabilidade de trechos de códigos fonte a partir de sua análise. Entende-se que para uma avaliação educacional o acompanhamento do desenvolvimento de competências é importante, e para isso é necessário estabelecer parâmetros de comparação entre o que é produzido, como na elaboração dos códigos que expressam uma solução de um exercício, e o que é desejado, como uma solução modelo. Dessa forma, percebe-se uma oportunidade para o uso da TF-IDF como um componente de avaliação em um contexto educacional: é possível determinar-se de que forma uma competência foi utilizada para a elaboração de soluções de problemas de programação expressas em códigos fonte e compará-la a outras soluções?

No presente trabalho propõe-se uma metodologia para avaliação das competências de estudantes associadas à escrita de um código em uma linguagem formal de programação. O modelo proposto foi aplicado em um estudo de caso associado ao desenvolvimento de códigos na linguagem “C”. Destaca-se como contribuição desta dissertação um modelo estatístico para identificação e mapeamento das competências associadas ao Pensamento Computacional presentes em códigos fonte, bem como sua implementação em ferramenta computacional capaz de ser usada em larga escala para avaliação de códigos em cursos de computação. Para a extração de informações sobre as competências de estudantes de programação a partir de códigos fonte, escolheu-se aplicar uma abordagem de recuperação de informações embasada em *Term Frequency - Inverse Document Frequency*. Como referência de competências, utilizou-se conceitos relacionados ao Pensamento Computacional (BARR; STEPHENSON, 2011).

1.1 Objetivos

O presente trabalho está voltado ao desenvolvimento de um modelo de avaliação estática de códigos fonte, que forneça informações sobre competências envolvidas na escrita de códigos fonte, de modo a permitir que os educadores possam identificar potenciais fraquezas no desenvolvimento das competências de alunos. Nessa perspectiva, definiu-se como objetivos:

- **Objetivo Geral:**

Propor um modelo de avaliação das competências de estudantes, envolvidas na elaboração de códigos fonte em uma linguagem formal de programação, a fim de auxiliar o ensino e aprendizado de programação em ambientes massivos.

- **Objetivos Específicos:**

- Definir um conjunto de competências presentes em um código fonte na linguagem C.
- Propor um método de extração de informações a partir de um código fonte.
- Estabelecer um modelo de avaliação de competências a partir de informações extraídas de um código fonte.
- Testar e validar o modelo através de ferramental estatístico.

1.2 Organização do trabalho

O presente estudo foi organizado em 5 capítulos. No segundo capítulo apresenta-se o embasamento teórico e as discussões acerca da avaliação de códigos fonte, os conceitos de relevância e TF-IDF, os métodos de transformação de códigos fonte em uma representação numérica e as competências do Pensamento Computacional. No terceiro capítulo apresenta-se a metodologia utilizada para elaboração do modelo proposto e utilizado na avaliação de códigos fonte em um estudo de caso. No quarto capítulo é apresentado o experimento realizado para validação do modelo proposto. Por fim, no quinto capítulo apresenta-se a discussão dos resultados obtidos e considerações finais, bem como os trabalhos futuros.

2 EMBASAMENTO TEÓRICO

O aprendizado de programação de computadores engloba o uso e desenvolvimento de um conjunto de habilidades que nem sempre são relacionadas às ciências da computação. Diferentemente dos conteúdos presentes nos cursos de introdução à programação de computadores, que possuem tópicos regulares mesmo que não se utilizem das mesmas ferramentas, as competências em programação nem sempre são consensuais (SOUZA et al., 2016). Quais são as principais competências para a elaboração de um algoritmo que expresse a solução de um problema? A literatura disponível não oferece um consenso a esta resposta. Um dos conjuntos de competências proeminentes no mundo contemporâneo é o Pensamento Computacional, que é referenciado não só em trabalhos relacionados ao ensino de programação de computadores ou às ciências da computação, mas também em outras áreas do conhecimento humano. O presente capítulo apresenta o embasamento teórico e discussões acerca da avaliação de códigos fonte, relevância e TF-IDF, métodos de representação de códigos fonte e Pensamento Computacional.

2.1 Avaliação de códigos fonte

A programação de computadores é um dos fundamentos centrais de cursos associados à ciência da computação (ALA-MUTKA, 2005), e no mundo contemporâneo, também se faz presente no currículo de outras áreas. Com a crescente demanda de programadores, e conseqüentemente crescente oferta de cursos, diferentes metodologias para avaliação das produções de alunos de programação tem sido desenvolvidas. A seguir destacam-se alguns trabalhos que abordam essas metodologias, em especial as metodologias de avaliação automática de códigos fonte.

2.1.1 Avaliação automática de códigos fonte

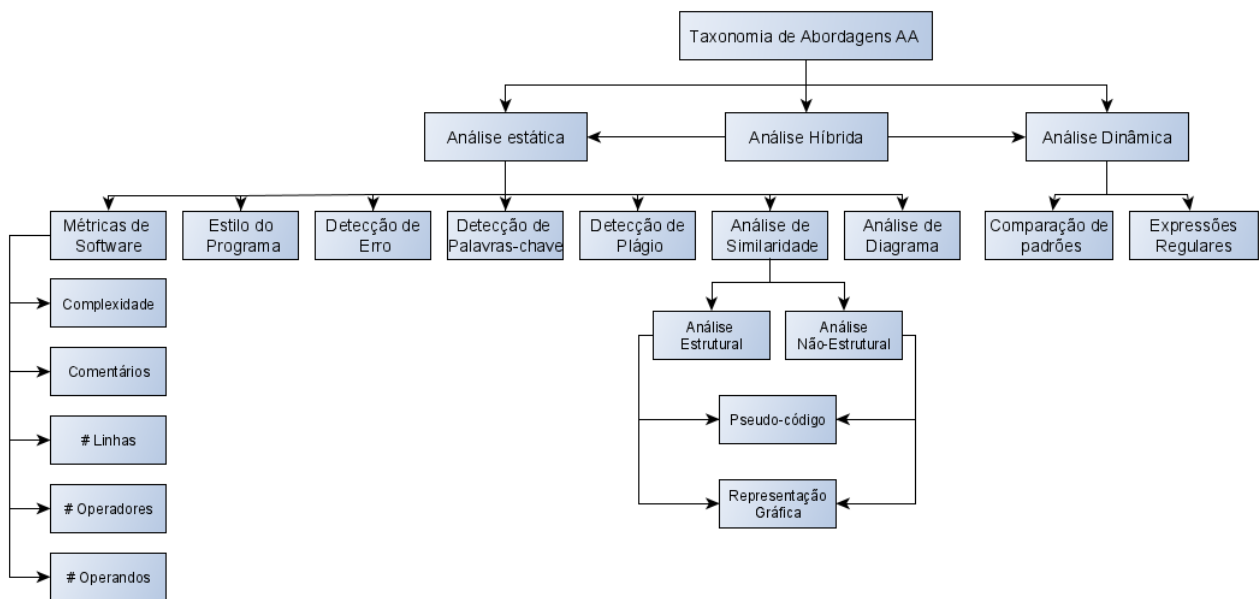
Além de listar os 17 principais sistemas que utilizam avaliação automática de algoritmos, o trabalho de Ullah et al. (2018b) faz uma análise sobre a principal contribuição dos trabalhos relacionados aos métodos de avaliação automática (AA) de códigos fonte. Para Ullah et al. (2018b), a literatura contemporânea classifica os métodos de AA em: (i) dinâmicos, aqueles que contemplam a execução do código fonte; (ii) estáticos, aqueles embasados na análise de métricas de software, sem a necessidade da execução do código; e (iii) híbridos, uma combinação das abordagens estáticas e dinâmicas.

A Figura 2.1 apresenta a taxonomia de sistemas que utilizam a avaliação automática proposta por Ullah et al. (2018b). Para Ullah et al. (2018b), a análise estática faz uso de: (i) métricas de software para avaliar a qualidade do código usando atributos qualitativos como complexidade, análise de comentários, número de linhas, número de operadores, e número de operandos; (ii) analisadores de estilo para verificar a legibilidade do código; (iii) detecção de erros para encontrar erros sintáticos ou semânticos; (iv) detecção de palavras-chave para procurar palavras-chave no código do estudante; (v) detecção de plágio para garantir a originalidade do programa do

estudante; (vi) análise de similaridade estrutural e não estrutural a partir de pseudocódigo ou representações gráficas do código em comparação com uma resposta modelo, e a (vii) análise de diagramas para comparar os fluxogramas do código do estudante e de um código modelo.

Ainda segundo Ullah et al. (2018b) a análise dinâmica faz uso de comparação de padrões e expressões regulares para avaliar a saída de um código. Comparação de padrões é usada para comparar resultados, enquanto que as expressões regulares são cadeias especiais de texto, as quais representam um padrão de procura, o que permite a simplificação da procura pelo resultado na saída do código, ou seja, a saída não precisa ser exatamente igual ao padrão, basta que contenha o padrão procurado.

Figura 2.1: Taxonomia das abordagens de AA



Fonte: Adaptado de Ullah et al. (2018b, p.4) - traduzido pelo autor

Arifi et al. (2015) apresentam as principais características dos sistemas de avaliação dinâmica. Sistemas de Avaliação Dinâmica (AD) de códigos fonte comparam se as saídas geradas são correspondentes àquelas esperadas, dado um conjunto de entradas, isto é, verifica se a resposta gerada é a correta. Sistemas que utilizam AD fazem a avaliação utilizando a comparação direta de *strings* (uma cadeia de caracteres de texto), ou através do uso de expressões regulares, ou ainda, através da verificação do estado da memória em um dado momento. Esses sistemas são amplamente utilizados em AA, dada a simplicidade da avaliação dos resultados, apenas considerando os algoritmos como corretos ou incorretos. Contudo, tanto Ullah et al. (2018b) e Arifi et al. (2015) apontam a incapacidade de avaliar algoritmos que não fornecem uma resposta correta como uma das desvantagens desse tipo de sistema.

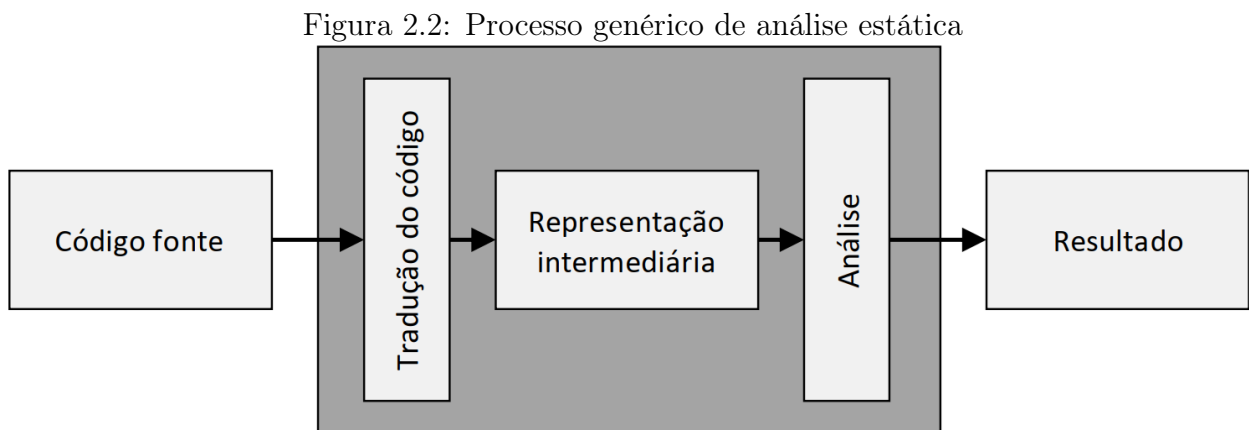
Nos trabalhos de Ullah et al. (2018b), Arifi et al. (2015) e Moreira e Favero (2009) as características de uma Avaliação Estática (AE), como instrumento de ensino, são apontadas como sendo vantajosas em relação à dinâmica. Através da AE é possível avaliar os códigos fonte que possuem erros que os incapacitem de serem executados. Ainda, é possível determinar

sua eficiência através da análise de métricas de software, tais características fornecem uma ferramenta importante para a compreensão e extração de informações inclusive de códigos fonte que possuem erros.

Segundo Alves (2013) pode-se distinguir duas abordagens para a análise estática:

1. Análise de similaridade estrutural: que tem por objetivo verificar como a estrutura da solução do aluno se compara à uma solução modelo, e nesse caso o *feedback* ao aluno é o resultado da comparação.
2. Análise de métricas de software: uma análise quantitativa, baseada em métricas de complexidade de software e diretrizes de boas práticas de programação para avaliar a qualidade das soluções dos alunos. Também pode ser parametrizada segundo uma rubrica, que é um conjunto de regras para determinar o resultado de uma avaliação, como por exemplo aferir critérios de uso e aplicação de conceitos de programação ligados ao Pensamento Computacional.

Ainda segundo Alves (2013), a realização do processo de análise estática consiste na análise léxica e sintática (*parsing*) do código fonte, avaliando-se os elementos relacionados segundo as métricas definidas. A partir do *parsing*, o código é traduzido para uma representação intermediária, no passo seguinte são analisados e os resultado apresentados, conforme Figura 2.2.



Fonte: Alves (2013) - Adaptado pelo autor

No trabalho de Oliveira (2018) são mapeadas diversas abordagens de avaliação de aprendizado a partir de códigos fonte. O artigo destaca as técnicas de avaliação da complexidade de algoritmos de Halstead (1977) e a complexidade ciclomática de McCabe (1976). Ambas as técnicas são citadas em diversos trabalhos que envolvem a análise de algoritmos, a primeira utiliza dados análogos a medidas espaciais para calcular a complexidade, a segunda utiliza uma análise similar ao estudo de grafos, preocupando-se com os diferentes caminhos que uma execução do código pode tomar.

Além da acurácia, isto é, a capacidade do algoritmo gerar o resultado esperado, e a eficácia, isto é, a capacidade de chegar ao resultado esperado com o melhor desempenho possível na elaboração de um código fonte, outros aspectos presentes são objetos de estudos educacionais. Em um trabalho que apresenta uma pesquisa ampla sobre as metodologias de avaliação automática e semiautomática de códigos fonte, Ala-Mutka (2005) faz considerações quanto às diferentes finalidades de uma avaliação automática, as dificuldades encontradas por educadores no uso dessas ferramentas, e os pontos fortes e fracos de suas aplicações.

Segundo Ala-Mutka (2005), muitas abordagens para a avaliação automática de programação podem ser encontradas em artigos, conferências e outras fontes. O requisito básico para a análise automática é que alguns tipos de valores de medição possam ser extraídos do código, e que os valores extraídos possam ser comparados a um dado requisito ou a uma solução modelo. Para o uso educacional, esses valores também precisam ser justificados por um objetivo de ensino.

Sistemas que utilizam a avaliação automática de códigos fonte são frequentemente utilizados não só pela praticidade da avaliação, mas também pela capacidade de serem utilizados como ferramentas de organização para os professores. Muitos desses sistemas também oferecem outros recursos como a possibilidades de criar-se repositórios de problemas, receber submissões das atividades resolvidas por alunos de programação e mesmo a atribuir notas. Esses sistemas atendem à uma necessidade de organização da prática de ensino de professores de programação, oportunizando o acompanhamento e *feedback* assíncrono das atividades propostas, conferindo assim vantagens como consistência e disponibilidade (ALA-MUTKA, 2005).

Algumas das plataformas mais proeminentes que possuem avaliação automática de códigos fonte são os *Online Judges*, que são sistemas que possuem um repositório de problemas de programação, aos quais os usuários podem submeter suas soluções escritas em uma linguagem de programação disponível. Esses ambientes começam a ser cada vez mais utilizados por instituições de ensino para auxiliar seus estudantes no desenvolvimento de suas atividades (WASIK et al., 2018).

Online Judges, como a plataforma URI Online Judge (BEZ et al., 2014) possuem uma classificação de seus problemas em categorias e níveis de dificuldade atribuídas pelos autores. Nessa plataforma, professores podem formar turmas e atribuir listas de exercícios a serem resolvidos por seus alunos. Outras plataformas como a Codeforces (MIRZAYANOV, 2015) tem o foco na promoção de competições em programação. Ambas plataformas possuem ambientes de avaliação automática, nos quais os usuários podem submeter soluções em diferentes linguagens de programação e receber um *feedback* sobre sua resposta. Os *feedbacks* gerados são resultado de uma avaliação dinâmica, e contemplam: a acurácia da submissão (aceito ou não aceito), em caso de não aceito informam o tipo de erro (compilação, execução, apresentação, etc.), o tempo de execução, o total de memória utilizada e os resultados dos casos de teste (Codeforces apenas).

Desta forma, observa-se que essas plataformas oferecem como *feedback* ao usuário um resultado binário, se a solução está correta ou não. Esse tipo de avaliação, não é similar a avaliação humana de um professor de programação, a qual engloba outros aspectos discutidos nas diversas teorias de aprendizagem contemporâneas, as quais não são objeto do presente

trabalho. Ala-Mutka (2005) discute a subjetividade necessária à avaliação educacional, trazendo à luz não só as vantagens já citadas dos sistemas de avaliação automática, mas também a necessidade de contemplar outros aspectos associados ao aprendizado de programação.

Para Ala-Mutka (2005), como estruturas bem definidas, programas de computador expressos por seus códigos fonte são elegíveis para a avaliação automática, assim, muitas abordagens de avaliação têm sido relatadas na literatura tanto para a avaliação dinâmica quanto para a avaliação estática. A avaliação automática é geralmente reconhecida por oferecer um resultado mais rápido, consistente, absolutamente objetivo e reutilizável sem ônus adicional tanto para educandos quanto para educadores. No entanto, nem todos os aspectos relacionados à programação podem ser avaliados automaticamente. Uma abordagem semiautomática pode ser usada para medir alguns aspectos automaticamente, enquanto outros devem ser medidos por uma avaliação manual. A seguir discute-se alguns aspectos sobre a avaliação no contexto do ensino de programação de computadores.

2.1.2 Avaliação e *feedback*

A avaliação é uma etapa importante do processo educativo, que comumente tem como objetivo aferir se o desempenho do estudante é adequado ao desenvolvimento esperado, ou seja, busca-se verificar se o sujeito avaliado demonstra as competências desejáveis. Portanto, quando objetiva-se avaliar um aluno, é necessário ter uma definição sobre os conceitos de habilidades e competências. Pode-se entender competências como o domínio global ou prático de uma situação cotidiana, e habilidades como o domínio de uma operação específica ou ações que atendam à uma ou mais competências (PERRENOUD, 1999).

Em abordagem similar Brasil (2005) define competência como uma habilidade de ordem geral, e habilidade como uma competência de ordem particular, ou seja, uma competência seria a resolução de problemas; enquanto que as habilidades seriam saber utilizar recursos para resolver determinados problemas. Uma competência seria constituída de várias habilidades, mas uma habilidade não pertence à determinada competência apenas, uma vez que a mesma habilidade pode contribuir para competências diferentes.

Quanto a avaliação em um contexto educacional, considera-se que: “... sua principal função deve ser a de permitir a análise crítica da realidade educacional [...], a descoberta de problemas novos, de novas necessidades e dimensões possíveis” (KENSKI, 1996). Dessa forma, a avaliação consiste em um processo diagnóstico (LUCKESI, 1986), isto é, visa identificar elementos positivos e negativos do processo educacional, possibilitando uma adequação no direcionamento do mesmo.

Naturalmente, o resultado da avaliação deve retornar uma informação pertinente à um dos sujeitos, educador ou aluno, ou ambos, chama-se esse retorno de *feedback*. O *feedback* gerado por uma avaliação humana está permeado por outros elementos, além dos ligados exclusivamente ao conteúdo da disciplina em questão. Para Kenski (1996), a avaliação também está permeada por juízos provisórios, relacionados a outros aspectos exteriores ao ambiente educacional, ou mesmo intrínsecos às idiossincrasias dos indivíduos envolvidos, como valores morais e empatia.

Ainda segundo Kenski (1996), o caráter dinâmico e subjetivo de formulação desses juízos provisórios faz com que o professor emita avaliações diferentes mesmo em situações aparentemente semelhantes. Neste sentido, pode ocorrer que um professor julgue diferentemente dois alunos que passaram por situações semelhantes apresentando o mesmo resultado. Vão estar presentes na efetivação dessas avaliações todo um forte contingente de elementos ligados à individualidade do professor, às relações existentes entre ele e cada um dos alunos, e às condições histórico-concretas em que ocorrem cada uma dessas situações.

Em uma avaliação humana o uso de uma rubrica pode mitigar os elementos subjetivos que causam a variabilidade no retorno da avaliação. A rubrica é um conjunto de regras que definem critérios padronizados para determinar o resultado de uma avaliação (ALBANO, 2017). Observando-se essa característica, estabelecer um conjunto de regras que padronizam o resultado gerado por uma avaliação, observa-se que os modelos de avaliação automatizadas contemplam o uso de uma rubrica, ainda que não o façam formalmente, pois são elaborados segundo um conjunto de regras bem definidas que geram um resultado.

Albano (2017) classifica as rubricas em dois tipos: analíticas ou holísticas. Rubricas analíticas são aquelas que dividem uma resposta em partes menores, categorias ou componentes, cada um dos quais pode ser avaliado de forma diferente. Por exemplo, uma redação pode ser dividida em introdução, corpo do texto e conclusão, e cada uma dessas seções pode ser avaliada individualmente. Rubricas holísticas são aquelas que fornecem um único resultado com base no desempenho geral da avaliação da tarefa. Rubricas que analisam os componentes de uma solução separadamente são mais difíceis de implementar do que as holísticas, mas tem como vantagem fornecer maiores detalhes na avaliação, proporcionando um mapeamento mais apurado sobre potenciais forças e fraquezas das competências avaliadas.

Entre outras reflexões, destaca-se a diferença de propósito entre avaliar um código fonte em um contexto geral e em um contexto educacional. Ambos buscam determinar a acurácia e eficácia de um código fonte, o primeiro com o propósito de estabelecer se o código é satisfatório à solução de um dado problema, o segundo com o propósito de aferir o desenvolvimento das habilidades e competências desejadas aos estudantes de programação.

Dessa forma, em um contexto educacional, entende-se que é desejável que um sistema de AA contemple não apenas uma avaliação pontual de desempenho, mas forneça elementos capazes de identificar potenciais fraquezas na evolução das competências necessárias a alunos de programação, oportunizando ações que busquem o seu desenvolvimento.

Portanto, observa-se que existem oportunidades de expandir os métodos de avaliação automática para contemplarem elementos que ofereçam outros subsídios que auxiliem na identificação e monitoramento da evolução de habilidades e competências necessárias ao ensino e aprendizado de programação de computadores.

2.2 Relevância e TF-IDF

Não há uma definição universal para informação, mas pode-se simplificar esse conceito como algo que possui uma representação simbólica, uma organização estruturada, e que possa ser entendida por um sujeito (BOYCE et al., 2017). Ainda segundo Boyce et al. (2017), sistemas de recuperação de informações são aplicações, computacionais ou não, que buscam recuperar uma informação desejada em uma coleção de dados.

Sistemas de recuperação de informações a partir de dados textuais frequentemente se utilizam de técnicas de balanceamento de termos (ROBERTSON; JONES, 1994). O balanceamento é um processo que atribui pesos que reflitam as distintas relevâncias de um termo para a recuperação de uma informação em particular.

