PRÓ-REITORIA DE PESQUISA E PÓS-GRADUAÇÃO - PPG
CENTRO DE CIÊNCIAS TECNOLÓGICAS - CCT
MESTRADO PROFISSIONAL EM ENGENHARIA DE COMPUTAÇÃO E SISTEMAS - PECS

Plataforma de Apoio ao Ensino e Aprendizagem de Programação de Computadores

FRANCISCO ALAN DE OLIVEIRA SANTOS

São Luis - MA, 2020

FRANCISCO ALAN DE OLIVEIRA SANTOS

Plataforma de Apoio ao Ensino e Aprendizagem de Programação de Computadores

Dissertação apresentada ao curso MESTRADO PROFISSIONAL do Centro de Ciências Tecnológicas, da Universidade Estadual do Maranhão, como requisito para a obtenção do grau de Mestre em ENGENHARIA DE COMPUTAÇÃO E SISTEMAS

Orientador Prof. Dr. Luis Carlos Costa Fonseca

Junho de 2020

Santos, Francisco Alan de Oliveira.

Plataforma de apoio ao ensino e aprendizagem de programação de computadores / Francisco Alan de Oliveira Santos. – São Luís, 2020.

96 f

Dissertação (Mestrado) – Curso de Engenharia de Computação e Sistemas, Universidade Estadual do Maranhão, 2020.

Orientador: Prof. Dr. Luís Carlos Costa Fonseca.

1.Avaliação automática. 2.Ensino – Aprendizagem. 3.Programação.

Elaborado por Giselle Frazão Tavares - CRB 13/665

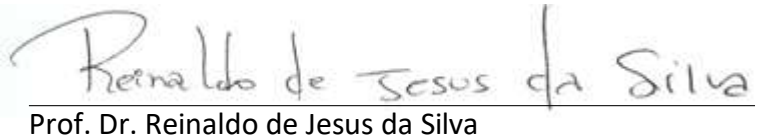
CCT - Centro de Ciências Tecnológicas
Universidade Estadual do Maranhão

Trabalho de Conclusão de Curso de MESTRADO PROFISSIONAL intitulado **Plataforma de Apoio ao Ensino e Aprendizagem de Programação de Computadores** de autoria de Francisco Alan de Oliveira Santos, aprovada pela banca examinadora constituída pelos seguintes professores:

Prof. Dr. Luis Carlos Costa Fonseca

Orientador

Prof. Dr. Cícero Costa Quarto
Membro interno



Prof. Dr. Reinaldo de Jesus da Silva
Membro interno

Prof. Dr. Othon de Carvalho Bastos Filho (UFMA)
Membro externo

São Luis, 6 de maio de 2020

RESUMO

O Ensino e aprendizagem de programação de computadores representa um dos mais complexos desafios para a educação em Computação. Um indicador que corrobora este fato são os altos índices de evasão dos cursos de Computação, principalmente nos períodos iniciais. Este trabalho apresenta uma proposta para apoiar o Ensino e aprendizagem de programação de computadores a partir da avaliação dos códigos produzidos pelos alunos. O diferencial desta abordagem apoia-se na combinação de análise estática, execução automática e testes, além do uso de métricas de código-fonte. Foi desenvolvido um sistema para avaliação e atribuição automática de notas com base na conformidade do código analisado com esses pilares. O uso o sistema demonstrou uma melhoria na capacidade de avaliar a aprendizagem.

Palavras-chave: <Avaliação automática>, <ensino-aprendizagem>, <programação> <Java>.

ABSTRACT

Teaching and learning computer programming represents one of the most complex challenges for computer education. An indicator that confirms this fact is the high dropout rates in computer courses, especially in the initial grades. This work presents a proposal to support Teaching and learning computer programming based on the evaluation of the codes used by students. The differential of this approach is to help in the combination of static analysis, automatic execution and tests, in addition to the use of source code metrics. A system was developed for the evaluation and automatic creation of notes based on the conformity of the analyzed code with these pillars. The use or system demonstrated an improvement in the ability to assess the learning.

Key-words: <Automatic assessment>, <teaching-learning>, <programming>, <java>.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

API Application Programming Interface (interface de programação de aplicação).

AVA Ambiente Virtual de Aprendizagem.

Brasscom Associação Brasileira das Empresas de Tecnologia da Informação e Comunicação.

CSS3 Cascading Style Sheets.

DDoS Distributed Denial Of Service.

DoS Denial Of Service.

EAPC Ensino e Aprendizagem de Programação de Computadores.

GC Google Classroom (Google Sala de Aula).

HTML5 HyperText Markup Language (Linguagem de Marcação de Hipertexto, versão 5).

HTTP Hypertext Transfer Protocol.

HTTPS Secure Hypertext Transfer Protocol.

ICPC ACM International Collegiate Programming Contest (Concurso de Programação da ACM).

IDE Integrated Development Environment (Ambiente Integrado de Desenvolvimento).

IE Informática na Educação.

IOI International Olympiads in Informatics (Olimpíadas Internacionais de Informática).

JDK Java Developer's Kit.

JO Juíz On-line.

JSON JavaScript Object Notation.

JVM Java Virtual Machine (Máquina Virtual Java).

LMS Learning Management System.

MOJO Módulo de Integração com os Juízes Online.

Moodle Modular Object-Oriented Dynamic Learning Environment (Ambiente de Aprendizado Modular Orientado a Objetos).

MSL Mapeamento Sistemático da Literatura.

POO Programação Orientada a Objetos.

RDBMS Relational Database Management System (Sistema Gerenciador de Banco de Dados).

REST Representational State Transfer (Transferência de Estado Representacional).

RSL Revisão Sistemática da Literatura.

SAA Sistema de Avaliação Automática.

SBC Sociedade Brasileira de Computação.

SQuaRE Software product Quality Requirements and Evaluation (Requisitos e Avaliação de Qualidade de Sistemas e Software).

SSL Secure Sockets Layer.

STI Sistema Tutor Inteligente.

TI Tecnologia da Informação.

TIC Tecnologia de Informação e Comunicação.

TSL Transport Layer Security.

UFAM Universidade Federal do Amazonas.

URI Uniform Resource Identifier.

WS Web Service.

SUMÁRIO

1 INTRODUÇÃO

As Tecnologias de Informação e Comunicação (TICs) se fazem presentes na sociedade moderna de tal forma que há uma enorme demanda mundial por profissionais habilitados a atuarem nesta área. De acordo com o Relatório Setorial de TIC 2018[1] da Associação Brasileira das Empresas de Tecnologia da Informação e Comunicação (Brasscom), o mercado brasileiro demandará 420 mil profissionais no setor de Software e Serviços entre 2018 e 2024, isso significa que serão necessários 70 mil profissionais ao ano até 2024. Atualmente, o Brasil oferta por ano 46 mil formandos em cursos com perfil tecnológico, valor notadamente insuficiente para suprir a necessidade de novos profissionais no mercado. Essa discrepância desperta para a necessidade de formação de mão de obra qualificada em um curto prazo (BRASSCOM, 2019).

Segundo Paes et al. (2013), Programação de Computadores é um dos principais fundamentos para profissionais de TIC, tanto nos níveis técnico, quanto superior e de pós-graduação. Disciplinas de programação são parte essencial dos currículos dos cursos da área de Computação (VENDRAME et al., 2019). Rocha et al. (2010) destacaram que os conceitos de programação têm uma grande importância na vida acadêmica e profissional dos estudantes da área.

Porém, apesar de sua importância, os números sobre a aprendizagem de programação são pouco promissores, pois disciplinas que abordam esses conteúdos costumam ter altas taxas de evasão e reprovação no mundo todo (VENDRAME et al., 2019). De acordo com Watson e Li (2014), a taxa de aprovação no Brasil é de 45%, sem perspectivas de melhoria futura. Segundo Watson e Li (2014 apud FONSECA et al., 2019), um terço dos estudantes não consegue obter o mínimo para a aprovação nas disciplinas de introdução à programação de computadores. Como consequência de tantas dificuldades e reprovações, muitas pesquisas apontam altos índices de desistência. Segundo Palmeira e Santos (2014 apud RABÊLO JÚNIOR et al., 2018), cursos de Ciência da Computação possuem uma taxa de ingressantes e concluintes de 14,3%. De acordo com Castro et al. (2002 apud ROCHA et al., 2010), há uma estreita ligação entre as altas taxas de evasão e as dificuldades de aprendizagem de programação.

Um fator agravante ao cenário de reprovação e evasão em Computação é o fato de que turmas de introdução à programação costumam ser numerosas (PEREIRA et al., 2019 apud FONSECA et al., 2019) (FRANÇA et al., 2011), com isso, atender individualmente os alunos se torna um grande desafio (FONSECA et al., 2019), exigindo tempo e empenho de professores e monitores que acabam não conseguindo realizar um acompanhamento

[1] https://brasscom.org.br/relatorio-setorial-de-tic-2019/

eficiente dos alunos (FRANÇA et al., 2011). Rocha et al. (2010) consideraram essa impossibilidade do professor de identificar as dificuldades individuais dos alunos como um fator limitante que torna o aprendizado passível de falhas. França et al. (2011) acrescentam que isto pode provocar desestímulo e dispersão em aulas de laboratório, dificultando o controle da turma e contribuindo negativamente para o aprendizado.