Assim como a informação, o conceito de relevância não atende uma única descrição, e é sujeito à interpretação. Para Saracevic (2006), as técnicas de recuperação de informações são bem definidas, enquanto que a informação não é. Da mesma forma, enquanto que as técnicas de informação são tangíveis, a relevância não é. As técnicas de informação são entendidas formalmente, enquanto que a relevância é entendida intuitivamente. Técnicas de informação devem ser aprendidas, enquanto que a relevância é implícita. Técnicas de informação devem ser explicadas às pessoas, enquanto que a relevância não tem essa necessidade.

Em seu trabalho Saracevic (1975) apresenta uma revisão sobre a evolução do conceito de relevância na ciência da informação. Esse trabalho original foi reelaborado e expandido em seus trabalhos Saracevic (2006) e Saracevic (2007). Fundamentalmente, a relevância é uma tupla, entendida pela relação entre suas diferentes partes: tangíveis como documentos, máquinas e processos; intangíveis como ideias, conceitos e informações; propriedades como topicalidade, utilidade e especificidade (SARACEVIC, 2006).

Observa-se que determinar a relevância de um termo para a recuperação de uma informação compreende determinar a relação do termo com suas propriedades. A especificidade é um dos elementos centrais das ciências de informação, é entendida como uma capacidade discriminatória, ou seja, a habilidade de distinguir uma classe de informações de outra, de forma a satisfazer algum critério de relevância, o qual é baseado na necessidade de obter essa informação (KIM, 2006).

Kim (2006) apresenta uma análise da relação entre a relevância e quatro interpretações de especificidades de um termo, estabelecendo as correlações entre os dois conceitos, e concluindo que são relacionados. Ainda segundo Kim (2006), visto que o julgamento de uma relevância é dado pela eficácia de sistema em recuperar informações, a especificidade também é um fator importante para o sucesso desses sistemas.

Entende-se que a determinação da especificidade é a atribuição de diferentes pesos. Para Boyce et al. (2017), a ideia geral de atribuir pesos à procura de termos em textos, é de melhorar o poder de discriminação da procura por melhorar uma importância relativa de um termo em uma consulta. Esses pesos também podem ser utilizados para classificar documentos quanto à sua semelhança à uma consulta ou mesmo outro documento. Alguns sistemas de recuperação de

informações também fazem uso da atribuição de pesos para termos na indexação de documentos, ou para indicar a relevância aplicada a um termo em uma matriz de termo-documento, que é uma forma de organizar conjuntos de documentos, onde cada documento corresponde a uma linha e cada termo presente nos documentos corresponde a uma coluna.

Ainda segundo Boyce et al. (2017), é possível criar uma consulta que consiste em uma sequência de termos ponderados e um limite de relevância. A presença de tal termo em um documento, adiciona o peso do termo a um acumulador para esse documento, e se houver termos suficientes presentes, em que a soma de seus pesos exceda o limite estabelecido, o documento também é considerado relevante para a procura e movido para o conjunto de respostas desejáveis. Ainda, o documento pode ser classificado segundo o valor pelo qual cada documento excede o limite estabelecido.

Existem diferentes metodologias utilizadas para atribuição de pesos, podemos distingui-las como sendo manuais ou automáticas. Inicialmente as ciências da informação se utilizavam de atribuições manuais para a especificidade, como uma atribuição booleana se o termo é relevante ou não, ou mesmo *rankings* dos termos. Esse tipo de interpretação se dá a partir de uma interpretação semântica de um avaliador, e é considerada uma tarefa onerosa, subjetiva e imprecisa (BOYCE et al., 2017).

Em seu trabalho, Jones (1972) propôs uma mudança da interpretação da especificidade de um termo para a sua relevância em um conjunto de dados. Para essa especificidade, ora estimada através de uma atribuição de uma relevância semântica, a autora propôs a mudança para uma medida estatística dada por uma função de uso do termo.

A TF-IDF é uma metodologia de recuperação de informações frequentemente utilizada para textos escritos em linguagem natural para a identificar a relevância de uma palavra ou expressão em um documento contido em uma coleção. Utiliza-se da combinação de duas estatísticas: a TF (*Term Frequency* ou frequência do termo) que determina a importância do item em um documento, de forma que quanto maior a frequência do termo no documento maior sua relevância; e a IDF (*Inverse Document Frequency* ou frequência inversa do documento) que determina a importância do termo na coleção, de forma que quanto maior a raridade do termo na coleção maior sua relevância (SALTON; BUCKLEY, 1988).

A TF-IDF é dado pelo produto da frequência do termo (TF - *Document Frequency*), apresentado na Equação 2.1 e da frequência inversa do documento (IDF - *Iverse Document Frequency*), apresentado na Equação 2.2, onde f representa a frequência de um termo no documento, T o número total de termos no documento, N o número total de documentos na coleção e n o número total de documentos no qual o termo está presente (SALTON; BUCKLEY, 1988).

$$TF = \frac{f}{T} \quad (2.1)$$

$$IDF = \log \left(\frac{N}{n} \right) \quad (2.2)$$

Desde então, TF-IDF desenvolveu-se e consolidou-se como um dos principais modelos de balanceamento de termos e determinação de sua relevância. Sua fundamentação teórica também consolidou-se conforme apresentado no trabalho de Robertson (2004), e sua equação serviu de inspiração à variações que permitam uma melhor aplicabilidade, como por exemplo a Equação 2.3 de Boyce et al. (2017).

$$Weight_{i,j} = Freq_{i,j} \left(\log_2 \left(\frac{n}{DocFreq_j} \right) + 1 \right) \quad (2.3)$$

onde $Weight_{i,j}$ é o peso do termo j no documento i ; $Freq_{i,j}$ a frequência de ocorrências do termo j no documento i ; n o número de documentos na base de dados; e $DocFreq_j$ o número de documentos nos quais o termo j está presente.

Métodos de recuperação de informações são amplamente utilizados na análise de textos em linguagem natural, mas também podem ser aplicados a outras formas de representação de informações, como áudio, vídeos, imagens e linguagens de programação. A seguir, apresenta-se alguns casos da aplicação de TF-IDF para análise de códigos fonte escritos em uma linguagem formal de programação.

2.2.1 TF-IDF na Avaliação de Códigos Fonte

Em uma abordagem ampla, códigos fonte são qualquer descrição executável de um sistema de software, e, portanto, compreende códigos de máquina, linguagens de alto nível e representações gráficas de sistemas (HARMAN, 2010). Em uma abordagem específica, códigos fonte são versões de software na versão originalmente escritas (i.e, digitadas em um computador) por um humano em texto plano (i.e., caracteres alfanuméricos legíveis aos humanos)(LINUX, 2005).

O conceito relacionado à definição de códigos fonte está relacionado com a definição de linguagem programação. Uma linguagem de programação é uma notação na qual podemos transmitir instruções ao computador, códigos fonte escritos em uma linguagem de programação não se diferem estruturalmente de uma linguagem natural, possuem uma sintaxe bem definida, ou seja, contém regras de como seus símbolos podem ser combinados para formar declarações e expressões; bem como uma semântica bem definida, isto é, regras que atribuem um significado a uma ou mais combinações de diferentes estruturas presentes na notação (LOUDEN; LAMBERT, 2011). Visto a semelhança das linguagens de programação de alto nível e das linguagens naturais, a aplicabilidade das técnicas de recuperação de informação advindas do processamento de linguagem natural é uma alternativa válida e viável, gerando resultados satisfatórios conforme apresentado nos trabalhos a seguir.

Uma das aplicações recorrentes de sistemas de recuperação de informações na análise de códigos fonte está no processo de identificação de plágio. Para esse propósito a TF-IDF é utilizada ora como ponto central, como no trabalho de Karnalim (2020) que propôs um modelo inspirado em TF-IDF para a identificação de plágio entre códigos fonte escritos em diferentes linguagens de programação; ora como parte de uma metodologia mais ampla, na qual a TF-IDF é utilizada como balanceador para as relevâncias de termos, como no trabalho de Fu et al.

(2017).

O potencial dos modelos de recuperação de informações para a construção de bibliotecas de software é bem reconhecido (MAAREK et al., 1990). A TF-IDF também se faz presente em modelos que buscam identificar e sumarizar trechos de códigos fonte quanto à sua função, como nos trabalhos de Lucia et al. (2012) e Haiduc et al. (2010), nesses modelos a TF-IDF é utilizada para determinar a relevância dos termos quanto à uma aplicabilidade e, em conjunto com outras medidas estatísticas, obtém resultados satisfatórios na sumarização de códigos fonte.

O trabalho de Yang et al. (2016) se utiliza da TF-IDF como parte de seu modelo para recomendar trechos de códigos fonte a desenvolvedores quanto à requisitos estabelecidos ao desenvolvimento, assim como no modelo proposto por Divya et al. (2014) para verificação se um código fonte atende a requisitos propostos. Antoniol et al. (2002) propõem um modelo para recuperar elos de rastreamento entre códigos fonte e documentação em texto livre, da mesma forma que nos trabalhos anteriores, a TF-IDF é utilizada para determinar a relevância de termos.

Com base nesses estudos apresentados, verifica-se que a análise de códigos fonte possui similaridades significativas com a análise de texto em linguagem natural, seja para a identificação de requisitos, usabilidade ou documentação. De forma geral entende-se que é possível utilizar a TF-IDF para determinar a relevância de termos ou expressões em um código fonte, e nos trabalhos apresentados observou-se sua utilização com sucesso.

2.3 Transformação do código fonte em uma representação numérica

Conforme definiu-se nas seções anteriores, o conceito de informação pode ser simplificado como uma representação simbólica, estruturada e que pode ser entendida (BOYCE et al., 2017). Também definiu-se que em uma abordagem mais específica códigos fonte são versões de software representados por caracteres alfanuméricos (LINUX, 2005). Dessa forma, pode-se entender códigos fonte como representações simbólicas de informações, ou conjuntos de informações, estruturadas escritas em uma linguagem de programação.

A representação simbólica presente nos códigos fonte é realizada de forma a facilitar a interpretação humana. Para o computador os símbolos representando o texto do código são bytes compostos por cadeias de bits, e em seu mais baixo nível toda a informação presente em um código fonte é transformada em uma linguagem binária (BOYCE et al., 2017). Da mesma forma a tabela ASCII (*American Standard Code for Information Interchange*) é utilizada para representar numericamente os símbolos alfanuméricos utilizados na maioria das linguagens de programação. Se considerarmos o código fonte apenas como texto plano, cada símbolo presente poderia ser transformado em um número correspondente, e preservando-se a ordem as informações presentes também seriam preservadas.

As linguagens de programação também possuem uma estrutura léxica, semelhante às palavras de linguagem natural, que são normalmente chamados de *tokens*. Assim, a estrutura léxica é similar à gramática em uma linguagem natural. Como por exemplo na linguagem C as palavras

if e *else* são tokens, e possuem um significado próprio, sendo reservado para uso específico. Outros tokens em linguagens de programação incluem identificadores (ou nomes), símbolos para operandos lógicos e relacionais, como “+” e “==” e símbolos de pontuação especiais como o ponto e vírgula (;) e o ponto final (.) (LOUDEN; LAMBERT, 2011).

Considerando-se os identificadores *SOMA*, *PRIMEIRO* e *SEGUNDO*, assim como os operadores + para soma e = para atribuição na expressão:

$$SOMA = PRIMEIRO + SEGUNDO \quad (2.4)$$

Se considerarmos ainda apenas a gramática da linguagem, atribuindo-se a uma numeração para os elementos presentes no código de exemplo, conforme a Tabela 1, teríamos uma representação numérica separada por vírgulas da seguinte forma:

$$55, 65, 55, 60, 55 \quad (2.5)$$

Tabela 1: Exemplo de tokens

Token	Número
identificador	55
operador aritmético	60
operador atribuição	65

Fonte: elaborado pelo autor

As informações que se deseja representar em um código fonte devem ser estruturadas de forma a respeitar um regramento sintático de acordo com a linguagem de programação utilizada. A sintaxe de uma linguagem é a descrição de como as diferentes partes da linguagem podem ser combinadas para formar frases e sentenças, em uma linguagem de programação formam declarações e expressões. O conjunto de regras que regem o regramento das estruturas léxicas e sintáticas são chamados de gramática da linguagem (LOUDEN; LAMBERT, 2011).

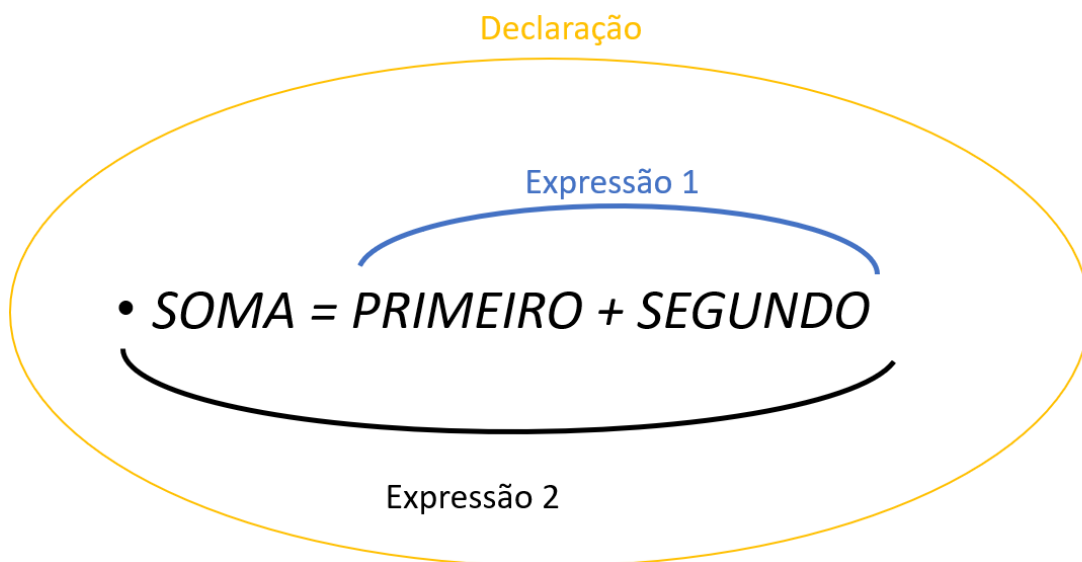
A sintaxe é apenas uma das camadas da estrutura de uma linguagem. A semântica é a interpretação que dá sentido ou significado a um trecho de código, um conceito muito mais difícil de ser explicado precisamente. Frequentemente, explicar o sentido de um trecho de código envolve descrever sua execução. O sentido atribuído a um conjunto de tokens que formam um trecho particular de código pode envolver interações com outros trechos, portanto uma descrição que compreenda todo o seu significado pode se tornar extremamente complexa (LOUDEN; LAMBERT, 2011).

Boyce et al. (2017) apresentam como exemplo de linguagens que requerem interpretação de contexto, as linguagens de comandos para um sistema computacional, seja para um propósito geral de computação, como *C++* ou *BASIC*, ou para um uso especializado, como para sistemas de bancos de dados como *APPROACH*, ou *DIALOG*. Essas linguagens possuem uma sintaxe, e existe pelo menos algum nível de dependência de contexto. Isso significa que um símbolo nem sempre carrega o mesmo significado, seja onde e como ele ocorra, o contexto determinará o seu

significado. Por exemplo, muitas linguagens de programação utilizam-se de aspas, assim como nós fazemos nas linguagens naturais, para alterar uma interpretação. *WRITE*("X + Y") em linguagens de programação tipicamente significa escrever os símbolos "X", "+" e "Y", enquanto que *WRITE*(X + Y) significa escrever a soma dos valores das variáveis X e Y.

No exemplo da *SOMA* utilizado anteriormente, observa-se pelo menos duas expressões. *PRIMEIRO* + *SEGUNDO* compõem uma expressão. A segunda expressão é composta pelo resultado da primeira expressão e a atribuição com *SOMA*, conforme a Figura 2.3. A união das duas expressões compõem ainda uma declaração, o sentido desse trecho poderia ser descrito como: soma-se o valor presente no identificador *PRIMEIRO* ao valor do identificador *SEGUNDO*, após atribui-se o resultado dessa soma ao identificador *SOMA*.

Figura 2.3: Exemplo de expressões



Fonte: elaborado pelo autor

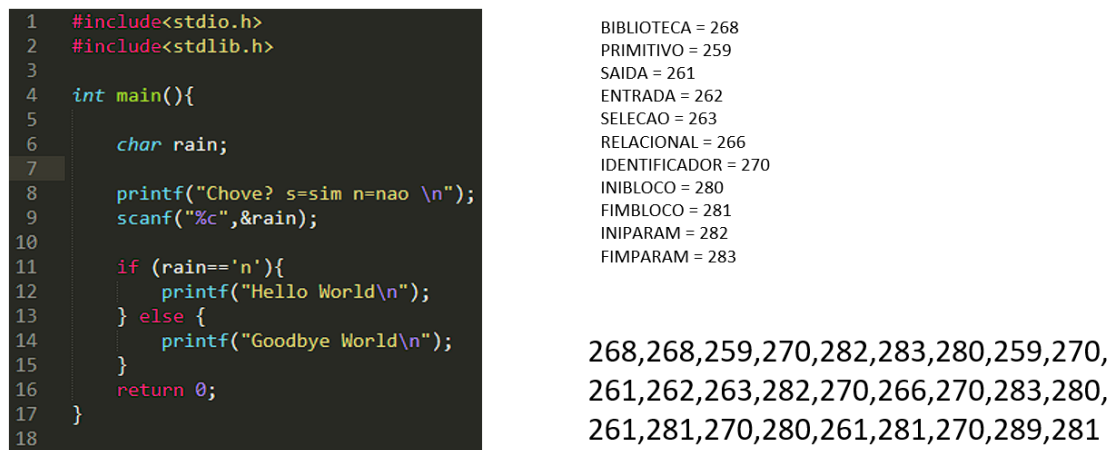
Dessa forma, existem dois aspectos que podem ser observados nas estruturas presentes em códigos fonte: a definição e o sentido. A definição está relacionada à sintaxe estrutural esperada, por exemplo, a declaração de um comando de seleção "Se" poderia ser definida como: Um "Se" consiste da palavra "if" seguida por uma expressão dentro de parênteses, seguida por uma declaração, seguida por uma parte "Se não" opcional que consiste na palavra "else" seguida por outra declaração (LOUDEN; LAMBERT, 2011).

O sentido está relacionado à conotação semântica da organização estrutural definida. Por exemplo, o sentido do comando "Se" pode ser descrito como: o "se" é executado em seu primeiro momento avaliando sua expressão, que deve conter um tipo aritmético ou ponteiro, incluindo todos seus efeitos colaterais, e se ele for diferente de zero, a declaração seguinte a expressão é executada. Se houver uma parte "se não", e a expressão for igual a zero, a declaração seguinte

ao “se não” é executada (LOUDEN; LAMBERT, 2011).

Observa-se então que em um código fonte é possível identificar não só elementos definidos pela gramática, os tokens, mas também sentido semântico a partir da interpretação de suas combinações. Da mesma forma que atribui-se uma representação numérica para tokens, pode-se atribuir uma representação numérica para declarações. A Figura 2.4 apresenta um exemplo de código fonte que contém representações numéricas tanto de tokens quanto de expressões e declarações.

Figura 2.4: Transformação léxica e semântica em representação numérica



Fonte: elaborado pelo autor

A análise estática de códigos fonte se utiliza amplamente de modelos de recuperação de informação. Esses modelos possuem estratégias de identificação dos diferentes tokens presentes nos códigos, bem como transformações que possibilitem a aplicação de ferramentas estatísticas conforme observado nos trabalhos a seguir.

O trabalho de (AZCONA et al., 2019) demonstra esses processos de transformação de dados, para posterior uso em um modelo de classificação de perfis de estudantes. A informação presente nos códigos fonte originais é convertida em diagramas de frequência de termos, um processo chamado de tokenização.

O trabalho de Ullah et al. (2018a) demonstra a utilização de técnicas de análise de texto e recuperação de informações em códigos fonte, utilizando-se do processo de transformação do código em tokens. No trabalho de Ullah et al. (2018a), a preparação desses tokens passa pelos processos de *lemannization*, *stemming*, remoção de números e remoção de *stop words*. Após o balanceamento da matriz de tokens, aplica-se a técnica de decomposição em valores singulares (SVD) e cálculo da análise de semântica latente (*Latent Semantic Analysis* - LSA)

O trabalho de Ganguly et al. (2018) também propõe o uso de técnicas de análise textual aplicadas a um código fonte para fim de identificação de plágio. Nele também é utilizada uma

matriz de frequência de tokens e, seguindo a nomenclatura presente na literatura de análise textual, o autor nomeia o conjunto de tokens selecionados como *Bag of Words* (BoW). Nesse trabalho, além da frequência dos termos o autor utiliza-se da análise de uma árvore de sintaxe abstrata (*Abstract Syntax Tree* – AST) a qual considera o contexto presente no código fonte.

Em um primeiro momento, no trabalho de (MOREIRA; FAVERO, 2009), é proposta uma avaliação que utiliza as métricas de software (HALSTEAD, 1977; MACCABE, 1976) aplicadas em uma regressão linear múltipla. O objetivo é gerar um *score*, i.e., uma nota, e compará-la à nota atribuída por um avaliador humano. Em seu experimento o autor utilizou uma base com 20 exercícios de programação solucionados por quatro estudantes, mais uma resposta modelo elaborada por um especialista, cada solução elaborada pelos educandos foi submetida à avaliação de um especialista, o qual atribuiu uma nota. As métricas de software foram extraídas das respostas e posteriormente inseridas em um algoritmo de regressão múltipla. As métricas de software são utilizadas como variáveis independentes, enquanto que a variável dependente é considerada a nota. O algoritmo de regressão múltipla gerou uma nota com até 90,42% de acerto em relação às avaliações do especialista.

No segundo momento do trabalho de (MOREIRA; FAVERO, 2009) foi utilizada a regressão linear múltipla com *N-grams*, outra técnica de análise de texto em linguagem natural, e da mesma forma o *score* foi comparado com a avaliação de um humano. A técnica de *N-grams* consiste em extrair um pedaço de *n*-caracteres de uma cadeia de texto, nesse caso de códigos fonte, que se combinado com técnicas de similaridade vetorial (tipo cosseno) permite comparar objetos tais como texto livre, ou códigos fonte. Nesse segundo experimento os autores não utilizaram as métricas de software, mas sim as incidências de unigramas, bigramas e trigramas como variáveis independentes da regressão linear, mantendo-se a variável dependente como a nota prevista. A regressão múltipla com *N-grams* teve uma média de acerto de 95,22% em relação às avaliações do especialista.

Observa-se que o ponto comum dos trabalhos apresentados é a busca de informações presentes a partir de códigos fontes, as diferenças de modelos aplicados refletem também as diferenças de quais informações deseja-se obter e como deseja-se utilizá-la. No contexto de obter-se informações acerca do desenvolvimento acadêmico de alunos de programação de computadores a partir da análise de códigos fonte, torna-se relevante obter-se informações sobre habilidades e competências a partir da observação dos códigos. Assim, buscou-se utilizar como referência um conjunto de competências já estabelecidos, escolheu-se as competências do Pensamento Computacional conforme será explicado nas seções seguintes.

2.4 Pensamento Computacional

O Pensamento Computacional (PC) trata do conjunto de habilidades relacionadas à uma abordagem de resolução de problemas análoga ao modo como um computador opera, i.e., “O processo de pensamento envolvido na formulação de um problema e na expressão de sua solução de forma que um computador – humano ou máquina – possa efetivamente realizar” (WING,

2006).