Embora a formação em massa de programadores seja uma necessidade, a viabilização deste processo se mostra difícil. No caso de cursos massivos, os desafios de trabalhos práticos são ampliados (SOUZA; BATISTA; BARBOSA, 2014; OLIVEIRA, 2015), pois a personalização do ensino se torna prejudicada, uma vez que se tem uma relação bem maior de alunos por professor (PAES et al., 2013). Diante do grande número de turmas, a replicação de aulas e quantidade de alunos, é inviável que o professor consiga fornecer um acompanhamento individual dos alunos sobre os erros que são cometidos (JESUS et al., 2018a),

Portanto, percebe-se um contraponto, pois, ao passo em que há uma carência de pessoas com conhecimentos sólidos em programação de computadores, também existe uma série de entraves para uma aprendizagem efetiva na formação de profissionais capazes de atender aos requisitos demandados pelo mercado. Diante deste cenário, Rabêlo Júnior et al. (2018) entendem que faz-se necessário buscar meios que tornem o ensino de algoritmos mais prático e atrativo.

Alinhado a essa necessidade emergente de capacitação de programadores aptos para a indústria, e na tentativa de contribuir para a redução do déficit de profissionais, este trabalho se propõe a fornecer uma alternativa para viabilizar a formação massificada de programadores, apresentando como inovação a sua diversidade de estratégias de avaliação, acompanhamento e controle da evolução do aprendizado individual para o sucesso da formação de novos profissionais de Tecnologia da Informação (TI).

1.1 DEFINIÇÃO DO PROBLEMA

O Ensino e Aprendizagem de Programação de Computadores (EAPC) representa um dos mais complexos desafios para a Educação na área de Computação (FILHO et al., 2007). Um indicador que corrobora este fato são os altos índices de evasão dos cursos de Computação, principalmente nos anos iniciais (GIRAFFA; MORAES; UDEN, 2014), período em que são introduzidas as primeiras noções de lógica algorítmica através das disciplinas básicas de programação. Uma possível explicação para as dificuldades presentes nessa fase da aprendizagem pode ser atribuída às habilidades prévias requeridas, pois, segundo Raposo, Maranhão e Soares Neto (2019), o processo de aprender algoritmos é composto de etapas bem definidas e interdependentes. Na visão de Paes et al. (2013),

programar se trata de uma atividade que exige criatividade e raciocínio lógico. Para Santos, Soares Segundo e Telvina (2019), o EAPC depende da absorção de diversos saberes relacionados e envolve a combinação de várias habilidades cognitivas, de forma que a avaliação desses conhecimentos demanda uma análise criteriosa (TOBAR et al., 2001).

Uma forma frequente de avaliar essas competências é por meio da revisão do código-fonte produzido pelos alunos. Nesse contexto, surgiram alternativas para a automatização e execução de testes de códigos-fonte (EBRAHIMI, 1994; BUYRUKOGLU; BATMAZ; LOCK, 2019), com o desenvolvimento de Sistemas de Avaliação Automática (SAAs). No geral, tais propostas incluem plataformas diversas para facilitar a correção de atividades avaliativas de programação, cuja adoção tem sido feita principalmente de forma integrada a Ambientes Virtuais de Aprendizagem (AVAs) (SANTOS; FONSECA, 2019).

A busca por métodos que facilitem a aprendizagem de conteúdos relacionados à prática de programação tem sido constante e alvo de muitos pesquisadores, que têm abordado o assunto de formas diversas. Dentre os trabalhos que objetivam apoiar o EAPC com o desenvolvimento de SAA, várias estratégias têm sido adotadas, como: Juízes On-line (JOs) (CHAVES et al., 2014), testes (SOUZA; MALDONADO; BARBOSA, 2011) análise automática de código-fonte, análise semi-automática (BUYRUKOGLU; BATMAZ; LOCK, 2019), análise sintática e semântica (CHA, 2007), verificação de convenções de codificação (CHEN; CHEN; LEE, 2018), além do uso de métricas de código-fonte (KOYYA; LEE; YANG, 2013; ELNAFFAR, 2016), até mesmo pesquisas interdisciplinares que envolvem neurociência (QUEIROZ; DANTAS, 2018) e psicologia (RAPKIEWICZ et al., 2006), entre outros.

Todavia, existem limitações nas abordagens existentes. Apesar da existência de diversas opções para facilitar o processo de ensino e aprendizagem nas disciplinas de programação, ainda há o que ser aprimorado (JESUS et al., 2019b). Um dos problemas apresentado por esses sistemas é a dificuldade que os professores encontram ao tratar as características individuais de cada aluno no decorrer do curso.

Oliveira (2015) relata que, no Brasil, a construção de sistemas de avaliação automática ainda é pouco expressiva. Segundo Oliveira et al. (2018), poucas soluções tecnológicas têm sido desenvolvidas para identificar dificuldades de aprendizagem e habilidades de programação a partir de informações de códigos-fontes escritos por alunos. De acordo com o autor, o desenvolvimento de estratégias tecnológicas nesse sentido tem sido um desafio para a Informática na Educação (IE). Para Oliveira (2015), a maioria das propostas ainda não venceu os desafios do domínio da programação introdutória.

Em sua pesquisa, Vendrame et al. (2019) constataram que nenhuma das ferramentas estudadas conseguiu ser amplamente adotada. Sobre esse fato, os autores levantam a hipótese de as ferramentas serem desenvolvidas por um pequeno grupo para solucionar um

problema local, não alcançando um público mais amplo. de forma que uma abordagem suficientemente aplicável a contextos diversos ainda está pendente.

Diante do exposto, percebe-se que a exploração de modalidades de avaliação representa uma oportunidade de pesquisa no campo da aprendizagem de programação, pois há questões em aberto acerca de encontrar formas de avaliação que sejam adaptáveis a diferentes contextos, posto que há uma dificuldade em superar as barreiras da heterogeneidade encontrada nesses ambientes e de encontrar uma solução para cobrir as peculiaridades de cada espaço de aprendizagem. Acredita-se que uma alternativa plausível seja munir o professor de ferramentas adaptáveis para que ele possa dispor de recursos que lhe permitam experimentar e definir seus próprios critérios avaliativos, guiado por uma perspectiva pedagógica, dentro de uma plataforma unificada para gerenciar a aprendizagem de seus alunos, seja um caminho para reduzir essas dificuldades.

1.2 PROPOSTA DE SOLUÇÃO

Este trabalho apresenta uma proposta para apoiar o EAPC a partir da avaliação dos códigos produzidos pelos alunos. Embora já existam ferramentas que possam ser utilizadas com esse fim, a abordagem aqui apresentada baseia-se na avaliação através da combinação de análise estática, execução automática e testes, verificação de convenções de codificação, além do uso de métricas de código-fonte. Para cumprir essa proposta, foi desenvolvido o CodeTeacher, um sistema web em Java para avaliação de atividades práticas de programação e atribuição automática de notas com base na conformidade do código analisado com critérios avaliativos pré-configurados, que dão composição a uma pontuação ao final do processo de avaliação. Em síntese, este trabalho objetiva mostrar a eficiência desta abordagem preliminar utilizada para auxiliar o professor a avaliar o aprendizado com o uso de um sistema flexível no que diz respeito à avaliação de práticas de programação, focando na automatização do processo de correção de trabalhos práticos dos alunos, utilizando critérios avaliativos definidos pelo professor, como tentativa de viabilizar a intensificação de atividades práticas de programação e proporcionar rápido *feedback* para os alunos e objetivando melhorar a produtividade do professor e dos alunos no processo de ensino-aprendizagem.

1.3 PREMISSAS E HIPÓTESE

No intuito de estabelecer um ponto de partida para este trabalho, formulou-se como questão central de pesquisa a seguinte pergunta: A combinação de diferentes elementos de avaliação automática tem o potencial de ajudar a construir uma análise holística para

encontrar uma visão mais ampla da aprendizagem em contextos de ensino de programação de computadores?

Para responder a essa questão, foram definidas as seguintes questões específicas de pesquisa (research questions):

RQ 1 Quais as maiores limitações dos principais SAAs?

RQ 2 Qual a eficácia da avaliação por análise automática de código-fonte?

RQ 3 Quais os benefícios da combinação proposta?

RQ 4 Quais as limitações do uso concomitante das estratégias adotadas?

Como resposta aos questionamentos especificados acima, elaborou-se a seguinte hipótese de pesquisa:

> O uso conjunto de análise estática, execução automática, verificação de convenções de codificação, e métricas de código-fonte pode prover evidências objetivas suficientes para embasar uma avaliação diagnóstica da aprendizagem de programação de computadores.

1.4 JUSTIFICATIVA

Um dos objetivos dos pesquisadores na área de educação é a melhoria dos métodos de avaliação da proficiência e habilidades dos estudantes (FERREIRA et al., 2019). De acordo com o mesmo autor, os métodos de avaliação educacional com maior destaque são aqueles que buscam apresentar dados precisos sobre a construção das competências através do uso de plataformas online que utilizam avaliações automáticas. Neste sentido, o desenvolvimento de estratégias tecnológicas de apoio à avaliação que identifiquem dificuldades de aprendizagem e habilidades de programação a partir de informações de códigos-fontes escritos por alunos, tem sido um desafio para a informática na educação, mas poucas soluções têm sido de fato desenvolvidas para esse fim (OLIVEIRA et al., 2018).