Ainda segundo Wing (2006), o processo de construção do Pensamento Computacional baseia-se em três estágios: formulação de um problema (abstração), expressão de uma solução (automação) e a execução da solução e análise (análise). A Sociedade Brasileira de Computação descreve esses estágios em seu itinerário formativo da computação (SBC, 2020) como:

1. Abstração - Resolução de problemas, representação da informação e processos:

Compreender e utilizar modelos e representações adequadas para organizar informação e descrever processos. Usa diferentes técnicas (decomposição, generalização, transformação, reuso, recursão, refinamento) para construir soluções para problemas, identificando vantagens do uso destas técnicas e as condições para utilizá-las, aperfeiçoando e articulando saberes escolares.

2. Automação - Programas e dados:

Identificar problemas de diversas áreas do conhecimento e criar soluções, de forma individual e colaborativa, usando computadores, celulares e/ou outras máquinas processadoras de instruções para resolver todo ou parte do problema. Criar modelos computacionais para simular e fazer previsões sobre diferentes fenômenos e processos.

3. Análise - Análise quantitativa e qualitativa de problemas/soluções:

Analisar criticamente os problemas e soluções para identificar não somente se existem soluções que podem ser automatizadas, mas também ser capaz de avaliar a eficiência e a correção destas soluções e de justificar tanto a adequação das soluções aos requisitos quanto às escolhas frente às diferentes possibilidades de soluções.

A definição para o Pensamento Computacional apresentada por (RAABE et al., 2018) consiste na capacidade de sistematizar, representar, analisar e resolver problemas considerando conhecimentos e práticas da computação. Ainda segundo (RAABE et al., 2018) as principais competências do Pensamento Computacional são:

1. Abstração: Este conceito envolve a filtragem dos dados e sua classificação, ignorando elementos que não são necessários, visando os que são relevantes. Envolve também formas de organizar informações em estruturas que possam auxiliar na resolução de problemas.
2. Algoritmos: É um conceito que agrega todos os demais. O algoritmo é um plano, uma estratégia ou um conjunto de instruções claras e necessárias para a solução de um problema. Em um algoritmo, as instruções são descritas e ordenadas para que o objetivo seja atingido e podem ser escritas em formato de diagramas, pseudocódigo (linguagem humana) ou escritos em códigos, por meio de uma linguagem de programação.
3. Decomposição: A decomposição trabalha o processo pelo qual os problemas são divididos em partes menores e mais fáceis de resolver. Compreende também a prática de analisar

problemas a fim de identificar que partes podem separadas, e também de que forma podem ser reconstituídas para a solução de um problema global. Essa prática também possibilita aumentar a atenção aos detalhes.

4. Reconhecimento de Padrões: Trabalha a identificação de características comuns entre os problemas e suas soluções. Resulta do fato de realizar a decomposição de um problema complexo para encontrar padrões entre os subproblemas gerados. Estes padrões são similaridades ou características que alguns dos problemas compartilham e que podem ser explorados para que sejam solucionados de forma mais eficiente.

A abstração é a chave do Pensamento Computacional (WING, 2008), e sua presença nas discussões acerca do tema é constante. No entanto, as definições do conceito de abstração, e mesmo de outros conceitos e competências do Pensamento Computacional, não compartilham dessa mesma constância, e são desenvolvidos e discutidos em diversos trabalhos, os quais fazem observações e adaptações de acordo com suas necessidades. Como no trabalho de Lowe e Brophy (2017), que sumarizaram os principais conceitos presentes na literatura:

1. Abstração: identificar e extrair informação relevante para definir ideias centrais.
2. Decomposição: repartir dados, processos ou problemas em partes menores.
3. Reconhecimento de padrões: observar padrões, tendências e regularidades em dados.
4. Algoritmos: criar uma série ordenada de instruções para resolver problemas similares ou executar uma tarefa.
5. Coleção e representação de dados: reunir informações, organizar dados em gráficos, desenhos, palavras ou imagens apropriadas.
6. Paralelização: processar simultaneamente pequenas tarefas de uma tarefa maior, para assim atingir de forma eficiente um objetivo comum.
7. Simulação ou automação: desenvolver modelos para imitar processos do mundo real.

Entre as diversas discussões e estudos acerca do Pensamento Computacional no desenvolvimento de atividades humanas, destaca-se o consenso do potencial do PC como referência de competências para a educação, fazendo parte inclusive das Diretrizes para o Ensino da Computação no Brasil (BRASIL, 2016). O vínculo com as ciências da computação demonstra-se natural ao Pensamento Computacional, naturalmente existem trabalhos para avaliação de produções de alunos no âmbito da programação, como no trabalho de Alves (2013) em que a autora propõe um modelo para avaliação de atividades em uma linguagem de programação visual, o qual atribui uma pontuação ao algoritmo com base nas estruturas encontradas no código fonte, nesse caso realizado em uma linguagem com blocos lógicos.

Há também abordagens orientadas à construção de exercícios de programação que propõem conceitos do Pensamento Computacional ligados à estruturas presentes no código fonte, como por exemplo no trabalho de Brennan e Resnick (2012) que propõe os seguintes conceitos do Pensamento Computacional:

1. Sequências: são uma série de passos individuais que podem ser executados pelo computador que compõem uma atividade ou tarefa.
2. Laços: são mecanismos para executar uma mesma sequência múltiplas vezes.
3. Eventos: são ações que causam a execução de outras ações.
4. Paralelismo: são sequências de instruções executadas simultaneamente.
5. Condições: é a habilidade de tomar uma decisão, baseada em uma condição, que suporta múltiplos resultados.
6. Operadores: são elementos que oferecem suporte para expressões matemáticas, lógicas, e textuais, permitindo ao programador realizar manipulações numéricas e textuais.
7. Dados: são estruturas capazes de armazenar, recuperar e atualizar valores.

Além dos conceitos do Pensamento Computacional propostos, Brennan e Resnick (2012) também apontam a necessidade da avaliação das competências do Pensamento Computacional em outras duas dimensões: práticas computacionais e perspectivas computacionais. As práticas computacionais representam o processo de construção de um algoritmo. As práticas são definidas como:

1. Iterativo e incremental - faz alusão ao processo de identificar conceitos centrais e definir uma estratégia para resolvê-lo.
2. Teste e depuração - é um processo fundamental a qualquer programador, na qual o sujeito faz uso da abstração para verificar a correção da estratégia idealizada para solução.
3. Reuso e reformulação - faz alusão à prática de identificar trechos de códigos, próprios ou não, que podem ser utilizados novamente.
4. Abstração e modularização - corresponde ao ato de decompor o problema em partes menores.

As perspectivas computacionais estão ligadas ao modo de se expressar, conectar e relacionar com agentes externos através da programação, como agir colaborativamente com a comunidade (CAVALCANTE et al., 2016). A Tabela 2 apresenta um resumo do *framework* de Brennan e Resnick (2012).

Tabela 2: Framework para avaliação das competências do Pensamento Computacional

Competências do pensamento computacional		
Conceitos computacionais	Práticas computacionais	Perspectivas Computacionais
Sequências	Iterativo e incremental	Se expressar
Laços	Teste e depuração	Se conectar
Eventos	Reuso e Reformulação	Se questionar
Paralelismo	Abstração e Modularização	
Condicionais		
Operadores		
Dados		

Fonte: Adaptado a partir de Brennan e Resnick (2012)

Seiter e Foreman (2013) propõem uma modelagem do processo de aprendizado e progressão do Pensamento Computacional. Em seu trabalho são relacionados o que a autora chamou de variáveis de evidência (e.g. declaração de variáveis, uso de condicionais, laços de repetição, etc.) e habilidades do Pensamento Computacional como decomposição do problema, abstração e análise. Outros trabalhos como os de Chang (2018) e Moreno-León (2015), também estudam a relação entre elementos presentes em um código fonte e as habilidades do Pensamento Computacional.

Barr e Stephenson (2011) realizam um estudo de como inserir as competências do Pensamento Computacional no currículo do K-12. Nesse processo, com a ajuda de especialistas, relaciona as competências do Pensamento Computacional com diversas áreas do currículo, entre elas as ciências da computação, e nesse contexto com estruturas presentes em um código fonte. Nesse trabalho, os autores apresentam as capacidades ou atividades relacionadas de cada área relacionadas com nove conceitos elaborados a partir do Pensamento Computacional de Wing (2006). Entre essas áreas está a ciência da computação, na qual as capacidades ou atividades elencadas envolvem estruturas ou elementos que podem ser observados em um código fonte como uso de comandos de repetição, condicionais e definição de funções, conforme Tabela 3.

Tabela 3: Conceitos centrais do Pensamento Computacional e competências

Conceito, Competência do PC	Ciência da computação	Matemática	Ciências	Estudos Sociais	Linguagem e artes
Coleção de dados	Encontrar uma fonte de dados para um problema	Encontrar uma fonte de dados para um problema, por exemplo, lançar uma moeda ou jogar dados	Coletar dados para um experimento	Estudar estatísticas de uma batalha ou de dados de uma população	Fazer uma análise linguística de sentenças
Análise de dados	Escrever um programa para fazer cálculo estatístico básico em um conjunto de dados	Contar a ocorrência dos lançamentos de moeda ou dados e analisar os resultados	Analisar dados de um experimento	identificar tendências em dados estatísticos	Identificar padrões para diferentes tipos de sentenças
Representação de dados	Usar estruturas de dados como vetores, listas encadeadas, pilha, fila, grafos, etc.	usar histograma, gráfico de barras para representar dados, usar conjuntos listas gráficos para representar dados	Sumarizar dados de um experimento	Sumarizar e representar tendências	Representar padrões em diferentes sentenças
Decomposição do problema	Definir objetos e métodos; definir main e funções	Aplicar a ordem de operação em uma expressão	Fazer a classificação de uma espécie		Escrever um esboço
Abstração	Usar procedimentos para encapsular um conjunto de comandos que se repetem; usar condicionais, laços, recursão, etc.	Usar variáveis em álgebra; identificar fatos essenciais em um problema com palavras; estudar funções em álgebra e compará-las a funções em programação; usar iteração para solucionar problemas de palavras	Construir um modelo de entidades físicas	Sumarizar fatos; deduzir conclusões a partir de fatos	Usar uma metáfora simples; escrever uma história com ramificações
Algoritmos e procedimentos	Estudar algoritmos clássicos; Implementar um algoritmo para um problema	fazer divisões longas, fatoração, carregamentos em adição ou subtração	Fazer um procedimento experimental		Escrever instruções
Automação		usar ferramentas como: geometer sketch pad; star logo; python code snippets	Usar probeware	Usar o Excel	Usar uma corretor ortográfico
Paralelização	Threading, pipelining, dividir um dado ou tarefa de forma a processá-lo em paralelo	Resolver sistemas lineares; multiplicar matrizes	Executar experimentos simultaneamente com diferentes parâmetros		
Simulação	Animação de um algoritmo, varredura de parâmetros	Criar um gráfico de uma função em um plano cartesiano e modificar os valores de variáveis	Simular o movimento do sistema solar	Jogar Age of Empires; Oregon Trail	Fazer uma reconstituição de uma história

Fonte: Adaptado de Barr e Stephenson (2011) - traduzido pelo autor.

Tomando-se como referência as competências do Pensamento Computacional apontadas por Barr e Stephenson (2011), complementando-se com os conceitos apontados nos demais trabalhos, entende-se que as relações entre o Pensamento Computacional e tokens presentes em um código fonte podem ser definidas da seguinte forma:

1. Coleção de dados: estruturas e tokens que são utilizados para obtenção de dados.
2. Análise de dados: estruturas e tokens que são utilizados para realizar operações aritméticas.
3. Representação de dados: estruturas e tokens que representem informações ou um conjunto de informações.

4. Decomposição do problema: estruturas e tokens que fragmentem o algoritmo em partes menores reutilizáveis.
5. Abstração: estruturas e tokens que representem relações centrais a um trecho de código.
6. Algoritmos: estruturas e tokens que unidas componham um algoritmo para solução de uma tarefa em sua totalidade.
7. Paralelização: estruturas e tokens que permitam o processamento em paralelo de tarefas.
8. Simulação: estruturas e tokens que sejam utilizados para depuração e testagem do algoritmo.

O Pensamento Computacional não é sinônimo de habilidades em programação. O objetivo de apresentá-lo é definir um conjunto de competências relacionadas à estruturas presentes em um código fonte. Observou-se nos trabalhos apresentados que é possível relacionar tokens e conceitos do PC, no entanto observou-se que não há um consenso sobre quais estruturas estão relacionadas a quais conceitos. Por fim, apresentou-se um resumo das relações entre as principais estruturas presentes em um código e conceitos do Pensamento Computacional, esse entendimento inicial será utilizado como referência para a definição da metodologia que será apresentada nas seções seguintes.

3 UM MODELO PARA AVALIAÇÃO DE CÓDIGOS FONTE

Neste capítulo será apresentada a metodologia do trabalho, que consiste na abordagem utilizada para definição das competências e suas representações em um código fonte, na elaboração das matrizes de representação dos códigos fonte e rubrica de avaliação.

3.1 Competências e suas representações no código fonte

O objetivo do trabalho proposto é avaliar competências em programação de computadores a partir da análise estática de códigos fonte, portanto entende-se como um passo necessário estabelecer um conjunto de competências possíveis de serem identificadas em um código fonte. Conforme apresentado nas discussões do embasamento teórico, a literatura disponível não apresenta um consenso sobre quais são as competências a serem desenvolvidas por alunos em cursos de programação de computadores. Para o presente trabalho buscou-se utilizar um dos conjuntos de competências proeminentes na avaliação de códigos fonte. Escolheu-se como referências as competências do Pensamento Computacional, mais especificamente utilizou-se para essa referência os conceitos/competências elaborados por Barr e Stephenson (2011).

Entende-se que o conjunto de conceitos do Pensamento Computacional, que será utilizado como referência para um conjunto de competências, não é referência única de habilidades em programação, tão pouco é exclusivo a este fim. O escopo do presente trabalho não é definir quais são as competências de programação, ou mesmo quais são as habilidades do Pensamento Computacional, mas sim, dado um conjunto de habilidades relacionadas a elementos presentes em um código fonte, verificar se essas competências podem ser quantificadas ou estimadas a partir da análise estática de um código fonte.

Barr e Stephenson (2011) identificaram e relacionaram nove conceitos do Pensamento Computacional de Wing (2006) com exemplos de atividades segundo cinco áreas do conhecimento do K-12 (currículo da educação básica em alguns países de língua inglesa), das quais a ciência da computação faz parte. Essa relação foi escolhida como referência para um conjunto de competências a ser tratada neste artigo por possuir elementos facilmente identificáveis em um código fonte, relacionados às competências do Pensamento Computacional.

Ainda que o trabalho de Barr e Stephenson (2011) descreva elementos possíveis de serem identificados em códigos fonte, a posterior transformação desses elementos em uma representação numérica requer que eles sejam definidos de forma específica em relação à uma linguagem formal de programação, de forma a serem estabelecidos como parâmetros de identificação. Escolheu-se a linguagem “C” para referência do presente trabalho, e definiu-se as competências do Pensamento Computacional respeitando-se os conceitos iniciais apontados por (BARR; STEPHENSON, 2011) para a ciência da computação, expandindo-se os exemplos àqueles das outras áreas quando necessário ao entendimento do contexto. A seguir apresenta-se os conceitos utilizados no presente

trabalho para determinação das competências analisadas nos códigos fonte.

3.1.1 Conceito - Coleção de dados

Coleção de dados foi descrito no trabalho escolhido como referência como: “encontrar uma fonte de dados para um problema”. Expandindo-se a descrição presente no trabalho original, tomando-se outras referências discutidas nas seções anteriores, entende-se que essa habilidade corresponde à estruturas de entrada de dados em um código fonte. Dessa forma os parâmetros definidos são tokens relacionados à obtenção de dados externos.

Observando-se a gramática da linguagem C, buscou-se parametrizar as estruturas comumente utilizadas para entrada de dados em exercícios de programação, presentes na biblioteca “stdio.h” e “mysql.h”, conforme Tabela 4.

Tabela 4: Estruturas de entrada de dados

Fução	Ação
scanf	Ler informação formatada da entrada padrão
fscanf	Ler informação formatada de um fluxo de dados
fopen	Abrir arquivo
gets	Obter string da entrada padrão
fgets	Obter uma string de um fluxo de dados
getc	Obter caractere de um fluxo de dados
fgetc	Obter caractere de um fluxo de dados
getchar	Obter caractere da entrada padrão
fread	Ler bloco de dados de um fluxo
mysql_query	Solicitação à um banco de dados sql.

Fonte: elaborado pelo autor

Considerando-se o escopo estabelecido para o presente trabalho as estruturas foram definidas conforme a gramática formal da linguagem C. O conceito central utilizado é o de estruturas utilizadas para entrada de dados, podendo ser adaptado para outras linguagens de programação.

A sintaxe da estrutura identificada no presente trabalho é:

<função de entrada> ::=

<função> '(' <expressão> ')'

3.1.2 Conceito - Análise de dados

Para as ciências da computação, Barr e Stephenson (2011) descrevem *Análise de dados* como: “escrever um programa para solucionar cálculos estatísticos básicos”. Nas outras áreas do conhecimento, também presentes no artigo citado, destaca-se a descrição dessa competência para a matemática: “contar as ocorrências de lançamentos de dados ou moedas e analisar os

resultados”. Entendeu-se que essa competência tem forte relação com a análise de como os dados se relacionam e como eles podem ser combinados de forma a elaborar uma expressão com conotação estatística.

Como exemplo, pode-se analisar um exercício comum em disciplinas de programação como o cálculo de um lucro. Normalmente, é oferecido ao educando algumas informações básicas como preço de custo, margem de lucro e número de itens vendidos. Espera-se que o educando tenha a capacidade de analisar os dados oferecidos estabelecendo as relações necessárias para determinar o objetivo final: calcular o lucro total. Existem diferentes combinações ou passos que o educando pode tomar para solucionar esse problema, mas as relações sempre deverão fazer uso de operadores aritméticos para o cálculo correto do objetivo proposto. O uso do operador de atribuição também pode representar uma síntese da análise que o educando fez sobre a expressão aritmética, pois quando ele realiza uma atribuição ele consolida seu entendimento de parte ou todo o objetivo do problema, é como se o aprendiz pudesse afirmar “Essa expressão é o lucro”.

Nesse sentido, determinou-se que os operadores de atribuição utilizados para definir ou sobrescrever o valor de uma variável, e os operadores aritméticos utilizados para realizar operações matemáticas são os elementos de interesse dessa competência. Dessa forma definiu-se que os parâmetros para observação da análise de dados são os apresentados na Tabela 5.

Tabela 5: Tabela operadores aritméticos

Operador	Ação
=	Atribuição
+	Soma
-	Subtração
/	Divisão
*	Multiplicação
%	Módulo
++	Incremento unitário
--	Decremento unitário
+=	Atribuição soma
-=	Atribuição subtração
/=	Atribuição divisão
*=	Atribuição multiplicação

Fonte: elaborado pelo autor

A sintaxe das operações aritméticas identificadas no presente trabalho é:

<operação aritmética> ::=

<identificador> <operador> <identificador>

ou <identificador> <operador> <expressão>

ou <expressão> <operador> <expressão>

ou <expressão> <operador> <identificador>

3.1.3 Conceito - Representação de dados

Conforme a seção 2.4, o conceito de *Representação de dados* refere-se ao uso de estruturas que representem dados no código fonte. Nas ciências da computação o estudo de estruturas de dados faz uso tanto de métodos de armazenamento, como variáveis de tipos de dados primitivos e *arrays*, quanto de métodos de organização de dados como pilhas e árvores. Existem diferentes implementações possíveis para os métodos de organização de dados, portanto na metodologia presente definiu-se observar apenas as estruturas pré-definidas para a linguagem C, ou seja, aquelas que podem ser diretamente observadas no código fonte, sem a interpretação abstrata da semântica do algoritmo como um todo.

1. Variável primitiva base - são as declarações de variáveis em sua forma básica. Os tipos primitivos parametrizados para esse trabalho são: *int*, *float*, *double*, *char* e *void*, assim como os modificadores *long*, *short*, *signed* e *unsigned*. A declaração de variáveis base é definida por:

<declaração de variável> ::=

<tipo primitivo> <identificador>

ou <modificador> <tipo primitivo> <identificador>

2. Arrays - são variáveis multidimensionais definidas por:

<declaração de um array> ::=

<Tipo primitivo> <identificador> '[' ']'

ou <Tipo primitivo> <identificador> '[' <constante> ']'

ou <Tipo primitivo> <identificador> '[' <identificador> ']'

ou <declaração de um array> '[' ']'

ou <declaração de um array> '[' <constante> ']'

ou <declaração de um array> '[' <identificador> ']'

A tupla dos caracteres '[' e ']' definem o número de dimensões do array.