A prática é um dos pilares do aprendizado de programação (PAES et al., 2013), pois segundo Kurnia, Lim e Cheang (2002) “Programação é uma habilidade adquirida através da prática”. Nesse sentido, Santos, Segundo e Telvina (2017) citam os exercícios práticos como essenciais no aprendizado de uma linguagem de programação. Paes et al. (2013) defende que "a prática por meio da repetição de exercícios é um dos pilares para o aperfeiçoamento do conhecimento em programação de computadores". Ala-Mutka (2005) e Rahman, Ahmad e Nordin (2007) apontam corretude e funcionalidade como importantes itens de avaliação.

Além da prática, estudantes de programação precisam receber feedback consistente e personalizado para aprimorar suas habilidades de programação efetivamente (BUY-

RUKOGLU; BATMAZ; LOCK, 2019). Para Paes et al. (2013), o feedback imediato é importante por que os estudantes precisam receber um retorno sobre as suas tentativas de resolução de problemas o mais rápido possível, pois o não recebimento desse retorno em tempo hábil gera um acúmulo de dúvidas no estudante, chegando a prejudicar o entendimento do próximo assunto a ser assimilado pelo não entendimento do assunto anterior. Segundo Paes et al. (2013), o feedback instantâneo aumenta o interesse do aluno e melhora sua capacidade de resolver problemas programaticamente. Nesse sentido, a avaliação automática é considerada a melhor abordagem para produzir feedback consistente. No entanto, sistemas de avaliação automática geralmente fornecem feedback menos personalizado e detalhado (BUYRUKOGLU; BATMAZ; LOCK, 2019).

Meirelles (2013) argumenta que muitas das características de um bom software podem ser percebidas no código-fonte, e algumas são exclusivas dele. Nesse sentido, compreender convenções de código tem se tornado indispensável em Engenharia de Software, no entanto, muitos cursos universitários de programação falham em preparar seus estudantes nesse quesito (CHEN; CHEN; LEE, 2018). Para Insa e Silva (2015), os cursos de programação também devem garantir que os alunos aprendam convenções de codificação, como nomes de variáveis apropriados, uso correto de comentários, entre outros. Dessa forma, as melhores práticas de desenvolvimento de software podem ser disseminadas na formação dos programadores.

Petersen, Spacco e Vihavainen (2015) consideram o uso de métricas da Engenharia de Software para análise do código do estudante uma estratégia revolucionária para o ensino de programação. De acordo com a norma ISO/IEC 25000 (2014), métricas são medidas realizadas no desenvolvimento de software, normalmente utilizadas para estimar cronogramas de projetos, prever custos de desenvolvimento, controlar produtividade de processos e medir a qualidade do produto. Métricas também são úteis para detectar falhas de projeto, e controlar a qualidade de código-fonte (SOMMERVILLE, 2011). No âmbito educacional, métricas de software podem ser usadas para comparar características específicas do código-fonte de vários alunos (PEREIRA; OLIVEIRA; FERNANDES, 2017). Essas comparações, se compartilhadas, podem ser utilizadas para intermediar discussões voltadas à evolução de critérios de avaliação baseados em métricas de código-fonte (SANTOS; FONSECA, 2019).

Meirelles (2013) defende que um programa pode ser compilado e testado e muitas informações sobre seu comportamento podem ser analisadas, mas características importantes como organização e legibilidade são perdidas. Nesse contexto, as métricas de código-fonte podem complementar as demais abordagens de monitoramento da qualidade de códigos escritos por aprendizes. A partir de uma coleção de métricas coletadas automaticamente e de uma forma objetiva de interpretar seus valores, podem-se identificar características específicas nos códigos dos alunos, estes ao serem reportados de suas implementações

problemáticas, podem ser orientados reescrevê-las de forma melhorada.

Portanto, a estratégia ora defendida investe na multiplicidade de possibilidades de avaliação de código-fonte, utilizando as melhores abordagens encontradas na literatura de forma complementar, em uma iniciativa guiada por uma perspectiva pedagógica, baseada em teorias construtivistas de aprendizagem para alcançar uma avaliação mais abrangente das competências de programação, que contemple as funções diagnóstica, formativa e somativa.

1.5 OBJETIVOS

1.5.1 Objetivo geral

O foco principal desta pesquisa concentra-se em alcançar potencialidades de coleta automática de dados relativos ao desempenho discente em programação e extrair informações suficientes para compor uma nota final, para subsidiar uma avaliação automática da aprendizagem de programação de computadores.

1.5.2 Objetivos específicos

• Fazer um levantamento do estado da arte acerca dos sistemas, ferramentas e técnicas empregados na análise de código-fonte para suportar o EAPC;

• Propor um instrumento de apoio à avaliação diagnóstica, formativa e somativa da aprendizagem de alunos na prática da programação de computadores;

• Realizar análises utilizando as informações extraídas e usar ferramentas de visualização de informação para comparar soluções de programação em detalhes.

• Produzir relatórios que ajudem a detectar deficiências de aprendizagem para uma abordagem personalizada, oportunizando maior clareza sobre o rendimento discente individual e coletivo.

1.6 ESTRUTURA DO TRABALHO

O restante deste documento está estruturado como segue: O Capítulo 2 detalha o fundamento teórico do trabalho proposto, discutindo as bases teóricas utilizadas para realização desta pesquisa.

O Capítulo 3 expõe e discute alguns trabalhos relacionados e apresenta várias ferramentas correlatas atuais, mostrando seus usos e limitações.

O Capítulo 4 descreve a arquitetura do CodeTeacher e como ela foi projetada para incorporar as propriedades de um serviço facilitador do EAPC. São apresentados conceitos sobre o modelo arquitetura e a modularidade do sistema. Também descreve o processo de análise de código-fonte adotado neste trabalho.

O Capítulo 5 traz um experimento empreendido para averiguar a viabilidade da abordagem proposta, uma configuração de valores de referência é definida com base em análises estatísticas envolvendo diversos projetos de alunos de uma instituição de ensino. Essa configuração é utilizada para avaliar o desempenho dos alunos, ilustrando o uso do CodeTeacher como ferramenta de suporte à interpretação do desempenho discente.

E por fim, o Capítulo 6 apresenta a conclusão com as considerações finais e trabalhos futuros.

2 CONCEITOS GERAIS E REVISÃO DA LITERATURA

Neste capítulo será apresentado o estado da arte/referencial teórico sobre os temas a que se refere este estudo. Serão discutidos os conceitos básicos relevantes para a compreensão do trabalho, tais como ensino de programação, avaliação automática de código-fonte, padrões de codificação, qualidade de software e métricas de código-fonte. Este capítulo está relacionado com a **RQ 2**. Os conceitos utilizados por esta pesquisa foram obtidos por meio de uma revisão da literatura e estão agrupados em quatro seções, A Seção 2.1 apresenta as abordagens para o EAPC e trata das características das disciplinas de programação. A Seção 2.2 discute os tipos de critérios e métodos de avaliação de códigos-fonte. A Seção 2.3 discorre sobre as definições e aplicabilidade das convenções de codificação. Na Seção 2.4 será apresentada a definição das métricas de código-fonte mais recorrentes na literatura.

2.1 ENSINO-APRENDIZAGEM DE PROGRAMAÇÃO DE COMPUTADORES

O EAPC tem sido foco de muitos trabalhos acadêmicos, especialmente nos últimos anos (ROCHA et al., 2010; VIANA; PORTELA, 2019). Dentre as principais motivações dos estudos recentes, observa-se uma ênfase significativa nas dificuldades de aprendizagem dos alunos, pois, segundo Aureliano e Tedesco (2012), Jesus et al. (2018a), várias são as dificuldades enfrentadas durante o processo de ensino-aprendizagem, principalmente no caso de alunos iniciantes. Essas dificuldades levam a uma alta taxa de reprovação em disciplinas introdutórias de programação (PEREIRA et al., 2019 apud FONSECA et al., 2019).

Weber e Steinle (1996 apud JESUS et al., 2019a) caracterizaram o ensino de programação de computadores como uma tarefa complexa e que demanda esforço não somente do professor, mas também depende de certa habilidade do aluno. Essa complexidade advém dos diversos problemas enfrentados pelos alunos e professores (MARCOLINO; BARBOSA; LOPES et al., 2015, 2016 apud OLIVEIRA et al., 2019).

Para Weinberg (1971 apud CASTRO; FUKS, 2011) a aprendizagem de programação é uma atividade cognitiva que requer alto nível de raciocínio abstrato. Raposo, Maranhão e Soares Neto (2019) afirmaram que "o processo de aprender algoritmos é modular, com etapas bem definidas, que são pré-requisitos umas das outras". Sobre essa dependência,

Rocha et al. (2010) ressaltaram que a retenção dos conhecimentos iniciais em programação impacta diretamente no desempenho das disciplinas mais avançadas no decorrer de todo o curso.