3. Ponteiros - são variáveis que são capazes de armazenar um endereço de memória de outras variáveis, nesse caso diz-se que ele aponta para a variável cujo endereço ele armazena. A sintaxe de um ponteiro é definida por:

<declaração de um ponteiro> ::=

<tipo primitivo> '*' <identificador>

ou '*' <declaração de um ponteiro>

4. Structs - são tipos de dados que agrupam diversas variáveis, permitindo que armazene-se dados de uma mesma entidade sob um identificador. Nesse trabalho definiu-se a estrutura de uma struct como:

```
<declaração de uma struct> ::=
'struct' '{' <lista de declarações> '}'
ou 'struct' <identificador> '{' <lista de declarações> '}'
ou 'struct' <identificador>
```

3.1.4 Conceito - Decomposição do problema

A *decomposição do problema* é uma das competências mais simples de observar-se diretamente em um código fonte, Barr e Stephenson (2011) descreveu a decomposição de um problema como “definir objetos, métodos e funções”. Portanto, entende-se como estruturas que visam decompor o problema geral em problemas menores. A partir desse entendimento, pode-se identificar duas formas de decomposição, a primeira é o ato de dividir o problema como uma função, um objeto ou método, ou seja, a elaboração de uma estrutura que contém um trecho de código que pode ser invocado inúmeras vezes posteriormente. A segunda é a invocação de um trecho de código previamente organizado para esse fim, e aqui o uso de funções e procedimento declarados no próprio código pode somar-se àqueles definidos pela própria linguagem ou arquivos externos. Dessa forma identificou-se os elementos relacionados a essa competência como declaração ou uso de funções, procedimentos, bibliotecas e cabeçalhos externos. Definidos por:

1. Funções e procedimentos - são estruturas que englobam um conjunto de instruções executadas quando a função é chamada, identificamos sua sintaxe como:

```
<função ou procedimento> ::=
<tipo primitivo> <identificador> '(' <expressão> ')' ' ' <declaração> ' '
ou <modificador> <função ou procedimento>
```

2. Bibliotecas e cabeçalhos - são recursos externos ao código fonte que reúnem um conjunto de funções pré-escritas. As bibliotecas podem ser incluídas no preâmbulo do código fonte e sua sintaxe é definida por:

```
<declaração biblioteca> ::=
'#include' '<' <expressão> '>'
```

3.1.5 Conceito - Abstração

Abstração foi descrito no trabalho de Barr e Stephenson (2011) como: “... encapsular um conjunto de comandos que se repetem com um mesmo fim, usar condicionais, laços, recursividade, etc.”. Expandindo-se a descrição presente no trabalho original, tomando-se outras referências

discutidas nas seções anteriores, entende-se como estruturas que representem relações centrais em um trecho de código fonte. Tomou-se como referência direta a ocorrência de operadores lógicos e relacionais utilizados em condições, comandos de repetição e estruturas de decisão. As estruturas identificadas para o conceito abstração no presente trabalho são definidas da seguinte forma:

1. Chamadas de função - a sintaxe do uso de funções foi identificado como:

<chamada de função> ::=

<identificador> '(' <expressão> ')'

2. Uso de operadores lógicos e relacionais - os operadores lógicos e relacionais são definidos conforme a Tabela 6.

Tabela 6: Operadores lógicos e relacionais

Operadores Lógicos		Operadores Relacionais	
Símbolo	Ação	Símbolo	Ação
&&	E	!=	Diferente
AND	E	==	Igualdade
!	Não	>	Maior
NOT	Não	<	Menor
	OU	>=	Maior igual
OR	OU	<=	Menor igual

Fonte: elaborado pelo autor

Operadores lógicos são utilizados quando é necessário verificar duas ou mais condições em uma mesma expressão, ou verificar a ausência de uma condição (NOT). Identificou-se a sintaxe como:

<teste logico> ::=

<operador logico> <identificador>

ou <operador logico> <expressão>

ou <identificador> <teste lógico>

ou <expressão> <teste lógico>

Os operadores relacionais realizam um teste comparativo entre duas entidades, retornando verdadeiro (1) ou falso (0). Para o teste relacional identificou-se a sintaxe como:

<teste relacional> ::=

<identificador> <operador relacional> <identificador>

ou <identificador> <operador relacional> <expressão>

ou <expressão> <operador relacional> <expressão>

ou <expressão> <operador relacional> <identificador>

3. Laços de repetição - laços de repetição são estruturas que permitem a execução de uma sequência de instruções repetidas vezes enquanto uma determinada condição é verdadeira. No presente trabalho identifica-se as seguintes estruturas de repetição:

- Comando ‘faça enquanto’ (*do while*) - essa estrutura garante a execução da sequência de instruções ao menos uma vez, até a condição de para expressa após o *while*, identifica-se sua sintaxe como:

<faça enquanto> ::=

‘do’ ‘{’ <declaração> ‘}’ ‘while’ ‘(’ <expressão> ‘)’

- Comando ‘enquanto’ (*while*) - essa estrutura permite o início e continuidade da execução da sequência de instruções somente quando a condição da expressão após o *while* é verdadeira, identifica-se sua sintaxe como:

<enquanto> ::=

‘while’ ‘(’ <expressão> ‘)’ ‘{’ <declaração> ‘}’

- comando ‘para’ (*for*) - essa estrutura é semelhante ao comando “enquanto”, com o diferencial que sua expressão utilizada como condição de continuidade é bem definida. No presente trabalho identifica-se como:

<para> ::=

‘for’ ‘(’ <expressão> ‘)’ ‘{’ <declaração> ‘}’

ou ‘for’ ‘(’ <expressão> ‘)’ <declaração>

4. Comando ‘Se’ (*if*) - O comando se é uma estrutura de decisão que executa uma sequência de instruções caso a condição de sua expressão seja verdadeira, oferecendo um caminho de execução alternativo (*else*) caso a condição seja falsa. Identifica-se no presente trabalho o comando se como:

<se então> ::=

‘if’ ‘(’ <expressão> ‘)’ <declaração>

ou ‘if’ ‘(’ <expressão> ‘)’ ‘{’ <declaração> ‘}’

ou ‘if’ ‘(’ <expressão> ‘)’ <declaração> ‘else’ <declaração>

ou ‘if’ ‘(’ <expressão> ‘)’ <declaração> ‘else’ ‘{’ <declaração> ‘}’

ou ‘if’ ‘(’ <expressão> ‘)’ ‘{’ <declaração> ‘}’ ‘else’ <declaração>

5. Comando ‘Caso’ (*switch*) - O comando “caso” é uma estrutura de decisão que executa uma sequência de instruções de acordo com o caso selecionado. No presente trabalho identificou-se a estrutura como:

`<caso> ::=`
`'switch' '(' <expressão> ')' '{'`
 ou `'switch' '(' <expressão> ')' '{' 'case' <expressão> ':' '{ <declaração> '}' '}'`
 ou `'switch' '(' <expressão> ')' '{' 'case' <expressão> ':' <declaração> '}'`
 ou `<caso> 'case' <expressão> ':' <declaração> '}'`
 ou `<caso> 'case' <expressão> ':' '{ <declaração> '}' '}'`
 ou `<caso> 'default' ':' <declaração> '}'`
 ou `<caso> 'default' ':' '{ <declaração> '}' '}'`

3.1.6 Conceito - Simulação

Barr e Stephenson (2011) definiram a competência *Simulação* como: “animação de algoritmo, busca de parâmetros”. Observando-se os conceitos apresentados para as outras áreas de conhecimento também relacionadas por Barr e Stephenson (2011), como “criar um gráfico de uma função” para a matemática, ou “simular o movimento do sistema solar” para as ciências, assim como ao analisar os trabalhos relacionados, entende-se que a *Simulação* está ligada à formas de concretizar o algoritmo em algo externo, como uma saída de dados. Portanto, entende-se que esta competência está relacionada à tokens que sejam utilizados para depuração, testagem e animação.

1. Comentários - comentários são trechos de código que não são executados pelo compilador. É comum o uso de comentários para documentação do código fonte, bem como para ocultar trechos que foram usados apenas para depuração, como a impressão do valor de uma variável. No presente trabalho, a presença de um comentário foi identificada como uma ocorrência de controle e depuração, sua sintaxe é definida como:

`<comentário> ::=`
`'/' <declaração>`
 ou `'/*' <declaração> '*/'`

2. Saídas - as saídas também são interpretadas no presente trabalho como estruturas de simulação e apresentação do resultado do algoritmo. Identifica-se a sintaxe dessas estruturas conforme as funções de saída da biblioteca “stdio.h”, apresentada na Tabela 7.

Tabela 7: Funções de saída stdio.h

Fução	Ação
fprintf	Escrever dados formatados para um fluxo
printf	Escrever dados formatados para saída padrão
sprintf	Escrever dados formatados como texto
fputc	Escrever caracter para fluxo de saída
fputs	Escrever texto para fluxo de saída
putc	Escrever caracter para fluxo de saída
putchar	Escrever caracter para a saída padrão
puts	Escrever texto para a saída padrão
fwrite	Escrever dados no fluxo

Fonte: elaborado pelo autor

3.1.7 Conceitos não relacionados nesse trabalho

Barr e Stephenson (2011) descrevem que o conceito de *Algoritmos e procedimentos* como: “estudar algoritmos clássicos, implementar um algoritmo para uma área específica”. Neste trabalho, não objetiva-se determinar se o código fonte analisado possui trechos de algoritmos clássicos, e entendendo-se que o ato de produzir um código fonte como expressão de uma solução de um problema, esse conceito não será diretamente avaliado. No presente trabalho, este conceito foi utilizado para armazenar o resultado do julgamento da correção da solução como certo ou errado.

O conceito de *Automação* não continha exemplos diretos à ciências da computação no trabalho de referência. Entende-se que elaborar um código fonte para como expressão de uma solução a um problema é uma forma de automatizar um processo ou solução, é a elaboração de um algoritmo. Portanto, esse conceito também não será utilizado no presente trabalho.

Para Barr e Stephenson (2011) *Paralelização* foi descrita como um competência voltada ao processamento paralelo. Dado a aspectos particulares da linguagem de programação “C”, a qual pretende-se utilizar para análise, nenhum token foi parametrizado para esse conceito, e não será analisado no presente trabalho.

Dessa forma serão estabelecidos parâmetros que identifiquem os tokens específicos para cada uma das habilidades conforme a Tabela 8.

Tabela 8: Habilidades e tokens extraídos

Habilidade/Conceito	Tokens
Coleção de dados	<i>Scanf, DB calls, File Calls</i>
Análise de dados	Operadores aritméticos
Representação de dados	<i>Identifiers, Arrays, Pointers, Structs</i>
Decomposição do problema	Definição de funções, Chamadas de bibliotecas, Definição de Macro
Abstração	Chamadas de função, Operadores lógicos, Laços de repetição, Chamadas recursivas, Seleções
Simulação	Comentários, printf, plots

Fonte: elaborado pelo autor

3.2 Matrizes de representação

O presente estudo busca a análise de códigos fonte a fim de determinar quais competências do Pensamento Computacional foram utilizadas na elaboração de códigos fonte e de que forma. Já apresentou-se em seções anteriores que a natureza da elaboração desses códigos assemelha-se à elaboração de soluções às questões dissertativas, ou seja, os sujeitos que elaboram as soluções podem fazê-lo de diferentes formas, e ainda assim obter um resultado satisfatório. A partir dessa diversidade de formas, é que entendeu-se existir a oportunidade de inferir que os autores de códigos fontes podem utilizar-se de diferentes competências, e em diferentes níveis.

Dessa forma, em uma análise intra-documento de códigos fonte pode-se determinar e relacionar estruturas com as competências do Pensamento Computacional definidas nas seções anteriores, mas para determinar-se aspectos qualitativos dessas competências utilizou-se uma análise inter-documentos de um conjunto, ou seja, uma comparação com outros códigos fontes pertencentes a um mesmo conjunto. Para essa comparação, determinou-se que um conjunto de códigos fonte elaborados para a solução de um mesmo problema compõem um conjunto de dados.

Objetiva-se com esse estudo a análise dos conjuntos de soluções de problemas de programação expressas em códigos fonte, dessa forma cada código fonte processado individualmente deve posteriormente ser organizado em conjunto com os demais pertencentes ao mesmo conjunto. Para cada problema, propõe-se a construção de três matrizes: (i) uma *matriz documento por token*, (ii) uma *matriz documento por frequência*, e (iii) uma *matriz balanceada*.

A *matriz documento por token* é formada pelo conjunto de soluções onde cada linha corresponde à um código fonte, e cada coluna corresponde aos tokens extraídos em sua forma sequencial. Essa matriz corresponde aos dados brutos, processadas de forma a transformar o código original em uma representação numérica.

A *matriz documento por frequência* é formada pelo conjunto de soluções onde cada linha

corresponde a um código fonte, e cada coluna corresponde à frequência dos tokens relacionados a cada uma das competências, conforme a Tabela 8. Para a construção da frequência das competências, conta-se o número de ocorrências para cada token e posteriormente soma-se aqueles relacionados com a mesma competência, ou seja, a frequência de uma competência é a soma de todas as ocorrências dos tokens relacionados a ela.

A *matriz de frequência* é então balanceada aplicando-se o método de TF-IDF (abreviação dos inglês *term frequency-inverse document frequency*), conforme as Equações 3.1, 3.2, 3.3. A variável $tf_{t,d}$ representa a *term-frequency* (frequência do termo) onde a variável f representa a frequência, ou seja, a contagem do número de ocorrências de um termo relacionado à uma competência, t corresponde ao termo ou token, d ao documento ou código fonte e a expressão $\max(f_{t,d} : t \in d)$ representa o número total de termos presentes no documento. A variável $idf_{t,D}$ representa a *inverse-document-frequency* (frequência inversa do documento), N corresponde ao número total de códigos fonte em um conjunto D , e a expressão $\{d \in D : t \in d\}$ corresponde ao número de documentos de um conjunto D nos quais um termo t está presente. A informação balanceada é então armazenada em outra matriz, uma *matriz balanceada documento por TF-IDF*.

$$tf_{t,d} = \frac{f_{t,d}}{\max(f_{t,d} : t \in d)} \quad (3.1)$$

$$idf_{t,D} = \log \frac{N}{\{d \in D : t \in d\}} \quad (3.2)$$

$$tfidf_{t,d,D} = tf_{t,d} \times idf_{t,D} \quad (3.3)$$

Observou-se em experimentos preliminares que todas as competências estavam presentes em todos os códigos fonte válidos analisados, ainda que em diferentes níveis. Aplicando-se os agrupamentos propostos de tokens por competências já esperava-se que em nenhuma competência a frequência seria igual a zero. No caso de uma habilidade estar presente em todos os códigos fonte, como é esperado mesmo que em diferentes proporções, observa-se que a Equação 3.2 obteria um resultado zero, inviabilizando o método. Com o intuito de ajustar o uso do IDF considerou-se apenas a frequência de documentos nos quais a contagem de tokens relacionados à uma competência é maior que a soma da média de ocorrência de tokens relacionados a essa competência $cm_{t,d}$ com o seu desvio padrão $std_{t,D}$, dessa forma a Equação 3.2 é balanceada como Equação 3.4.

$$idf_{t,D} = \log \frac{N}{\{d \in D : t > (m_{t,D} + std_{t,D}) \in d\}} \quad (3.4)$$

O uso da TF-IDF no presente trabalho objetiva emergir a importância de uma competência para a construção de uma solução a partir de um código fonte. Observa-se também a importância média de uma competência para o conjunto de soluções. Contudo, entende-se que a TF-IDF não se aplica à comparação dos níveis de competências presentes em diferentes soluções, ela apenas determina a relevância de cada competência para um determinado código fonte, pois observou-se

em experimentos preliminares que a TF-IDF tem um efeito normalizador. Dessa forma, é possível que dois códigos fontes diferentes, um considerado com um alto nível em uma dada competência, e o outro considerado com um baixo nível nessa mesma competência, apresentem um valor semelhante para TF-IDF. A seguir, busca-se atender o propósito de estabelecer a comparação de níveis de competência.

3.3 Rubrica de avaliação

Nesta seção busca-se apresentar a metodologia utilizada para determinar a rubrica de avaliação de um conjunto de códigos fonte. Conforme apresentado nas seções anteriores, rubricas são um conjunto de regras que definem critérios padronizados para determinar o resultado de uma avaliação (ALBANO, 2017). No presente trabalho, a rubrica proposta utiliza-se das relevâncias de cada competência na elaboração dos códigos fonte, determinado pelo cálculo da TF-IDF, em conjunto com a determinação de uma medida para o nível da competência analisada, a qual denominou-se proporção Theta. Por fim, ao produto da relevância com a proporção Theta, que entende-se como uma medida que represente o nível das competências empregadas na elaboração do código fonte, denominou-se *Habilidade Theta*.

3.3.1 Proporção Theta

O número de tokens presentes em um código fonte já foi utilizado em outros trabalhos (MOREIRA; FAVERO, 2009)(CAPITÁN; VOGEL-HEUSER, 2017) para determinar a qualidade de um software. O número de elementos presentes em um software, entre outras características, pode ser utilizado como uma indicação de eficiência, de forma que se dois softwares executam a mesma tarefa, aquele com a menor complexidade é provavelmente o mais eficiente (CAPITÁN; VOGEL-HEUSER, 2017). Da mesma forma, ao considerarmos uma habilidade utilizada para elaborar uma solução, a soma direta de um elemento pode não representar uma proficiência elevada, isto é, ter uma ocorrência alta de elementos em um código fonte pode inferir uma baixa habilidade relacionada.

Para o presente trabalho utilizou-se como balizador para inferência do nível das competências empregadas na elaboração de códigos fonte um sistema de comparação entre códigos que compõem um conjunto de respostas para um mesmo exercício.

Dada a característica do presente trabalho, no qual a informação é extraída a partir da frequência de tokens e utilizada para estimar competências, dado um determinado conjunto de soluções de um mesmo problema, assumiu-se que ou uma competência tem um nível proporcional θI em função da razão entre a menor frequência de tokens observada no conjunto de dados e a frequência de tokens da solução, indicada pela Equação 3.5, ou um nível proporcional θD em função da razão entre a frequência de tokens observada na solução e a maior frequência de tokens observada no conjunto de soluções, indicado pela Equação 3.6.

$$\theta I_{t,d} = \frac{\min f_{t,d}}{f_t} \quad (3.5)$$

$$\theta D_{t,d} = \frac{f_t}{\max f_{t,d}} \quad (3.6)$$

onde $\theta I_{t,d}$ é a Proporção Theta para as competências que usam a proporção inversa, $\theta D_{t,d}$ é a Proporção Theta para as competências que usam a proporção direta, f_t é a frequência de tokens no código fonte para uma competência, $\min f_{t,d}$ é a menor frequência de tokens observada no conjunto de soluções e $\max f_{t,d}$ é a maior frequência de tokens observada no conjunto de soluções.

Supôs-se, portanto, que para uma competência cuja maior ocorrência de estruturas em um código fonte é desejável, seu nível é diretamente proporcional à frequência de tokens relacionados a esta mesma competência, e nesse caso utiliza-se a proporção θD . Da mesma forma, para uma competência cuja maior ocorrência de estruturas em um código fonte é indesejável, seu nível é inversamente proporcional à frequência de tokens relacionados a esta mesma competência, e nesse caso utiliza-se a proporção θI .

A suposição utilizada para determinar a proporcionalidade parte do princípio que, para as competências inversamente proporcionais, se o código fonte A consegue apresentar uma solução com um menor número de tokens relacionados à competência X em comparação ao código fonte B, A provavelmente faz uso da competência analisada de forma mais adequada. Em contrapartida, para as competências diretamente proporcionais, se o código fonte A possui uma frequência mais elevada de tokens relacionados à competência X em comparação ao código fonte B, A provavelmente faz uso da competência analisada de forma mais adequada.

Dessa forma, ao analisar-se o conjunto de soluções para um mesmo problema, toma-se como referência sempre aquele que fez um uso mais desejável das estruturas relacionadas às competências, seja a maior ou menor frequência. Assim, determinou-se a proporcionalidade da seguinte forma:

1. *Coleção de dados*: considerou-se como inversamente proporcional.

O controle sobre a entrada de dados está relacionado à capacidade do programador em sintetizar e organizar as informações externas utilizadas. Dessa forma, entende-se que quanto menor a frequência de tokens relacionados à *Coleção de Dados* maior a capacidade de síntese, ou melhor o controle sobre as estruturas relacionadas à essa competência.

2. *Análise de dados*: considerou-se como inversamente proporcional.

O uso de um maior número de operadores resulta também em um maior custo computacional para execução do código, dessa forma é desejável que o programador otimize seu código para a simplificação das operações realizadas.

3. *Representação de dados*: considerou-se como inversamente proporcional.

O uso das estruturas relacionadas com a *Representação de dados* está relacionado com a quantidade de memória requerida pelo código fonte. Considere-se que um código que faz uso de um menor uso de memória alocada é melhor otimizado, portanto é desejável uma menor frequência de estruturas para competência *Representação de dados*.

4. *Decomposição do problema*: diretamente proporcional.

O ato de decompor um problema, separando-o em partes menores que podem ser reutilizadas é desejável em programação. Dessa forma, o uso de estruturas relacionadas com a *Decomposição do problema* demonstram um maior domínio dessa competência.

5. *Abstração*: inversamente proporcional.

Os elementos relacionados com a *Abstração* também podem ter um impacto significativo no custo computacional de execução do código fonte. Considera-se que quando o programador consegue solucionar um problema com o menor número possível de estruturas dessa competência ele demonstra o melhor nível de domínio sobre ela.

6. *Simulação*: diretamente proporcional.

A capacidade de depurar e documentar um código é desejável para o aprendizado de programação. Dessa forma, entende-se que a maior ocorrência de tokens relacionados com a *Simulação* indica um maior nível de domínio sobre essa competência.

O uso da proporção θ no presente trabalho objetiva determinar o nível de uma competência em uma solução escrita em um código fonte em relação às demais presentes no conjunto de dados. O nível determinado para cada competência representa uma comparação entre os códigos presentes no conjunto de dados, de forma que a qualidade das soluções presentes nesse conjunto é o que determina o máximo e mínimo calculado. Se considerarmos a soma dos θ pode-se determinar que uma solução é a melhor no universo do conjunto de dados que ela está inserida, no entanto não se pode determinar que ela é a melhor solução possível para o problema.

3.3.2 Habilidade Theta

A TF-IDF objetiva emergir a importância de uma habilidade em uma submissão, o valor θ por sua vez visa estabelecer uma comparação de nível de competências de uma solução em relação às outras respostas referentes a um mesmo problema. Utilizou-se a TF-IDF determinada para cada competência para balancear os valores calculados para a proporção Theta.

Considera-se que o produto da Proporção θ e a TF-IDF, representado por $H\theta$ na Equação 3.7, representa tanto a importância de uma competência para uma solução, como também permite a comparação de nível das competências empregadas na elaboração de diferentes soluções, denominou-se esse produto como Habilidade θ .

$$H\theta_{t,d} = tfidf_{t,d,D} \times \theta_{t,d} \quad (3.7)$$

Considera-se como $\theta_{t,d}$ na Equação 3.7 ou o resultado da Equação 3.5 ou o resultado da Equação 3.6. O valor determinado para a Habilidade Theta para cada uma das competências representa o nível estimado de cada uma delas, considerando-se também a relevância de cada competência para o problema como um todo. Para a validação do modelo proposto, a seguir apresenta-se um experimento no qual aplica-se a metodologia proposta em um conjunto de dados.

4 EXPERIMENTO DE VALIDAÇÃO

Neste capítulo serão apresentadas as ações voltadas à aplicação da metodologia proposta em um determinado conjunto de dados. Fazem parte dessas ações a definição e coleta dos dados utilizados no experimento, a avaliação de parte dos dados por especialistas, a aplicação do modelo e a comparação dos resultados.

4.1 Definição e coleta dos dados

Para o desenvolvimento do presente experimento escolheu-se utilizar uma base de dados composta por códigos fonte submetidos para solução de problemas computacionais em uma plataforma *Online Judge* denominada Codeforces. Visando a padronização das estruturas possíveis de serem observadas na base definiu-se que seriam utilizados apenas códigos fontes pertencentes à mesma linguagem, escolheu-se a linguagem “GNU C” para realização desse experimento, e dessa forma todos os códigos fonte que não correspondiam a essa linguagem foram ignorados. Ainda, apenas os problemas que totalizavam pelo menos 300 submissões foram selecionados. Dessa forma, foram processados 3144 códigos fonte distribuídos em seis problemas, a Tabela 9 apresenta a distribuição de submissões entre os problemas analisados, bem como o número de soluções que foram julgadas como corretas para cada problema e aquelas julgadas incorretas.

Tabela 9: Distribuição das submissões por problema

Problema	Total de submissões	Corretas	Incorretas
P1	406	85	321
P2	900	176	724
P3	675	172	503
P4	362	101	261
P5	423	105	318
P6	378	38	340

Fonte: elaborado pelo autor

Para a coleta dos dados, utilizou-se a API¹ disponibilizada pelo próprio sítio Codeforces para gerar a lista de submissões relacionadas a cada problema. As informações coletadas de cada submissão foram: *submission Id*, referente à identificação da submissão, a qual foi omitida

¹ *Application Programming Interface* - padrão de requisição e resposta utilizado para troca de informações entre aplicações

no pré-processamento a fim de impossibilitar a identificação do usuário; *problem Id*, como a identificação do problema; *language*, como a linguagem formal de programação na qual a solução foi escrita. A API disponibilizada não possibilitava a obtenção do código fonte das submissões diretamente, portanto elaborou-se uma aplicação *scraper*² para a coleta dessa informação, também diretamente no sítio Codeforces. A informação do texto do código fonte foi incorporada à submissão correspondente como *source*.

4.2 Avaliação humana

Uma vez definidos e coletado o conjunto de dados, no presente experimento buscou-se submeter um subconjunto do lote inicial à avaliação humana. Além do autor do presente trabalho, buscou-se agregar a avaliação de outros especialistas. Entende-se que avaliar o conjunto completo de dados demonstra-se-ia uma tarefa morosa, que limitaria a quantidade de especialistas disponíveis à participação da avaliação. Dessa forma, limitou-se o escopo desse subconjunto dos dados apenas aqueles que obtiveram um resultado considerado como correto para a solução dos problemas. Assim, para realização dessa avaliação, encontrou-se disponibilidade de ao menos um segundo especialista, professor de programação e Mestre em Ciências da Computação, além do autor do presente trabalho, Licenciado em Computação e também professor de programação. Destaca-se que durante a avaliação realizada não houve troca de pareceres entre os avaliadores, ou seja, a avaliação foi realizada individualmente.