Um pré-requisito importante no aprendizado de linguagens de programação é o conhecimento prévio sobre a resolução de problemas dentro do contexto anterior à computação (LEMOS; BARROS; LOPES, 2003 apud JESUS et al., 2018a). Nesse sentido, Jesus et al. (2018a) afirmam que geralmente os alunos têm muita dificuldade em aplicar suas habilidades prévias. Para estudantes em disciplinas introdutórias, o nível de complexidade dessas habilidades é alto (PEREIRA, 2018 apud FONSECA et al., 2019). Raposo, Maranhão e Soares Neto (2019) declararam que o aprendizado de algoritmos requer o conhecimento de fundamentos de assimilação complexa, como abstração, decomposição e identificação de padrões, cuja aquisição exige muita prática.

Raabe e Silva (2005 apud ROCHA et al., 2010), declaram que há várias possibilidades quando se investigam as origens das dificuldades em aprender a programar. Segundo Almeida et al. (2020) a atividade de programação é permeada por conceitos abstratos que atrapalham a elaboração de um raciocínio, pois geralmente o aluno tem que se preocupar com muitas questões fora do domínio do problema, já que o ato de programar normalmente envolve conhecimento de caraterísticas próprias do ambiente de desenvolvimento, especificidades da máquina, desviando a atenção do aluno e limitando sua capacidade de se abster de detalhes subjacentes ao problema central. Almeida et al. (2020) levantaram a hipótese de que o fato de os conceitos de programação tradicionalmente serem introduzidos por meio de atividades em papel, logo, sem a possibilidade de poder testá-lo no computador e dispor de recursos como feedback e depuração, reduz a motivação do aluno.

2.2 ABORDAGENS DE ANÁLISE DE CÓDIGO-FONTE

Em geral, os erros cometidos por alunos iniciantes são os mesmos, devido ao fato de que em disciplinas introdutórias, os conceitos básicos apresentados são comuns na grande maioria das turmas (JESUS et al., 2018a). Nesta seção são contextualizadas as abordagens de avaliação das habilidades dos estudantes em plataformas virtuais de aprendizagem.

A auditoria de código é uma prática proveniente da Engenharia de Software que tem sido utilizada para detectar inconformidades em códigos escritos por aprendizes de programação. As soluções de programação podem ser avaliadas usando três tipos de abordagem: avaliação manual, automatizada e semi-automatizada (ALA-MUTKA, 2005). Por sua vez, os sistemas de avaliação automática e semi-automática usam o método estático, dinâmico, ou estático-dinâmico.

2.2.1 Avaliação Manual

Este é o método mais antigo para detectar vulnerabilidades e problemas de programação. Esse método é considerado por muitos autores como a forma mais eficiente de gerar feedback personalizado. No entanto, esse é um processo demorado e susceptível a falhas humanas. Embora na avaliação manual possam ser usadas ferramentas de computador para exibir código, essas ferramentas não possuem recursos para auxiliar no processo de avaliação propriamente dito, contando apenas com as habilidades do avaliador (BUYRUKOGLU; BATMAZ; LOCK, 2019). Além de representar uma alta carga de trabalho para o avaliador humano, uma das grandes desvantagens da avaliação manual é que ela está sujeita a gerar feedbacks inconsistentes, principalmente quando os códigos são extensos (CARTER et al., 2003).

2.2.2 Avaliação Semi-automática

Essa abordagem requer que o professor faça parte do processo de avaliação para monitorá-lo manualmente (SAIKKONEN; MALMI; KORHONEN, 2001). Na avaliação semi-automática, a análise dinâmica é inicialmente realizada, seguida de uma análise estática executada por um humano, cujo resultado gera um feedback (INSA; SILVA, 2015), em outras palavras, a avaliação é realizada automaticamente por uma ferramenta, mas é exigida a inspeção manual do código de programação pelo instrutor (ALA-MUTKA, 2005 apud GUPTA; GUPTA, 2018).

Jackson (2000 apud BUYRUKOGLU; BATMAZ; LOCK, 2016) declararam que a fusão das avaliações automática e manual é uma solução benéfica, porque as tarefas de programação dos alunos podem ser avaliadas não apenas em pouco tempo, mas também com feedback mais detalhado do que com base na avaliação feita apenas por computador.

Em um exemplo descrito por Jackson (2000 apud SOUZA; FELIZARDO; BARBOSA, 2016), a ferramenta verifica automaticamente se os programas dos alunos produzem a saída correta para alguns dados de entrada. Quando um dos programas produz uma saída diferente da esperada, a ferramenta solicita ao instrutor que confira manualmente as diferenças.

2.2.3 Avaliação Automática

O conceito de avaliação automática de programação foi introduzido na década de 60 por Hollingsworth (OLIVEIRA, 2015). Segundo Schneider e Jaques (2016), atualmente, vários AVAs e Sistemas Tutores Inteligentes (STIs) da área de ensino de programação suportam a avaliação automática das soluções dos exercícios. Essa validação geralmente

ocorre por meio da aplicação de casos de teste no sistema alvo, validando as pós-condições e os resultados obtidos.