4.2.1 Critérios de avaliação

Para cada um dos avaliadores forneceu-se os conjuntos de códigos fonte julgados corretos, agrupados pelos problemas aos quais eles pertencem. Elaborou-se um instrumento com instruções de avaliação e preenchimento, conforme Tabela 10.

²Um aplicativo que acessa um sítio na internet e guarda uma informação presente no site

Tabela 10: Instruções do instrumento de avaliação

Essa planilha destina-se às anotações relevante ao conjunto de soluções referentes apenas ao problema 1. Considere que todas as soluções apresentadas estão corretas.				
Para cada resposta analisada deseja-se saber a percepção do avaliador quanto a Relevância de uma habilidade para a construção de uma resposta, assim como a percepção da qualidade da habilidade utilizada na elaboração dessa mesma resposta. Toma-se como referência os conceitos de relevância e qualidade conforme segue:				
Relevância:				
Entende-se como a importância dos elementos relativos a competência analisada presentes no código fonte, frente aos demais elementos presentes nesse mesmo código fonte, bem como se eles são determinantes para a solução.				
Qualidade:				
Entende-se como a capacidade de utilizar e manipular elementos relativos a competência analisada de forma e quantidade adequados à otimização da solução. Considera-se o conjunto de soluções disponíveis para o problema, inclusive os exemplos ofertados.				
Atribua um valor inteiro entre 1 (inclusive) e 5 (inclusive) para a sua avaliação de cada uma das competências para a elaboração do código fonte, levando em consideração a relevância e a qualidade, sendo: o menor valor (1) para a mais baixa relevância/qualidade e o maior valor (5) para a mais alta relevância/qualidade.				
Relação de competências e elementos do código fonte.				
Competências:	Elementos			
Coleção de dados	Entrada de dados			
Análise de dados	Operações aritméticas	Atribuições		
Representação de dados	Identificadores	Arrays	Ponteiros	Estruturas
Decomposição do problema	Definição de funções	Bibliotecas	Macros	Chamadas de função
Abstração	Seleções	Operadores lógicos	Laços	
Simulação	Comentários	saídas	plots	

Fonte: elaborado pelo autor

O instrumento de avaliação foi elaborado em um arquivo em formato xlsx, composto por uma planilha com as instruções de preenchimento, outra planilha com dois códigos fonte exemplo e uma planilha para preenchimento das habilidades. Os códigos fonte exemplo foram selecionados pelo autor, um como um exemplo de uma solução considerada boa e outro com uma solução considerada ruim.

4.2.2 Resultados gerais da avaliação humana

Os resultados apresentados pelas avaliações dos dois avaliadores foram comparados para determinar sua similaridade. Entende-se que avaliações realizadas por dois humanos podem

apresentar resultados diferentes, pois elas são o resultado de suas próprias perspectivas quanto à adequação e relevância das diversas estruturas presentes em um código fonte. Utilizou-se a correlação Tau de Kendall na comparação das avaliações, a Tabela 11 apresenta os valores obtidos.

Tabela 11: Correlações das avaliações entre os avaliadores

Problema	Legenda	Competências				
1		data_analysis	data_representation	problem_decomposition	abstraction	simulation
	tau	0,402501429	0,428502286	0,769503596	0,454622164	0,377557308
	p	3,82E-05	3,45E-06	1,60E-14	1,05E-06	0,000299249
	h0	Rejeita	Rejeita	Rejeita	Rejeita	Rejeita
	Intensidade	Média	Média	Forte	Média	Média
2	tau	0,411078113	0,610296478	0,636396138	0,735179402	0,303717991
	p	1,65E-08	1,35E-17	1,42E-17	5,05E-27	4,45E-05
	h0	Rejeita	Rejeita	Rejeita	Rejeita	Rejeita
	Intensidade	Média	Forte	Forte	Forte	Média
3	tau	0,684017087	0,855144935	0,775523477	0,482086997	0,340835078
	p	5,10E-23	9,56E-38	7,64E-28	4,56E-12	4,21E-06
	h0	Rejeita	Rejeita	Rejeita	Rejeita	Rejeita
	Intensidade	Forte	Forte	Forte	Média	Média
4	tau	0,472698345	0,70060269	0,408492869	0,490850076	0,546494325
	p	1,23E-07	3,87E-15	2,90E-05	7,09E-08	1,93E-08
	h0	Rejeita	Rejeita	Rejeita	Rejeita	Rejeita
	Intensidade	Média	Forte	Média	Média	Forte
5	tau	0,645689248	0,628678686	0,477050663	0,665750597	0,482950785
	p	3,48E-14	2,42E-13	4,69E-07	1,45E-14	3,49E-07
	ho	Rejeita	Rejeita	Rejeita	Rejeita	Rejeita
	Intensidade	Forte	Forte	Média	Forte	Média
6	tau	0,640944411	0,66517455	0,618690657	0,655293919	0,525505449
	p	2,46E-06	5,59E-07	1,27E-05	9,88E-07	0,000737307
	ho	Rejeita	Rejeita	Rejeita	Rejeita	Rejeita
	Intensidade	Forte	Forte	Forte	Forte	Forte

Fonte: elaborado pelo autor

Escolheu-se o teste de correlação não paramétrica de Kendall pois observou-se em estudo prévio dos dados um comportamento não normal para algumas competências, conforme a Tabela 12 que apresenta o teste de Shapiro-Wilk. Observa-se que o teste encontrou evidências para rejeitar a hipótese nula de que os dados apresentam um comportamento normal. Não encontrando-se um padrão para referência, estabeleceu-se como valores limites para as correlações como: (correlação $< 0,1$) = nula; ($0,1 < \text{correlação} < 0,3$) = média; (correlação $> 0,5$) = forte. Observa-se que para todas as correlações analisadas rejeitou-se a hipótese nula de que os dados são independentes.

Tabela 12: Teste de normalidade dos dados do avaliador 1

Problema	Legenda	Competências				
1		data_analysis	data_representation	problem_decomposition	abstraction	simulation
	S-W	0,353834801	0,798727348	0,887059451	0,668135218	0,869228107
	p	1,50E-17	2,02E-09	1,99E-06	1,46E-12	4,02E-07
	h0	Rejeita	Rejeita	Rejeita	Rejeita	Rejeita
2	S-W	0,353834801	0,798727348	0,887059451	0,668135218	0,869228107
	p	1,50E-17	2,02E-09	1,99E-06	1,46E-12	4,02E-07
	h0	Rejeita	Rejeita	Rejeita	Rejeita	Rejeita
3	S-W	0,353834801	0,798727348	0,887059451	0,668135218	0,869228107
	p	1,50E-17	2,02E-09	1,99E-06	1,46E-12	4,02E-07
	h0	Rejeita	Rejeita	Rejeita	Rejeita	Rejeita
4	S-W	0,353834801	0,798727348	0,887059451	0,668135218	0,869228107
	p	1,50E-17	2,02E-09	1,99E-06	1,46E-12	4,02E-07
	h0	Rejeita	Rejeita	Rejeita	Rejeita	Rejeita
5	S-W	0,353834801	0,798727348	0,887059451	0,668135218	0,869228107
	p	1,50E-17	2,02E-09	1,99E-06	1,46E-12	4,02E-07
	ho	Rejeita	Rejeita	Rejeita	Rejeita	Rejeita
6	S-W	0,353834801	0,798727348	0,887059451	0,668135218	0,869228107
	p	1,50E-17	2,02E-09	1,99E-06	1,46E-12	4,02E-07
	ho	Rejeita	Rejeita	Rejeita	Rejeita	Rejeita

Fonte: elaborado pelo autor

Uma das características que observou-se prontamente com essa base de dados, é que as estruturas relacionadas com a competência *Coleção de Dados* apresentaram uma frequência baixa de tokens. De fato, dado as características dos *Online Judges*, nos quais compara-se se um determinado conjunto de entradas geram o conjunto de saídas esperado, quase não observa-se variações no número de estruturas de entrada de dados, a entrada de dados para os problemas é fixa.

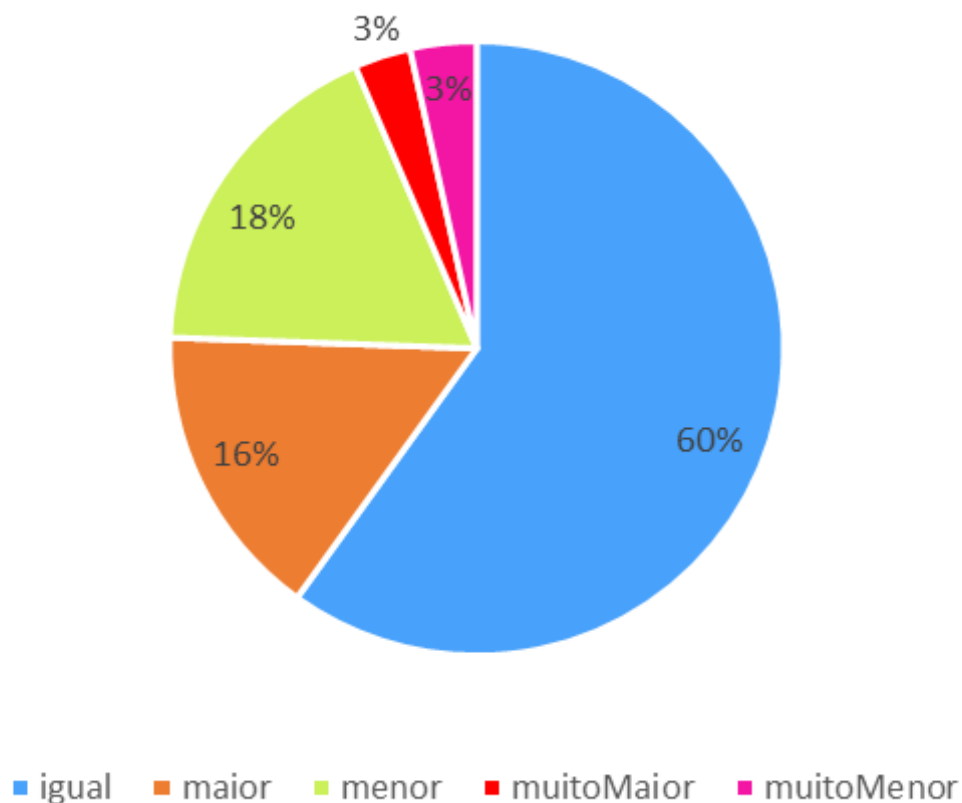
Devido a essa característica, observou-se um indício de que avaliadores humanos podem ter percepções diferentes na avaliação de uma competência; para o avaliador 1 a nota atribuída à competência *Coleção de Dados* nos códigos fontes presentes na base foi inferior em pelo menos dois pontos da nota atribuída pelo avaliador 2; visto que, salvo em casos raros, o número de tokens presentes nesses códigos fontes é fixo, entende-se que a diferença pode estar relacionada com a percepção da relevância.

De fato, não há variação significativa do uso das estruturas da competência *Coleção de Dados* para a elaboração dos códigos fonte presentes na base de dados utilizada nesse experimento, mas observa-se uma diferença significativa nas avaliações humanas. Portanto, a fim de concentrar as comparações realizadas a seguir em competências que registram variação também no uso de suas estruturas, nos experimentos a seguir serão omitidos os resultados da competência *Coleção de Dados*.

Procurou-se verificar de que forma as variações das correlações observadas ocorreram, observou-se os valores das notas atribuídas por cada avaliador e comparou-se a variação das

mesmas. A Figura 4.1 ilustra a variação das notas observadas somando-se todos os problemas e todas as competências. Para cada submissão, comparou-se a nota atribuída a cada competência, tomando como referência sempre o avaliador 1. Assim, a variação denominada no gráfico como *maior* é obtida quando o resultado da diferença entre a nota atribuída pelo avaliador 1 e a nota atribuída pelo avaliador 2 for igual a apenas um ponto. Por exemplo: nota avaliador 1 = 4, nota do avaliador 2 = 3, $4 - 3 = 1$, variação = *maior*. Quando o resultado da diferença for superior (resultado da diferença > 1) denominou-se *muito maior*. A mesma lógica foi aplicada para os valores inversos, *menor* e *muito menor*. Quando as avaliações foram idênticas denominou-se o resultado como *igual*.

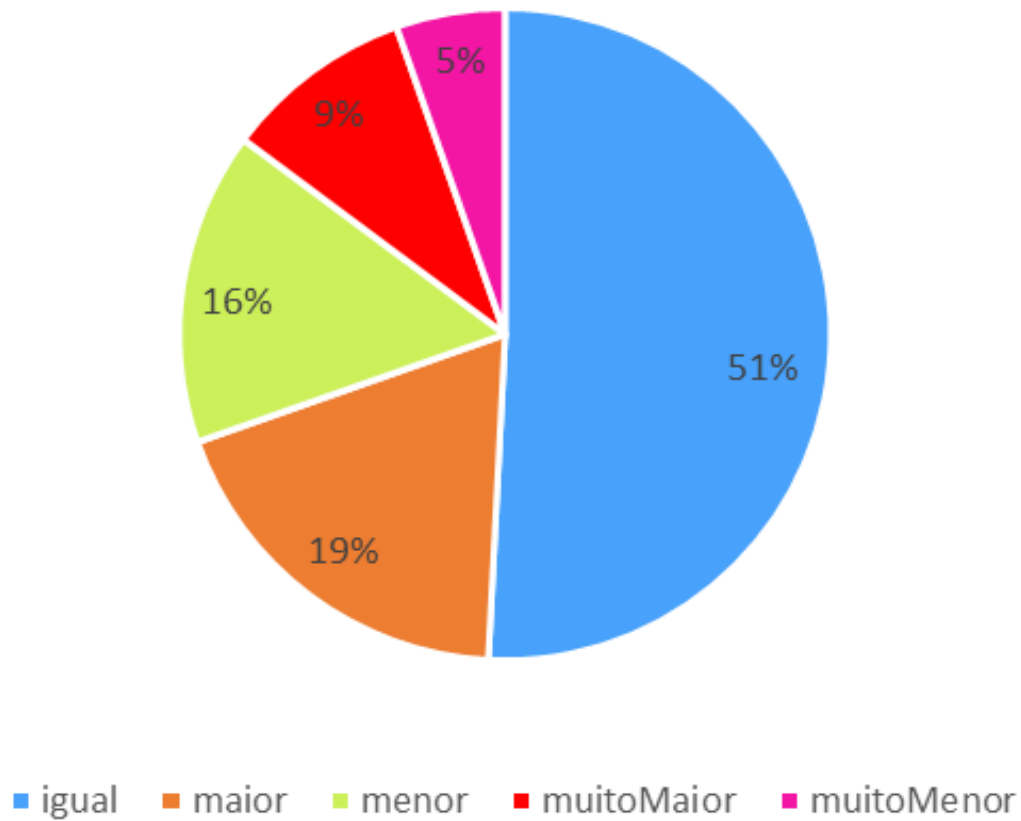
Figura 4.1: Variação das notas entre os avaliadores



Fonte: elaborado pelo autor

Os resultados obtidos através da comparação de todas as avaliações apresentam indícios de que entre dois avaliadores pode haver variação de critérios, mas as avaliações com variação maiores que um ponto (para mais ou para menos) somam apenas 6% do total, o que poderia indicar evidências de que os avaliadores podem divergir, mas raramente atribuem notas discrepantes. No entanto, quando observados os problemas separadamente essa variação não é uniformemente distribuída. A Figura 4.2 apresenta as variações das avaliações no problema um e demonstra que os avaliadores podem divergir, ainda que não significativamente.

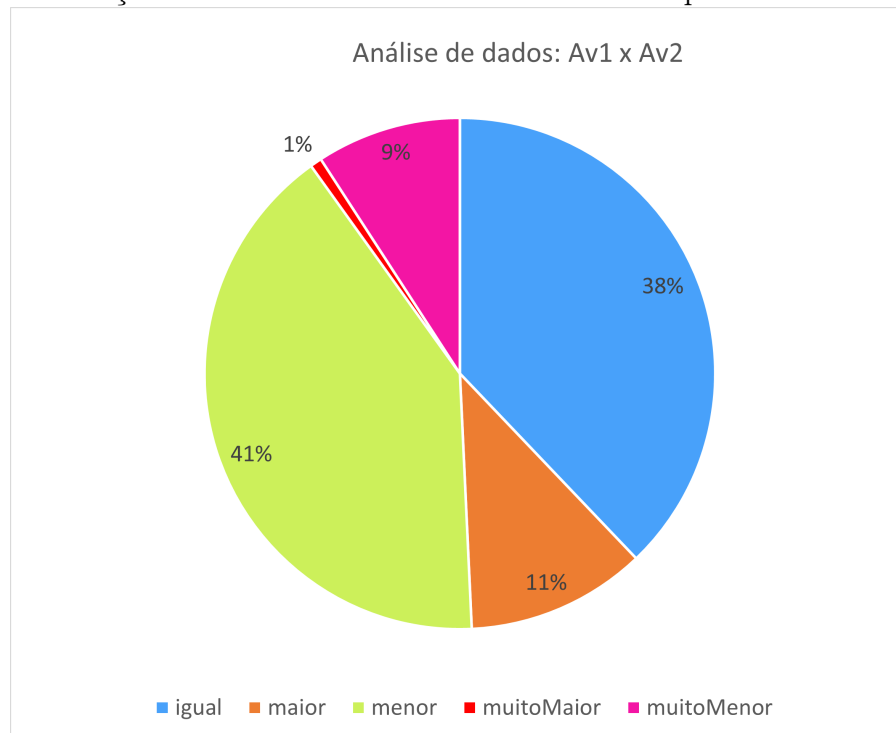
Figura 4.2: Variação das notas entre os avaliadores no problema 1



Fonte: elaborado pelo autor

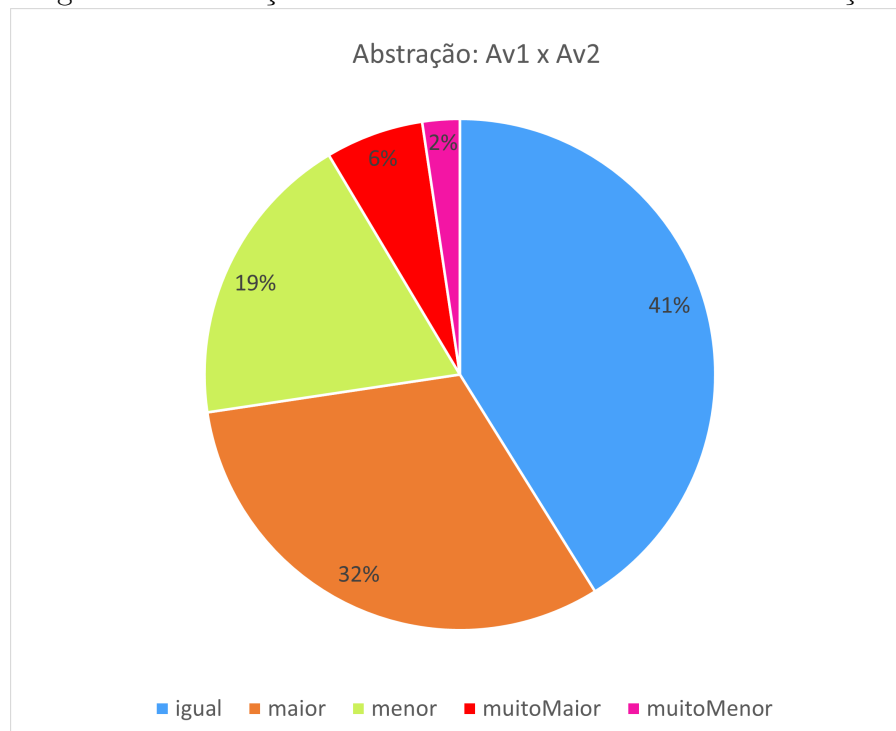
Nesse mesmo contexto, considerando-se as avaliações por competências, os indícios de divergências tornam-se ainda mais evidentes. Destaca-se as divergências nas avaliações das competências *Análise de Dados* e *Abstração*, a primeira ilustrada na Figura 4.3 e a segunda ilustrada na Figura 4.4. Destaca-se que para a competência *Análise de Dados*, pode-se considerar que o avaliador 2 tende a ser mais benevolente, ou seja atribui uma nota maior que o avaliador 1, enquanto que o avaliador 1 tende a ser mais benevolente na competência *Abstração*.

Figura 4.3: Variação das notas entre os avaliadores na competência *Análise de dados*



Fonte: elaborado pelo autor

Figura 4.4: Variação das notas entre os avaliadores - *Abstração*

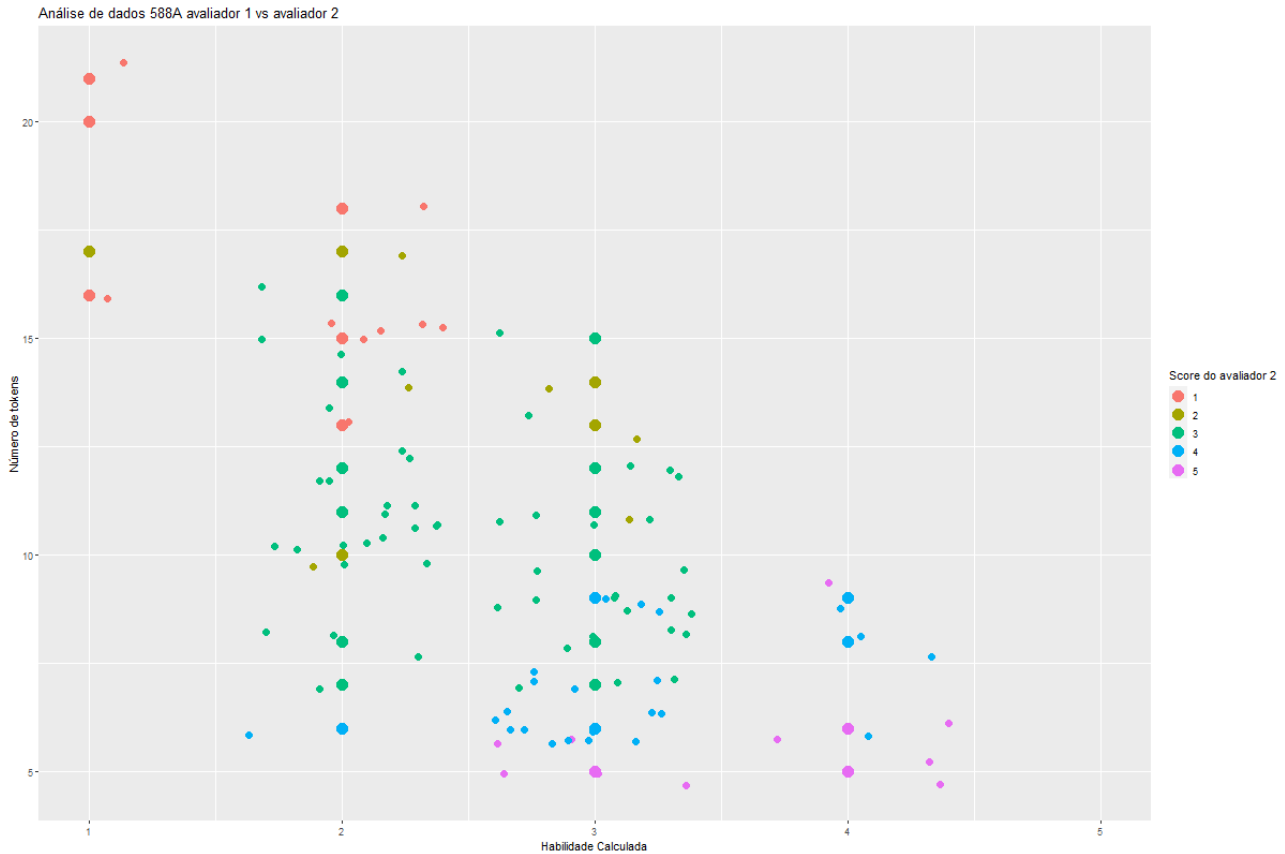


Fonte: elaborado pelo autor

Considera-se que as causas das diferentes avaliações observadas entre os avaliadores podem estar relacionadas ao próprio entendimento das competências analisadas, uma vez que são

conceitos abstratos, passíveis de interpretações. Percebe-se que a nota atribuída pelos avaliadores pode ter relação com a frequência de tokens presentes no código, a Figura 4.5 apresenta um gráfico de dispersão que ilustra a relação da frequência (eixo y) com as notas atribuídas pelos avaliadores (eixo x para o avaliador 1 e cores para o avaliador 2).

Figura 4.5: Dispersão com Jitter - *Análise de Dados* - Problema 5



Fonte: elaborado pelo autor

Os conceitos de relevância e qualidade utilizados como referência para as avaliações também são abstratos, dependentes da interpretação dos autores e de suas experiências em avaliar os códigos. Nessa experiência de avaliação apresenta-se uma característica importante observada nos conjuntos de avaliações, a referência do avaliador quanto à relevância e qualidade empregadas na construção do código fonte não é constante. Essa referência é sequencial, e está relacionada aos códigos avaliados anteriormente, ou seja, conforme o avaliador percorre a lista de códigos a serem avaliados sequencialmente, na ocasião de encontrar um código que ele julgue melhor que aquele que estava sendo utilizado como referência, esse código melhor elaborado passa a ser a referência, e todos os seguintes serão julgados por ele, sem efeito retroativo aos que já haviam sido analisados. Portanto, as avaliações humanas podem utilizar referências distintas em momentos distintos.

4.3 Avaliação do modelo

O objetivo das ações apresentadas nesta seção é observar o comportamento do modelo proposto quando aplicado à um conjunto de dados. Para tanto, foi necessária a elaboração de um analisador de código fonte, doravante denominado *parser*, para a transformação dos dados brutos em representações numéricas. Para esse esforço de desenvolvimento utilizou-se os softwares GNU Flex para a construção do analisador léxico, e GNU Bison para a construção de um analisador sintático. Tomou-se como referência de estruturas a serem analisadas aquelas mencionadas na seção 3.1. O objetivo da construção do *parser*, resultante da combinação dos dois analisadores, foi o de possibilitar a identificação de estruturas sintáticas e semânticas nos códigos fontes, gerando-se assim uma representação numérica dos códigos presentes na base.

Na ocasião do processamento do *parser*, a base de dados foi transformada em uma representação numérica, ou seja, cada código fonte foi vetorizado em uma representação numérica e posteriormente organizado em *matrizes de representação* dos conjuntos de soluções relativas a cada problema. Dessa forma, buscou-se analisar cada problema separadamente, preservando-se as características dos conjuntos que possuem o mesmo fim, a solução do mesmo problema.

Uma vez que as *matrizes de representação* dos problemas foram definidas, os dados foram processados utilizando-se do software RStudio. Para o processamento das informações, algumas funções padrão do software foram utilizadas, como `std()`, `sum()`, `mean()`, etc, enquanto que outras necessárias à realização do experimento foram elaboradas pelo autor. A seguir apresenta-se os resultados obtidos com a aplicação do modelo à base de dados.

4.3.1 Resultados do modelo

Com o conjunto de submissões transformado em representações numéricas e organizado em seis matrizes de representação, cada uma correspondente ao conjunto de soluções de um problema, o modelo proposto foi aplicado em cada matriz separadamente. Utilizou-se o software RStudio para o processamento da base e aplicação do modelo, assim foram realizadas rotinas para:

- I) importação das *matrizes de representação*;
- II) pré-processamento, etapa na qual foi omitida identificação das submissões e realizada a verificação da consistência dos dados numéricos;
- III) contabilização das frequências de cada estrutura presente no código, transformando-se assim as *matrizes de representação* iniciais em *matrizes de documento por frequência* de tokens estruturais, ou *documento por frequência*;
- IV) agrupamento das frequências de acordo com cada competência, transformando-se assim a *matriz de documento por frequência* em uma matriz de *documentos por frequência de tokens por competência*, ou *documento por competência*;

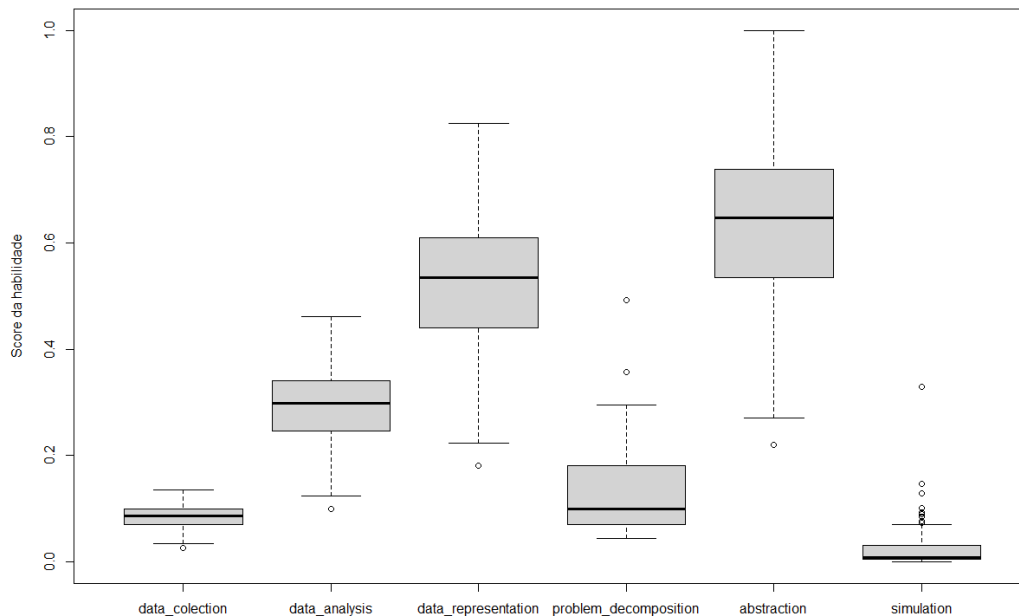
- V) cálculo das TFs a partir da matriz de documento por competência;
- VI) cálculo das IDFs a partir da matriz de documento por competência;
- VII) cálculo das Proporções Theta;
- VIII) cálculo das Habilidades Thetas.

Os valores obtidos para a Habilidade Theta foram normalizados entre os valores um e cinco utilizando-se o método de máximos e mínimos conforme equação 4.1, onde x' corresponde ao valor atualizado, x o valor original, a o menor valor desejado e b o maior valor desejado.

$$x' = a + \frac{(x - \min(x)) (b - a)}{\max(x) - \min(x)} \quad (4.1)$$

Da mesma forma que nas avaliações humanas observou-se que alguns dos dados obtidos não apresentavam um comportamento normal, conforme pode ser observado na Figura 4.6 que ilustra o gráfico Box-Plot para as Habilidade Theta do problema um. Observou-se o mesmo comportamento dos dados nos demais problemas conforme o cálculo do teste de normalidade de Shapiro-Wilk apresentado na Tabela 13.

Figura 4.6: Gráfico Box-Plot Habilidade Theta Problema 1



Fonte: elaborado pelo autor

Assim como nas avaliações humanas, as características da base de dados onde a entrada de dados é fixada, a competência *Coleção de Dados* não se mostrou relevante à análise do comportamento do modelo, pois sua relevância foi considerada desprezível à elaboração do problema, conforme também pode ser observado na Figura 4.6, esse mesmo comportamento

também foi observado nos demais problemas, portanto a competência *Coleção de Dados* será omitida das análises a seguir.

Outra observação que pode ser feita a partir da Figura 4.6 é que em geral a competência *Simulação* não é relevante para elaboração das soluções das respostas ao problema 1, no entanto, diferentemente da competência *Coleção de Dados*, a presença dos inúmeros *outliers* são indício de que para alguns códigos a competência *Simulação* foi relevante para a elaboração dos códigos fontes. A presença dos *outliers*, portanto, pode apontar indícios que indicam a existência de códigos fonte muito diferenciados da média, tanto positivamente quanto negativamente.

Tabela 13: Teste de normalidade Habilidade Theta

Problema	Legenda	Competências				
1		data_analysis	data_representation	problem_decomposition	abstraction	simulation
	S-W	0,986987734	0,986987734	0,847541117	0,986987734	0,571882662
	p	0,555184034	0,555184034	6,75E-08	0,555184034	2,32E-14
	h0	Não rejeita	Não rejeita	Rejeita	Não rejeita	Rejeita
2	S-W	0,872596813	0,872596813	0,457624202	0,872596813	0,272310185
	p	4,65E-11	4,65E-11	6,25E-23	4,65E-11	6,51E-26
	h0	Rejeita	Rejeita	Rejeita	Rejeita	Rejeita
3	S-W	0,966144537	0,966144537	0,628956821	0,966144537	0,511700026
	p	0,000338008	0,000338008	3,80E-19	0,000338008	1,13E-21
	h0	Rejeita	Rejeita	Rejeita	Rejeita	Rejeita
4	S-W	0,971885011	0,971885011	0,860678009	0,971885011	0,574425339
	p	0,029540019	0,029540019	2,66E-08	0,029540019	1,20E-15
	h0	Rejeita	Rejeita	Rejeita	Rejeita	Rejeita
5	S-W	0,985542734	0,985542734	0,958190118	0,985542734	0,203871904
	p	0,314026847	0,314026847	0,002209977	0,314026847	2,71E-21
	h0	Não rejeita	Não rejeita	Rejeita	Não rejeita	Rejeita
6	S-W	0,935776885	0,935776885	0,886600808	0,935776885	0,566146102
	p	0,030615362	0,030615362	0,00108981	0,030615362	2,06E-09
	h0	Rejeita	Rejeita	Rejeita	Rejeita	Rejeita

Fonte: elaborado pelo autor

Para utilizar-se uma referência para análise do comportamento do modelo, serão utilizadas comparações dos dados obtidos pelo modelo e as avaliações humanas. As análises e comparações serão realizadas a partir das correlações, da benevolência, que é a propensão do modelo atribuir uma *score* maior ou menor que os avaliadores, e da diferença entre a habilidade média para cada nota.

O estudo das correlações entre o modelo e os avaliadores humanos objetiva determinar se há uma relação de associação entre o comportamento das notas atribuídas pelos avaliadores e a habilidade calculada pelo modelo. Observou-se conforme apresentado nas seções anteriores que a correlação entre os avaliadores é de média a forte, conforme Tabela 11. O teste não paramétrico Tau, de Kendall, será utilizado da mesma forma que nas seções anteriores, segundo as mesmas referências de intensidade.

Para o cálculo das correlações utilizou-se a função `cor.test()` do RStudio, com o parâmetro *kendall*. Analisa-se o valor Tau, que determina a intensidade da correlação, e o valor *p*, que

determina a significância do teste para a amostra, servindo de referência para indicar a presença de evidências que permitam a rejeição da hipótese nula de que os dados são independentes. A Tabela 14 apresenta as correlações calculadas entre o modelo proposto e as notas atribuídas pelo avaliador 1, já a Tabela 15 apresenta as correlações entre o modelo e o avaliador 2.

Tabela 14: Correlações Modelo x Avaliador 1

Problema	Legenda	Competências				
1		data_analysis	data_representation	problem_decomposition	abstraction	simulation
	tau	0,494942725	0,511394145	0,575361302	0,575368106	0,513103117
	p	1,25E-08	1,61E-09	6,90E-11	1,53E-11	7,76E-09
	h0	Rejeita	Rejeita	Rejeita	Rejeita	Rejeita
	Intensidade	Média	Forte	Forte	Forte	Forte
2	tau	0,467609334	0,420689033	0,250875318	0,609428475	0,404355601
	p	7,23E-13	1,50E-11	0,000114404	5,69E-24	3,42E-10
	h0	Rejeita	Rejeita	Rejeita	Rejeita	Rejeita
		Média	Média	Fraco	Forte	Média
3	tau	0,603877457	0,574565552	0,397525917	0,618620194	0,418457597
	p	3,22E-23	1,37E-21	1,70E-10	3,44E-25	4,77E-11
	h0	Rejeita	Rejeita	Rejeita	Rejeita	Rejeita
		Forte	Forte	Média	Forte	Média
4	tau	0,456005716	0,412374956	0,235358998	0,575378778	0,390601976
	p	7,75E-09	1,03E-07	0,004781373	1,39E-13	2,01E-06
	h0	Rejeita	Rejeita	Rejeita	Rejeita	Rejeita
		Média	Média	Fraco	Forte	Média
5	tau	0,545599537	0,386207634	0,224623455	0,582442936	0,419116477
	p	3,63E-12	6,68E-07	0,006025687	1,49E-14	2,46E-07
	ho	Rejeita	Rejeita	Rejeita	Rejeita	Rejeita
		Forte	Média	Fraco	Forte	Média
6	tau	0,622694908	0,665988402	0,480741102	0,76359387	0,508697392
	p	1,38E-06	1,18E-07	0,000166953	1,01E-09	0,000125646
	ho	Rejeita	Rejeita	Rejeita	Rejeita	Rejeita
		Forte	Forte	Média	Forte	Forte

Fonte: elaborado pelo autor

Tabela 15: Correlações Modelo x Avaliador 2

Problema	Legenda	Competências				
1		data_analysis	data_representation	problem_decomposition	abstraction	simulation
	tau	0,285134663	0,546448156	0,636431607	0,261732664	0,443662308
	p	0,000895726	4,77E-11	1,73E-13	0,001669373	6,96E-07
	h0	Rejeita	Rejeita	Rejeita	Rejeita	Rejeita
	Intensidade	Fraco	Forte	Forte	Fraco	Média
2	tau	0,505636416	0,385896066	0,418001088	0,510838601	0,11361223
	p	9,88E-16	2,15E-09	1,18E-10	1,30E-15	0,077553964
	h0	Rejeita	Rejeita	Rejeita	Rejeita	Não rejeita
	Intensidade	Forte	Média	Média	Forte	Fraco
3	tau	0,660737588	0,616863733	0,458963521	0,398467672	0,281782669
	p	1,58E-26	2,05E-24	2,76E-13	3,28E-10	8,78E-06
	h0	Rejeita	Rejeita	Rejeita	Rejeita	Rejeita
	Intensidade	Forte	Forte	Média	Média	Fraco
4	tau	0,631101432	0,50528655	0,321521716	0,58958336	0,209123951
	p	1,66E-15	2,69E-10	0,000108815	5,59E-13	0,011122595
	h0	Rejeita	Rejeita	Rejeita	Rejeita	Rejeita
	Intensidade	Forte	Forte	Média	Forte	Fraco
5	tau	0,697724662	0,520942718	0,350659116	0,484820009	0,261096146
	p	5,87E-20	1,30E-11	1,73E-05	1,53E-09	0,001340205
	ho	Rejeita	Rejeita	Rejeita	Rejeita	Rejeita
	Intensidade	Forte	Forte	Média	Média	Fraco
6	tau	0,695236076	0,781885793	0,637084129	0,704086419	0,294619371
	p	2,10E-08	2,58E-10	1,59E-06	2,01E-08	0,030928921
	ho	Rejeita	Rejeita	Rejeita	Rejeita	Rejeita
	Intensidade	Forte	Forte	Forte	Forte	Fraco

Fonte: elaborado pelo autor

Observou-se indícios de que os dados gerados pelo modelo e as notas dos avaliadores possuem uma correlação de média a forte nas competências *Análise de Dados*, *Representação de Dados*, e *Abstração*. Ainda que tenha-se encontrado indícios para rejeitar a hipótese nula na maioria das competências (exceto uma), as correlações da *Decomposição do Problema* com o avaliador 1, e *Simulação* com o avaliador 2 apresentaram uma intensidade fraca em alguns casos. Espera-se que ocorra a associação quando os comportamentos das variáveis forem similares em relação ao seu sentido, enquanto uma cresce a outra também deve crescer, mas não tanto quanto a sua intensidade, ou seja, não é necessário que o crescimento seja equivalente. Para o entendimento da baixa intensidade das correlações, realizou-se uma análise pontual a respeito do comportamento da associação das competências *Decomposição* e *Simulação*.

Analisando manualmente os códigos fontes que geraram os resultados discrepantes na

comparação, observou-se que para a competência *Decomposição do Problema* a frequência das declarações de bibliotecas e cabeçalhos (*includes*) não se mostrou adequada para a análise. Da forma como estão elaboradas, as rotinas criadas não fazem distinção entre bibliotecas que possuem recursos que são utilizados no código fonte daquelas que não possuem. O avaliador humano consegue fazer essa distinção com facilidade. Uma das características que observou-se em muitos códigos foi a de possuírem várias bibliotecas declaradas mas não utilizadas, atribui-se ao fato de ser uma prática comum os programadores inicializarem seus códigos com um padrão, que pode conter bibliotecas usuais e funções pré definidas, mesmo que não funcionais para a solução do problema proposto. Dessa forma, o modelo foi ajustado para não contabilizar o uso de bibliotecas na frequência da competência *Decomposição do Problema*.

Uma situação similar ocorre para a competência *Simulação*, também gerando uma diferença entre os avaliadores e o modelo. Espera-se que os comentários presentes no código fonte sejam parte de uma informação útil, como documentação ou saídas de controle e depuração, como de fato ocorreu na maioria dos casos. Entende-se como saídas de controle trechos de código que informem ao usuário a informação de uma variável em uma determinada situação, é um recurso de depuração utilizado por alguns programadores, uma vez que não haja mais a necessidade dessa informação, esse trecho de código é excluído ou comentado. Da forma como estão elaboradas as rotinas do presente trabalho, não é possível diferenciar os trechos de código comentado entre aqueles que realmente atendem ao comportamento esperado daqueles que tem outro fim. O avaliador humano consegue fazer essa diferenciação, o que nos casos de uso inesperado do recurso, resulta em uma avaliação mais adequada da competência. Mesmo sem a diferenciação adequada do uso os resultados obtidos pelo modelo foram considerados satisfatórios, e o uso dos comentários foram mantidos.

Percebeu-se no decorrer dos experimentos uma diferença entre a relevância utilizada pelo modelo, a partir da TF-IDF, e o critério de relevância dos avaliadores. Entende-se que mesmo entre os avaliadores existe uma diferença aparente de critérios para atribuição de uma nota, toma-se como exemplo a competência *Coleta de Dados*, cujo resultado são duas notas muito diferentes. Não foi possível identificar de que forma os avaliadores estabeleceram seus critérios para gerar a divergência de notas, pois não há variação nos tokens utilizados para *Coleta de Dados*. Ainda, mesmo com a variação de tokens relacionados com as demais competências a nota se manteve constante. O modo como o modelo estabelece a relevância pode ser diferente dos avaliadores, mas certamente ele é constante e bem definido.

Outra ocorrência observável na avaliação humana é a variação dos critérios mesmo em códigos similares do ponto de vista estrutural. Encontrou-se casos nos quais códigos similares receberam notas diferentes, entende-se que essa diferença está relacionada à ordem na qual os códigos são avaliados. Para os avaliadores a relevância parece estar restrita ao conjunto de submissões já conhecidas, dessa forma, conforme o avaliador progride na lista de códigos a serem avaliados, suas referências são sempre aquelas já conhecidas por ele.

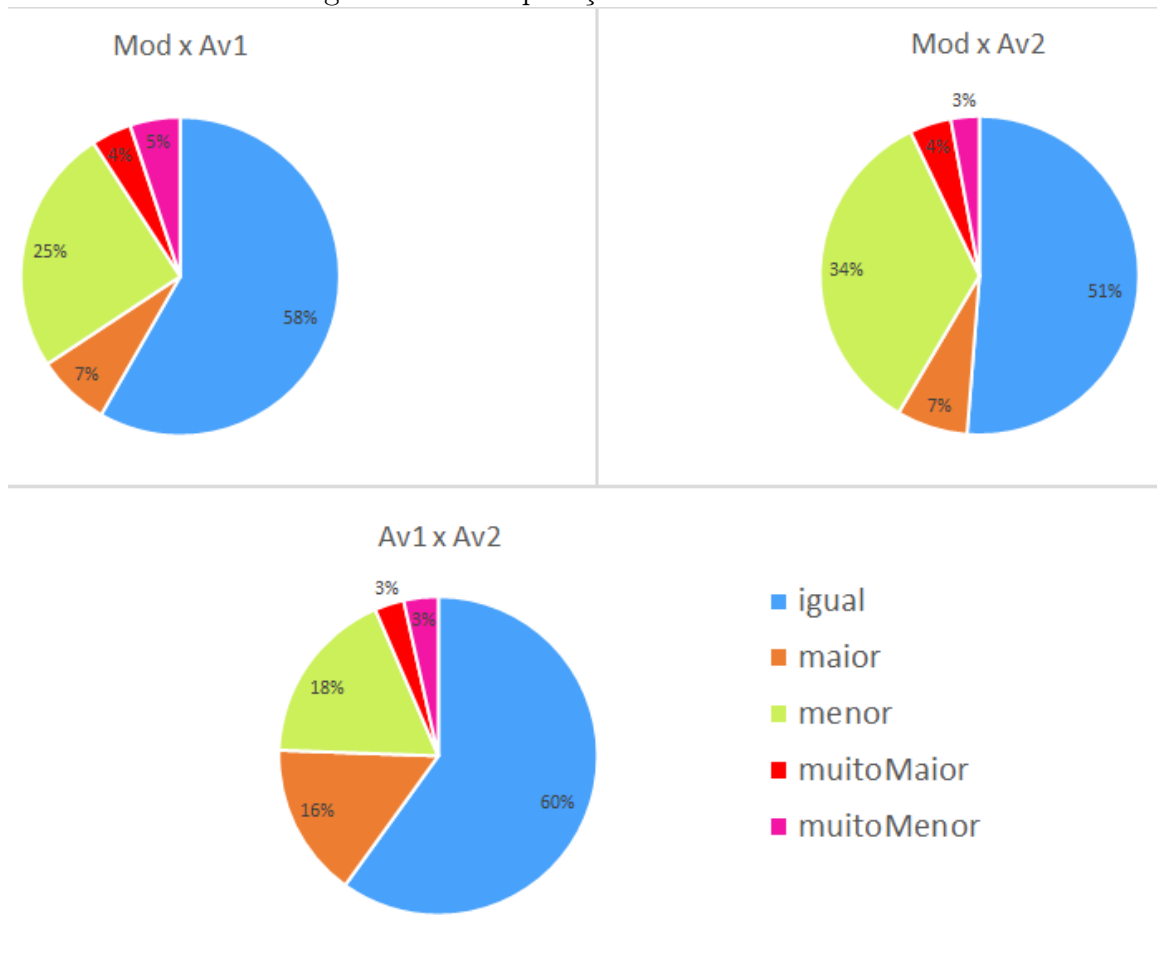
No entanto, é possível que ele encontre uma solução bastante diferente das anteriores e, no caso dessa solução ser considerada de maior qualidade para uma competência, ela passa a

ser a referência das avaliações a serem realizadas. Assim, a cada novo modelo de referência encontrado pelos avaliadores pode ocorrer variações nos critérios utilizados. Na ocorrência de códigos similares estarem em posições diferentes de uma nova referência, um antes e outro depois, podem haver variações nas notas dadas a esses códigos que, em teoria, deveriam receber pontuações similares. Para o modelo, a relevância considera o contexto de todas as submissões e é influenciado pelos tokens não só da competência analisada, mas também pelos demais presentes no conjunto de soluções.

Portanto, é notável que os avaliadores humanos tomaram como referência alguns códigos fonte considerados mais adequados, mas ainda assim os critérios utilizados para as suas notas não são tão claros quanto à interação entre as competências. Para os avaliadores, a qualidade empregada para a elaboração da solução parece estar diretamente ligada ao número de tokens da competência em análise na submissão. Nesse contexto é mais fácil para os avaliadores compararem diferentes submissões quanto a uma competência específica, comparando-se o número de tokens entre uma submissão e outra, mas ao utilizar-se tão somente do número de tokens, perde-se a referência para a relevância da interação de uma competência com as demais.

Aparentemente, o resultado prático é que os avaliadores humanos analisam as competências separadamente, enquanto que o modelo faz essa análise em conjunto, dessa forma as habilidades calculadas pelo modelo são muitas vezes menores que as atribuídas pelos avaliadores humanos. A Figura 4.7 apresenta uma comparação entre as benevolências, no primeiro e segundo quadros toma-se como referência o modelo e compara-se com os avaliadores um e dois respectivamente, o quadro inferior é a comparação entre os dois avaliadores.

Figura 4.7: Comparação das benevolências

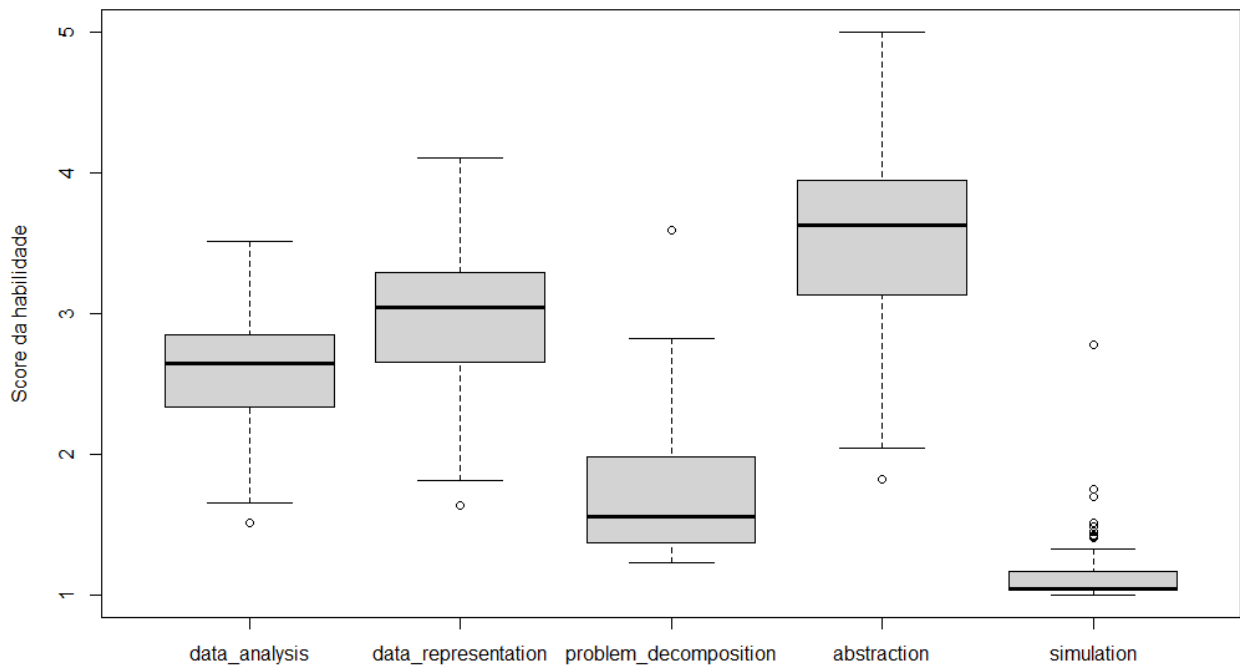


Fonte: elaborado pelo autor

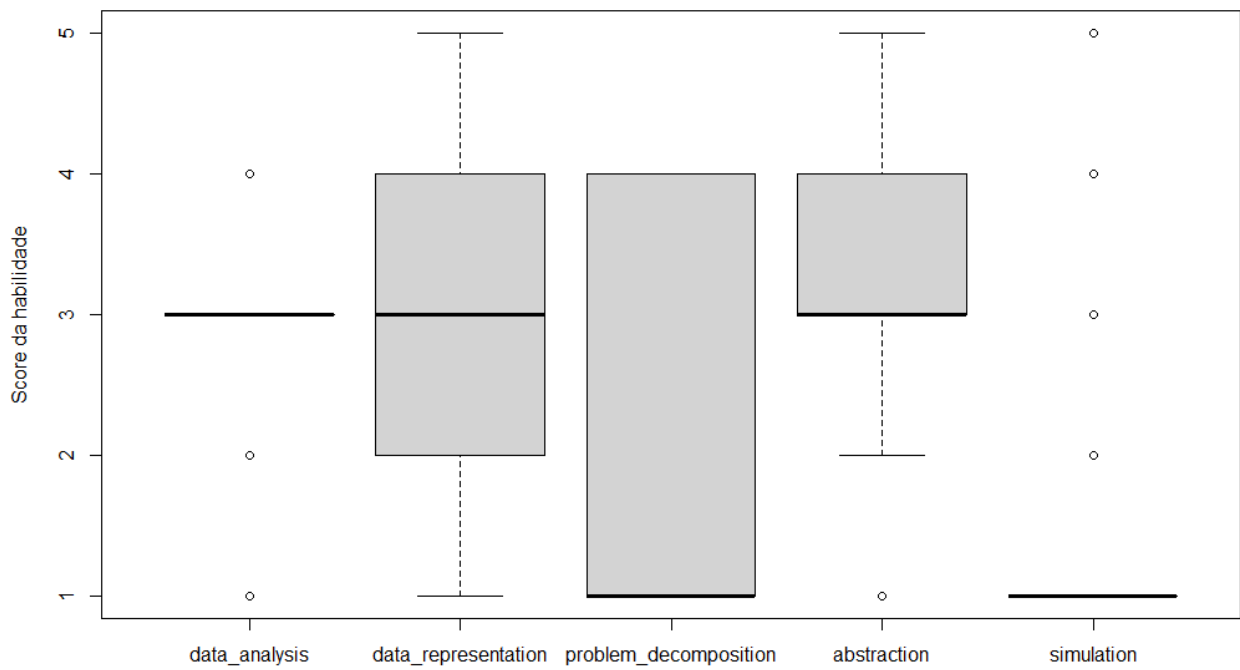
Observando as amplitudes das notas atribuídas pelos avaliadores em relação ao *score* do modelo, entende-se que o modelo fornece informações mais detalhadas sobre as competências envolvidas nas elaborações dos códigos fontes. A Figura 4.8 apresenta as competências do problema 1 calculadas pelo modelo e pelo avaliador 1. A metodologia utilizada para coletar as notas dos avaliadores (inteiros de 1 a 5) impossibilita uma escala tão detalhada quanto aquela calculada pelo modelo, ainda assim ficam evidentes as diferenças de amplitudes das avaliações. Pode-se observar indícios de que a variação do *score* gerado pelo modelo é mais suave em relação as notas do avaliador. Entende-se que essa suavização está relacionada com a relevância, para o modelo mesmo que uma competência tenha um desempenho ótimo, se ela não for relevante no contexto geral, não receberá um *score* alto.

Ainda sobre as diferenças do *score* do modelo e das notas atribuídas pelos avaliadores, pode-se dizer que mesmo que os valores atribuídos sejam menores, a tendência de variação de valores é semelhante, o que é corroborado também pelas correlações. A Figura 4.9 apresenta o box-plot do problema 6 (a), assim como a média do *score* gerado pelo modelo para cada uma das notas dos avaliadores (b e c).

Figura 4.8: Box-plot Modelo (a) e do Avaliador 1 (b) do problema 1



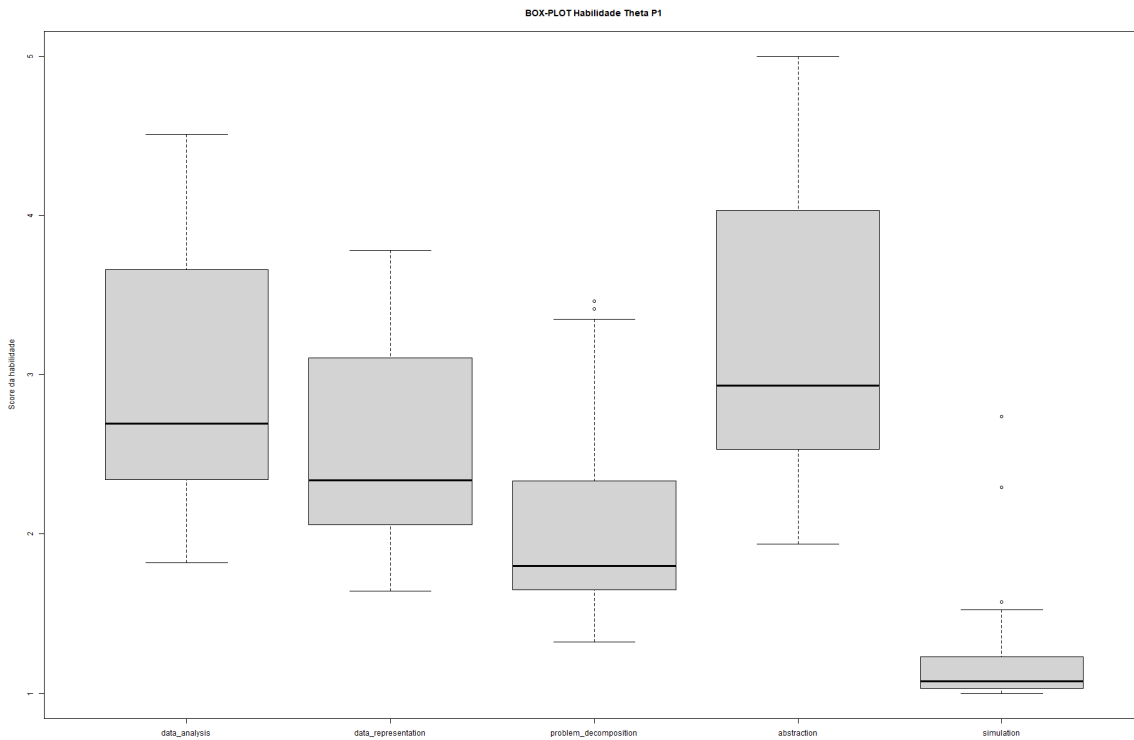
(a) Box-plot Problema 1 - modelo



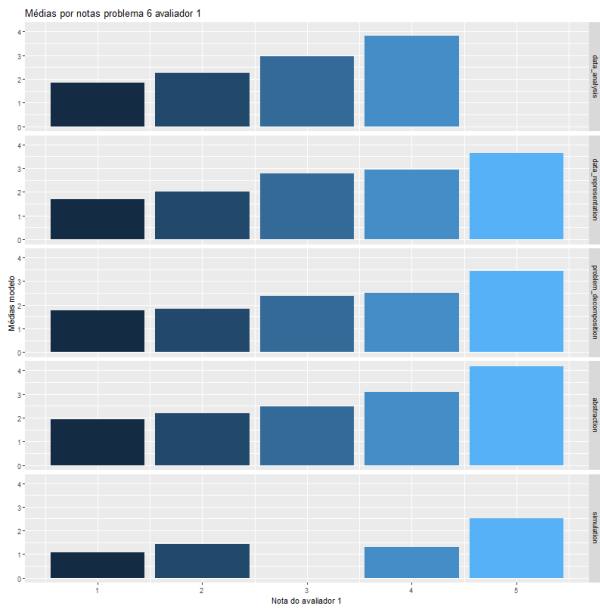
(b) Box-plot problema 1 - Avaliador 1

Fonte: elaborado pelo autor

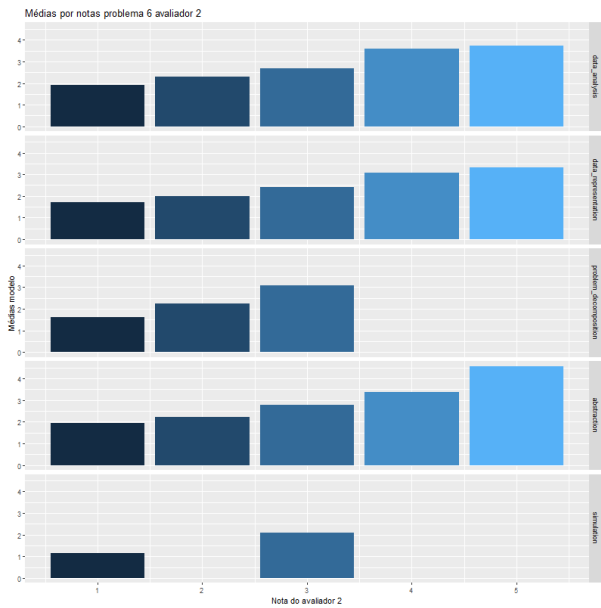
Figura 4.9: Box-plot Modelo - problema 6 e médias por nota dos avaliadores



(a) Box-plot Problema 6 - modelo



(b) Médias do modelo por nota do avaliador 1



(c) Médias do modelo por nota do avaliador 2

Fonte: elaborado pelo autor

Utilizando-se de um arredondamento simples para o *score* do modelo, e comparando-o ao resultado dos avaliadores identificou-se que a incidência de valores iguais entre o modelo e os avaliadores é maior justamente nas competências que obtiveram as menores correlações, conforme a Tabela 16 que apresenta a comparação dos resultados obtidos pelo modelo com

os atribuídos pelos avaliadores. Utilizou-se esse arredondamento para aproximar a escala dos *scores* ao formato utilizado pelos avaliadores.

Tabela 16: Benevolência por habilidade

Modelo vs Av1					
habilidade	Resultado				
	igual	maior	menor	muitoMaior	muitoMenor
data_analisys	48%	10%	38%	3%	1%
data_representaion	42%	15%	30%	11%	2%
problem_decomposition	87%	0%	6%	0%	6%
abstraction	26%	11%	43%	7%	13%
simulation	88%	1%	8%	0%	3%

Modelo vs Av2					
habilidade	Resultado				
	igual	maior	menor	muitoMaior	muitoMenor
data_analisys	22,02%	4,52%	68,78%	2,11%	2,56%
data_representaion	43,14%	10,11%	36,20%	9,35%	1,21%
problem_decomposition	83,26%	0,30%	15,84%	0,00%	0,60%
abstraction	18,85%	19,46%	43,74%	9,35%	8,60%
simulation	88,99%	1,81%	7,39%	0,30%	1,51%

Fonte: elaborado pelo autor

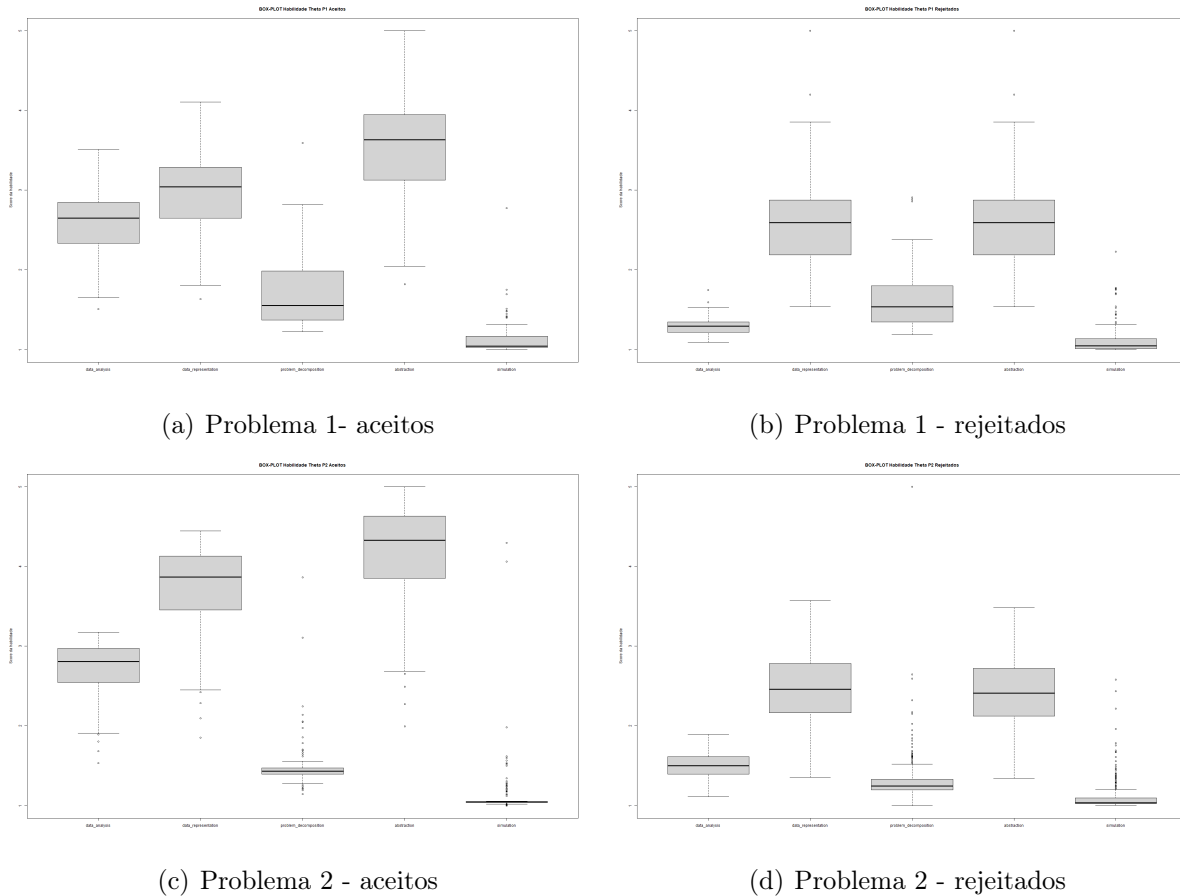
As duas competências, *Decomposição do Problema* e *Simulação*, possuem duas características que julgou-se importante para análise: a primeira é que essas competências são aquelas com a menor frequência de termos na grande maioria das submissões; a segunda é que elas possuem os conceitos mais claros para definição da competência, ou seja, são os mais fáceis de identificar em um código fonte.

Essas duas características podem ser usadas para explicar tanto o alto número (em relação às demais competências) de resultados iguais quanto as baixas correlações. Para o *score* calculado normalmente existe variação de valores, pois ele considera a relevância para o conjunto de submissões como um todo, enquanto que para a avaliação humana, na ausência de evidências que indiquem uma maior competência na elaboração da solução, o valor é normalmente o mesmo, fixo.

Portanto, por um lado utilizando-se o arredondamento encontramos valores iguais pois a avaliação é fixa, por outro como não utiliza-se o arredondamento nas correlações a variação prejudica o desempenho. Em um terceiro efeito, quando o avaliador encontra evidências dessas duas competências a amplitude da variação da nota é sempre maior, devido aos efeitos da interpretação da relevância discutidos anteriormente. Observar o comportamento médio dos dados gerados pelo modelo pode fornecer informações importantes a respeito das competências

envolvidas na elaboração das soluções dos problemas. A Figura 4.10 apresenta uma comparação gráfica dos conjuntos de soluções aceitos como corretos (à esquerda) e o conjunto de soluções rejeitadas (à direita).

Figura 4.10: Box-plot Aceitos x Rejeitados

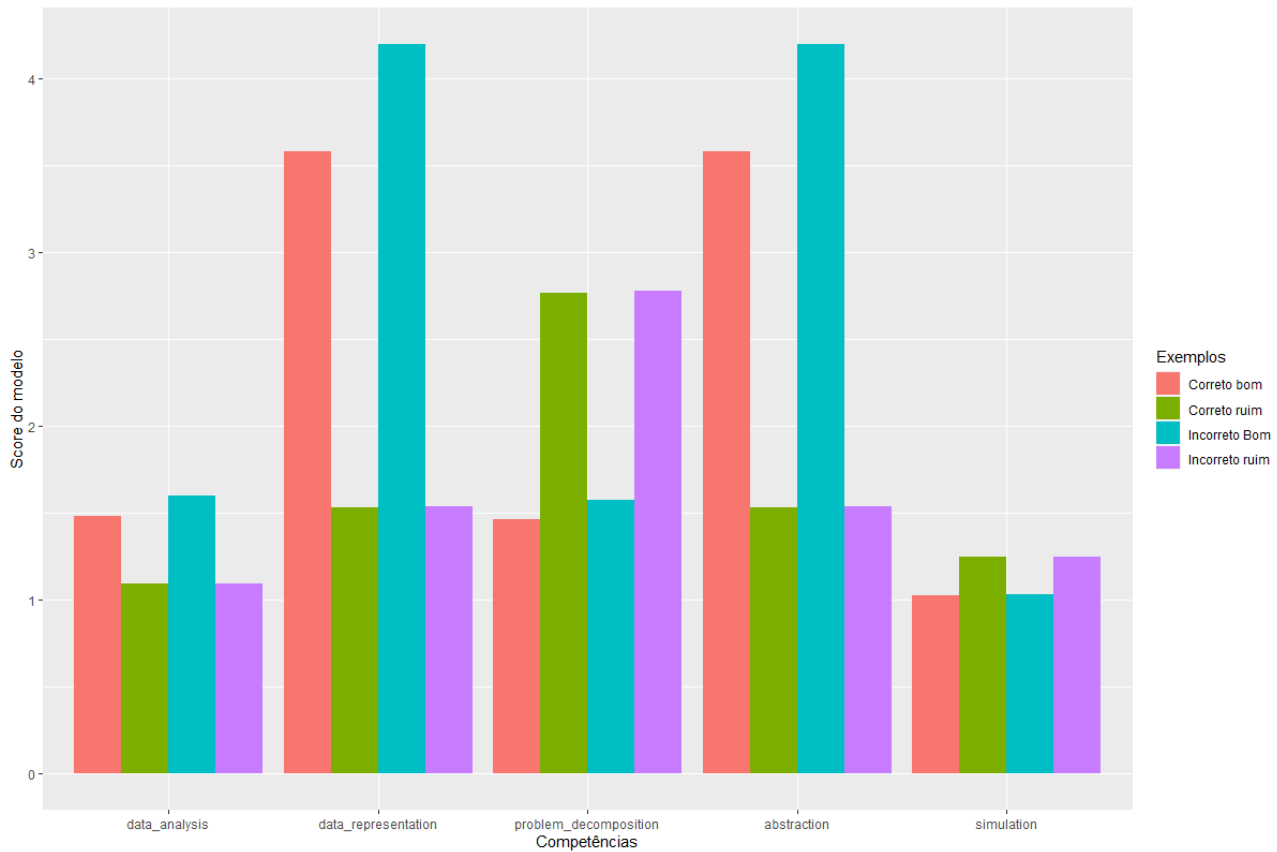


Fonte: elaborado pelo autor

Pode-se observar que soluções corretas e rejeitadas compartilham uma tendência de uso das competências para a elaboração das soluções, no entanto é notável que as soluções julgadas corretas possuem uma intensidade maior para algumas dessas competências, como por exemplo o problema um, na qual o *score* das competências *Análise de Dados* e *Abstração* são significativamente maior nas soluções aceitas em relação as soluções rejeitadas.

Percebe-se então que, para o problema 1, há indícios de que o domínio das competências *Análise de Dados* e *Abstração* contribui para a elaboração de soluções bem sucedidas, considerou-se assim que há evidências de uma competência influente. No entanto, não é possível afirmar que essas competências são determinantes para a solução do problema, uma vez que observou-se soluções que utilizavam níveis baixos dessas competências (e mesmo níveis baixos em todas as competências) e ainda assim foram consideradas bem sucedidas. A Figura 4.11 apresenta uma comparação entre códigos considerados corretos com níveis de competência bons e ruins, outros incorretos também com níveis considerados bons e ruins.

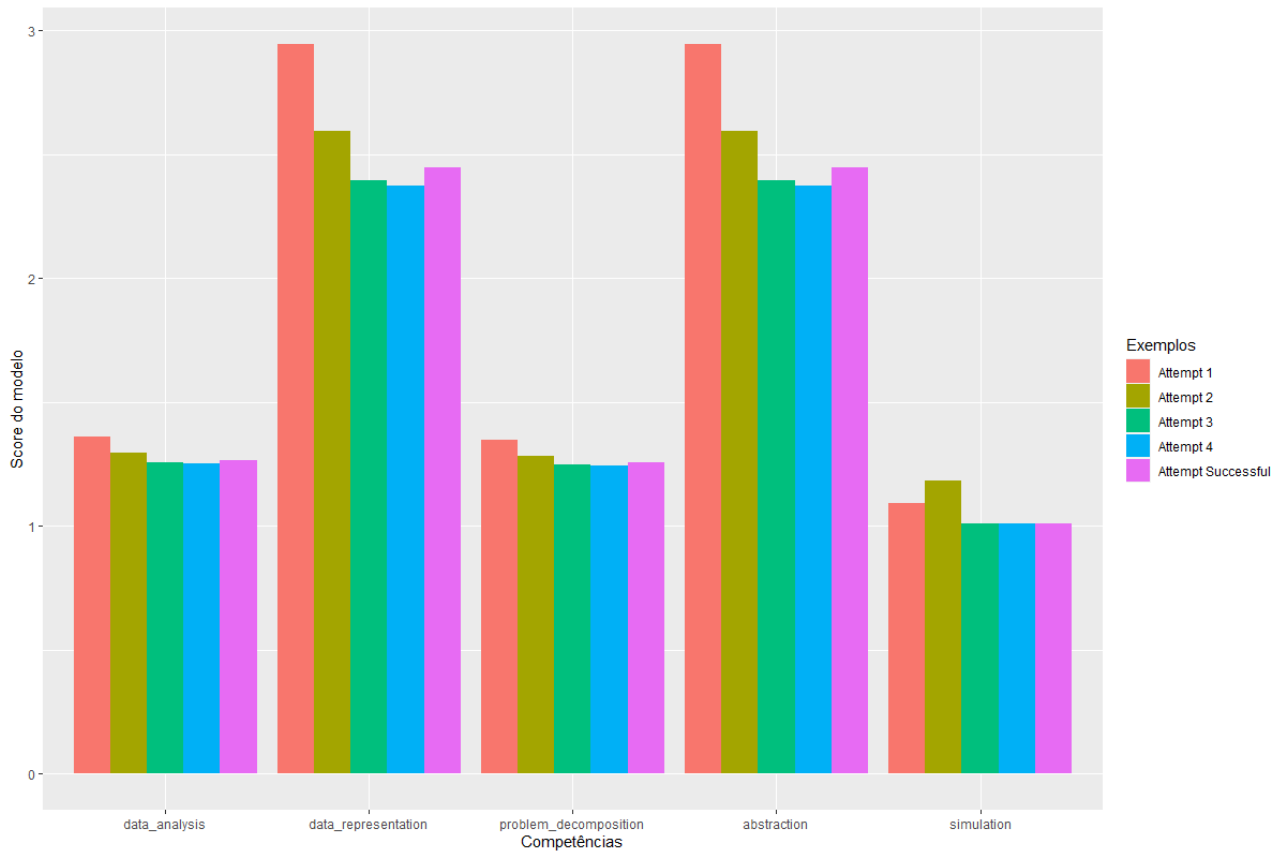
Figura 4.11: Comparação entre submissões corretas e incorretas



Fonte: elaborado pelo autor

O domínio de uma competência não é a causa primária do erro, ou mesmo do acerto, em programação o detalhe é fundamental a ponto de um simples caractere como um ponto e vírgula determinarem o sucesso ou fracasso da solução. A Figura 4.12 apresenta uma sequência de submissões de um mesmo usuário para o problema um, nota-se que conforme o usuário submete novos códigos seu score tem uma leve queda, isso se deve à necessidade de acrescentar elementos a sua solução, tornando-a menos eficiente frente às soluções com melhor desempenho, ainda assim a figura traz indícios de que o limite entre um código bem sucedido e um incorreto é tênue.

Figura 4.12: Sequência de submissões



Fonte: elaborado pelo autor

De forma geral, percebe-se que em ordem de grandeza o modelo produz valores compatíveis com os dos avaliadores humanos, com a tendência de serem menores. Através da análise dos dados obtidos, considera-se que essa é uma vantagem em relação à avaliação humana, pois os dados apontam indícios de que o modelo consegue inferir uma relevância mais consistente, que considera o conjunto de dados como um todo e não está sujeito a variação de interpretações subjetivas. Entende-se ainda que o modelo possa ser utilizado para identificar competências que são mais influentes no sucesso de uma solução, assim como pode indicar o nível das competências utilizadas na elaboração de códigos fonte, podendo ser utilizado como ferramenta para auxílio da avaliação e acompanhamento no desenvolvimento de educandos.

5 CONSIDERAÇÕES FINAIS

A avaliação de códigos fonte em programação de computadores tem sido amplamente discutida em diversos aspectos e aplicações, entre eles o propósito educacional. No presente trabalho apresentou-se um modelo para avaliação de códigos fonte para a identificação de competências do Pensamento Computacional, na busca de elementos que auxiliem no ensino e aprendizado de programação de computadores.

Como embasamento teórico, apresentou-se trabalhos relacionados com metodologias para análise de códigos fonte, e observou-se que enquanto a avaliação dinâmica proporciona um modelo prático de aferir a correção de uma solução, em educação é frequentemente mais compensatório olhar além de um resultado que indique apenas se uma resposta está certa ou errada, e dessa forma optou-se por uma metodologia que utilize uma avaliação estática.

A fim de possibilitar a aplicação de metodologias estatísticas para a análise estática dos códigos, também tomou-se como referência outros trabalhos relacionados com metodologias de transformação de códigos fonte em uma representação numérica. Buscou-se na literatura um conjunto de competências que pudessem ser associadas à estruturas presentes em códigos fonte, escolheu-se utilizar Pensamento Computacional como referência, e a partir dos trabalhos relacionados, foi proposto uma metodologia para identificar conceitos de PC em códigos fonte. Inspirados em metodologias de recuperação de informação em textos escritos em linguagem natural, foi proposto um modelo de avaliação de competências do Pensamento Computacional em códigos fonte utilizando-se a TF-IDF e uma medida estatística que denominou-se de *Habilidade Theta*.

Para a validação do modelo proposto, elaborou-se um experimento com uma base de dados composta por códigos fonte. Foram implementadas as rotinas necessárias para coleta, processamento e organização dos dados segundo o modelo proposto. Também realizou-se a avaliação de parte da base de dados, correspondente às soluções consideradas corretas, e concluiu-se que em ordem de grandeza os resultados gerados pelo modelo são compatíveis com aqueles gerados pela avaliação humana.

Os resultados obtidos também revelaram ser possível identificar e estimar níveis de qualidade de competências em códigos fonte. A partir da análise dos resultados, observou-se diversidade possível de respostas aceitáveis para um mesmo problema em exercícios de programação. Em comparação com outros modelos de avaliação, os quais fazem uso de uma resposta modelo, o modelo proposto no presente trabalho apresenta a vantagem de adequar-se às diferentes respostas presentes no conjunto, permitindo observar-se as diferentes abordagens que os programadores utilizaram na elaboração de suas soluções a partir da análise da diferença das competências empregadas.

Conclui-se, portanto, que os programadores podem fazer uso de diferentes competências e em diferentes níveis, para elaborar uma solução aceitável. Também é possível concluir que nem sempre o nível da competência é determinante para o sucesso ou insucesso, pois em geral soluções consideradas corretas e aquelas consideradas incorretas podem ter valores iguais para

as competências. Quanto à avaliação utilizando-se o modelo individualmente para as submissões, conclui-se que demonstrou-se eficiente ao determinar a importância de cada habilidade para solução. Para a análise do conjunto de soluções, foi possível determinar um nível médio para as competências utilizadas na elaboração de códigos fontes do conjunto.

Espera-se que as informações extraídas a partir dessa metodologia possam ser utilizadas para fins educacionais, de modo a contribuir para o desenvolvimento de metodologias educacionais voltadas ao acompanhamento do desenvolvimento das competências e perfil de estudantes em programação de computadores. Como trabalhos futuros almeja-se o desenvolvimento contínuo do modelo, integrando-se a outras tecnologias de avaliação, assim como a incorporação do modelo à um sistema de recomendação de exercícios de programação em um contexto educacional.

LISTA DE PUBLICAÇÕES

DE SOUZA, Ricardo Lemos; FERREIRA, Fabiana Zaffalon; DA COSTA BOTELHO, Silvia Silva. Proposta para Avaliação de Códigos Fonte com TF-IDF. In: Anais do XXXI Simpósio Brasileiro de Informática na Educação. SBC, 2020. p. 112-121.

DE SOUZA, Ricardo Lemos; FERREIRA, Fabiana Zaffalon; DA SILVA BOTELHO, Silvia. A Proposal for Source Code Assessment Through Static Analysis. In: 2020 IEEE Frontiers in Education Conference (FIE). IEEE, 2020. p. 1-5.

Trabalho no prelo

DE SOUZA, Ricardo Lemos; FERREIRA, Fabiana Zaffalon; DA SILVA BOTELHO, Silvia. Static analysis model for assessing source codes with TFIDF. In: 2021 IEEE Frontiers in Education Conference (FIE).

Referências

- ALA-MUTKA, K. M. A survey of automated assessment approaches for programming assignments. **Computer science education**, Taylor & Francis, v. 15, n. 2, p. 83–102, 2005.
- ALBANO, A. **Introduction to educational and psychological measurement using R**. 2017.
- ALVES, N. d. C. **Codemaster: Um modelo de avaliação do pensamento computacional na educação básica através da análise de código de linguagem de programação visual**. 143 p. Dissertação (Mestrado) — Programa de Pós-graduação em Ciências da Computação, Universidade Federal de Santa Catarina, Florianópolis, 2013.
- ANTONIOL, G.; CANFORA, G.; CASAZZA, G.; LUCIA, A. D.; MERLO, E. Recovering traceability links between code and documentation. **IEEE transactions on software engineering**, IEEE, v. 28, n. 10, p. 970–983, 2002.
- ARIFI, S. M.; ABDELLAH, I. N.; ZAHI, A.; BENABBOU, R. Automatic program assessment using static and dynamic analysis. **2015 Third World Conference on Complex Systems (WCCS)**, 2015.
- AZCONA, D.; ARORA, P.; HSIAO, I.-H.; SMEATON, A. user2code2vec: Embeddings for profiling students based on distributional representations of source code. In: **Proceedings of the 9th International Conference on Learning Analytics & Knowledge**. [S.l.: s.n.], 2019. p. 86–95.
- BARR, V.; STEPHENSON, C. Bringing computational thinking to k-12: what is involved and what is the role of the computer science education community? **Acm Inroads**, ACM New York, NY, USA, v. 2, n. 1, p. 48–54, 2011.
- BERMEJO, P. H.; TONELLI, A. O.; GALLIERS, R. D.; OLIVEIRA, T.; ZAMBALDE, A. L. d. S. Conceptualizing organizational innovation: The case of the brazilian software industry. **Information & Management**, Elsevier, v. 53, n. 4, p. 493–503, 2016.
- BEZ, J. L.; TONIN, N. A.; RODEGHERI, P. R. Uri online judge academic: A tool for algorithms and programming classes. In: IEEE. **2014 9th International Conference on Computer Science & Education**. [S.l.], 2014. p. 149–152.
- BOYCE, B. R.; KRAFT, D. H.; MEADOW, C. T.; KRAFT, D. H.; BOYCE, B. R.; MEADOW, C. T. **Text information retrieval systems**. [S.l.]: Elsevier, 2017.
- BRASIL. **Exame Nacional do Ensino Médio (ENEM): Fundamentação teórico metodológica**. Brasília/DF, 2005. Disponível em: <<http://portal.inep.gov.br/documents/186968/484421/>>. Acesso em: 10.10.2020.
- BRASIL, M. Diretrizes curriculares de cursos da área de computação e informática. **Resolução Nº 5, de 16 de novembro DE 2016, (MEC–SESU (Secretaria de Educação Superior). CEEInf–Comissão de Especialistas de Ensino de Computação e Informática**. Brasília, 2016. Disponível em: <http://portal.mec.gov.br/index.php?option=com_docman&view=download&alias=52101-rces005-16-pdf&category_slug=novembro-2016-pdf&Itemid=30192>. Acesso em: 10.10.2020.

BRENNAN, K.; RESNICK, M. New frameworks for studying and assessing the development of computational thinking. In: **Proceedings of the 2012 annual meeting of the American educational research association, Vancouver, Canada**. [S.l.: s.n.], 2012. v. 1, p. 25.

CABO, C. Student progress in learning computer programming: Insights from association analysis. In: IEEE. **2019 IEEE Frontiers in Education Conference (FIE)**. [S.l.], 2019. p. 1–8.

CAPITÁN, L.; VOGEL-HEUSER, B. Metrics for software quality in automated production systems as an indicator for technical debt. In: IEEE. **2017 13th IEEE Conference on Automation Science and Engineering (CASE)**. [S.l.], 2017. p. 709–716.

CAVALCANTE, A.; COSTA, L. D. S.; ARAUJO, A. L. Um estudo de caso sobre competências do pensamento computacional desenvolvidas na programação em blocos no code. org. In: **Anais dos Workshops do Congresso Brasileiro de Informática na Educação**. [S.l.: s.n.], 2016. v. 5, n. 1, p. 1117.

CHANG, Z. e. a. Scratch analysis tool (sat): a modern scratch project analysis tool based on antlr to assess computational thinking skills. **2018 14th International Wireless Communications & Mobile Computing Conference (IWCMC)**. IEEE, p. 950–955, 2018.

CHRYSAFIADI, K.; VIRVOU, M. Persiva: An empirical evaluation method of a student model of an intelligent e-learning environment for computer programming. **Computers & Education**, Elsevier, v. 68, p. 322–333, 2013.

DIVYA, K.; SUBHA, R.; PALANISWAMI, S. Similar words identification using naive and tf-idf method. **International Journal of Information Technology and Computer Science (IJITCS)**, v. 6, n. 11, p. 42, 2014.

FU, D.; XU, Y.; YU, H.; YANG, B. Wastk: a weighted abstract syntax tree kernel method for source code plagiarism detection. **Scientific Programming**, Hindawi, v. 2017, 2017.

GANGULY, D.; JONES, G. J.; RAMÍREZ-DE-LA-CRUZ, A.; RAMÍREZ-DE-LA-ROSA, G.; VILLATORO-TELLO, E. Retrieving and classifying instances of source code plagiarism. **Information Retrieval Journal**, Springer, v. 21, n. 1, p. 1–23, 2018.

GOMES, A.; HENRIQUES, J.; MENDES, A. Uma proposta para ajudar alunos com dificuldades na aprendizagem inicial de programação de computadores. **Educação, Formação & Tecnologias-ISSN 1646-933X**, v. 1, n. 1, p. 93–103, 2008.

HAIDUC, S.; APONTE, J.; MORENO, L.; MARCUS, A. On the use of automated text summarization techniques for summarizing source code. In: IEEE. **2010 17th Working Conference on Reverse Engineering**. [S.l.], 2010. p. 35–44.

HALSTEAD, M. H. Elements of software science (operating and programming systems series). **Elsevier Science Inc.**, New York, NY, USA, 1977.

HARMAN, M. Why source code analysis and manipulation will always be important. In: IEEE. **2010 10th IEEE Working Conference on Source Code Analysis and Manipulation**. [S.l.], 2010. p. 7–19.

JONES, K. S. A statistical interpretation of term specificity and its application in retrieval. **Journal of documentation**, MCB UP Ltd, 1972.

- KARNALIM, O. Tf-idf inspired detection for cross-language source code plagiarism and collusion. **Computer Science**, v. 21, n. 1, 2020.
- KENSKI, V. M. Avaliação da aprendizagem. **VEIGA, Ilma Passos de Alencastro. (Coord). Repensando a didática**, p. 131–144, 1996.
- KIM, G. Relationship between index term specificity and relevance judgment. **Information processing & management**, Elsevier, v. 42, n. 5, p. 1218–1229, 2006.
- LINUX, T. I. P. **Source Code Definition**. [S.l.], 2005. Disponível em: <http://www.linfo.org/source_code.html>. Acesso em: 10.10.2020.
- LOUDEN, K. C.; LAMBERT, K. A. **Programming languages: principles and practices**. [S.l.]: Cengage Learning, 2011.
- LOWE, T.; BROPHY, S. An operationalized model for defining computational thinking. In: IEEE. **2017 IEEE Frontiers in Education Conference (FIE)**. [S.l.], 2017. p. 1–8.
- LUCIA, A. D.; PENTA, M. D.; OLIVETO, R.; PANICHELLA, A.; PANICHELLA, S. Using ir methods for labeling source code artifacts: Is it worthwhile? In: IEEE. **2012 20th IEEE International Conference on Program Comprehension (ICPC)**. [S.l.], 2012. p. 193–202.
- LUCKESI, C. C. Avaliação educacional escolar; para além do autoritarismo. **Revista da Ande**, São Paulo, n. 10, p. 47–51, 1986.
- MAAREK, Y. S.; BERRY, D. M.; KAISER, G. E. An information retrieval approach for automatically constructing software libraries. 1990.
- MACCABE, T. J. A complexity measure. **IEEE Transactions on software Engineering**, n. 4, p. 308–320, 1976.
- MIRZAYANOV, M. Codeforces. **Statistics**, 2015. Disponível em: <www.codeforces.com>. Acesso em: 10.10.2020.
- MOREIRA, M. P.; FAVERO, E. L. Um ambiente para ensino de programação com feedback automático de exercícios. **Workshop sobre Educação em Computação (WEI 2009)**, 2009.
- MORENO-LEÓN, J. e. a. Dr. scratch: a web tool to automatically evaluate scratch projects. **WiPSCE**, p. 132–133, 2015.
- OLIVEIRA, M. e. a. Análise de aprendizagem a partir de códigos-fontes e uma proposta de seleção automática de métricas de avaliação. **Anais dos Workshops do Congresso Brasileiro de Informática na Educação.**, 2018.
- PERRENOUD, P. Construir competências é virar as costas aos saberes. **Revista Pátio, Porto Alegre: ARTMED**, ano, v. 3, p. 15–19, 1999.
- RAABE, A. L. A.; BRACKMANN, C. P.; CAMPOS, F. R. Currículo de referência em tecnologia e computação: da educação infantil ao ensino fundamental. **Centro de Inovação para a Educação Básica-CIEB**, 2018.

- RAIGOZA, J. A study of students' progress through introductory computer science programming courses. In: IEEE. **2017 IEEE Frontiers in Education Conference (FIE)**. [S.l.], 2017. p. 1–7.
- ROBERTSON, S. Understanding inverse document frequency: on theoretical arguments for idf. **Journal of documentation**, Emerald Group Publishing Limited, 2004.
- ROBERTSON, S. E.; JONES, K. S. **Simple, proven approaches to text retrieval**. [S.l.], 1994.
- ROBINS, A. Learning edge momentum: A new account of outcomes in cs1. **Computer Science Education**, Taylor & Francis, v. 20, n. 1, p. 37–71, 2010.
- SALTON, G.; BUCKLEY, C. Term-weighting approaches in automatic text retrieval. **Information processing & management**, Elsevier, v. 24, n. 5, p. 513–523, 1988.
- SARACEVIC, T. Relevance: A review of and a framework for the thinking on the notion in information science. **Journal of the American Society for information science**, Wiley Online Library, v. 26, n. 6, p. 321–343, 1975.
- SARACEVIC, T. Relevance: A review of the literature and a framework for thinking on the notion in information science. part ii. **Advances in librarianship**, ACADEMIC PRESS INC., v. 30, p. 03, 2006.
- SARACEVIC, T. Relevance: A review of the literature and a framework for thinking on the notion in information science. part iii: Behavior and effects of relevance. **Journal of the American Society for information Science and Technology**, Wiley Online Library, v. 58, n. 13, p. 2126–2144, 2007.
- SBC, S. B. d. C. **Itinerário Formativo da Computação**. [S.l.], 2020. Disponível em: <<https://www.sbc.org.br/documentos-da-sbc/summary/203-educacao-basica/1216-itinerario-informativo-da-computacao>>. Acesso em: 10.10.2020.
- SEITER, L.; FOREMAN, B. Modeling the learning progressions of computational thinking of primary grade students. **Proceedings of the ninth annual international ACM conference on International computing education research**. ACM., p. 59–66, 2013.
- SOUZA, D.; FELIZARDO, k.; BARBOSA, E. A systematic literature review of assessment tools for programming assignments. In: IEEE. **2016 IEEE 29th international Conference on Software Engineering Education and Training (CSEET)**. [S.l.], 2016. p. 147–156.
- SU, X.; QIU, J.; WANG, T.; ZHAO, L. Optimization and improvements of a moodle-based online learning system for c programming. In: IEEE. **2016 IEEE Frontiers in Education Conference (FIE)**. [S.l.], 2016. p. 1–8.
- ULLAH, F.; WANG, J.; FARHAN, M.; JABBAR, S.; WU, Z.; KHALID, S. Plagiarism detection in students' programming assignments based on semantics: multimedia e-learning based smart assessment methodology. **Multimedia Tools and Applications**, Springer, p. 1–18, 2018.
- ULLAH, Z.; LAJIS, A.; JAMJOOM, M.; ALTALHI, A.; AL-GHAMDI, A.; SALEEM, F. The effect of automatic assessment on novice programming: Strengths and limitations of existing systems. **Computer Applications in Engineering Education**, Wiley Online Library, v. 26, n. 6, p. 2328–2341, 2018.

WASIK, S.; ANTCHAK, M.; BADURA, J.; LASKOWSKI, A.; STERNAL, T. A survey on online judge systems and their applications. **ACM Computing Surveys (CSUR)**, ACM New York, NY, USA, v. 51, n. 1, p. 1–34, 2018.

WING, J. M. Computational thinking. **Communications of the ACM**, ACM New York, NY, USA, v. 49, n. 3, p. 33–35, 2006.

WING, J. M. Computational thinking and thinking about computing. **Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences**, The Royal Society London, v. 366, n. 1881, p. 3717–3725, 2008.

YANG, H.; SUN, X.; LI, B.; DUAN, Y. Dr_psf: Enhancing developer recommendation by leveraging personalized source-code files. In: IEEE. **2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC)**. [S.l.], 2016. v. 1, p. 239–244.