Caso de teste é o nome dado a um conjunto de instruções elaboradas para testar um software. O caso de teste é composto basicamente por dados de entrada, pré-condições de execução e ações a serem executadas no sistema alvo, resultados esperados e pós-condições de execução. Para que um sistema seja considerado correto, ele deve sempre satisfazer as pós-condições e trazer os resultados esperados dado o conjunto de entradas e pré-condições. Além de avaliar a solução por meio da avaliação dinâmica e informar o resultado ao aluno, alguns AVAs e STIs de ensino de programação também se utilizam de técnicas de análise estática para aprimorar o feedback apresentado para o aluno.

Atualmente utilizam-se três abordagens de avaliação automática de programação: análise dinâmica, a análise estática e a análise dinâmico-estática (OLIVEIRA, 2015).

2.2.4 Análise Estática

Dá-se o nome Análise Estática (ALA-MUTKA, 2005) a um conjunto técnicas, como controle de fluxo e procura de padrões, desenvolvidas ao longo dos anos para detectar problemas em programas, sendo uma estratégia de avaliação automática baseada na análise de códigos-fontes (OLIVEIRA, 2015). Esta abordagem envolve a coleta de informações sobre o código, sem que este precise ser executado (JESUS et al., 2019b), pois são verificados apenas os elementos estruturais do código (OLIVEIRA, 2015). Por ser baseada na escrita do código-fonte, esse grupo de técnicas possui aplicação na avaliação de programas escritos por aprendizes para extrair dados e permitir uma visão analítica da aprendizagem. A seguir são relatados alguns trabalhos que abordam o emprego na analise estática em contextos de aprendizagem de programação.

Jesus et al. (2019b) afirmam que esse tipo de procedimento analisa a estrutura da solução com o objetivo de verificar a existência de más práticas de programação. Os autores elencam as diversas checagens que podem ser realizadas por meio da abordagem estática, são elas: verificação de erros sintáticos na solução do aluno, aferição do grau de similaridade entre dois ou mais algoritmos e averiguar se a solução do aluno apresenta um conjunto pre-definido de palavras-chaves.

Oliveira et al. (2018) declararam que o emprego da análise estática permite analisar esforço, complexidade, eficiência e qualidade de programação, tornando possível avaliar itens como erros de programação de ordem sintática, semântica e estrutural bem como o estilo de programação. Oliveira (2015) destacam que, nessa abordagem, aplicam-se várias técnicas para analisar estilo de programação, métricas de software, similaridade estrutural e não-estrutural bem como para detectar palavras-chave, erros sintáticos e semânticos e

até plágios (RAHMAN; AHMAD; NORDIN, 2007).

As possibilidades citadas por Schneider e Jaques (2016) são o desrespeito aos padrões de nomenclatura de classes e variáveis, os métodos muito complexos, as variáveis não usadas, entre outros problemas que possam comprometer a qualidade do código. Schneider e Jaques (2016) acrescentam que a análise estática também pode ser usada para identificar erros comuns de programação, que tem o potencial de causar comportamentos inesperados durante a execução do programa, como por exemplo, conversões de dados para tipos não compatíveis, loops infinitos, comparação incorreta de valores, entre outros.

Oliveira (2015) listaram algumas vantagens e desvantagens da analise estática. Dentre as principais vantagens estão o menor custo, a menor dependência de oráculos (modelos de soluções ou gabaritos) e a possibilidade de oferecer uma avaliação mais próxima da avaliação humana. O baixo custo é justificado pelo fato de que, como a análise estática do código não requer sua execução, o consumo de recursos computacionais é reduzido, pois não há gasto de tempo de processamento e esforço de execução em máquinas servidoras. A principal desvantagem reside no fato de a abordagem estática não permitir a análise dos resultados da execução para avaliar o desempenho do programa, deixando de contemplar requisitos considerados importantes para a avaliação da aprendizagem como a corretude, funcionalidade e eficiência (ALA-MUTKA, 2005; RAHMAN; AHMAD; NORDIN, 2007). A avaliação de análise sintática é mais justa porque contempla o processo de construção de programas.

Apesar dos exemplos de uso relatados, Schneider e Jaques (2016) afirmam que na avaliação automática da aprendizagem de programação, esse tipo de análise é menos comum em AVAs ou STIs e geralmente é feito com a utilização de ferramentas que automatizam o processo e retornam os problemas encontrados.

2.2.5 Análise Dinâmica

Esse modelo de auditoria analisa o comportamento da aplicação em execução por um conjunto de dados de testes, ou seja, o código é executado e então verifica-se se o programa retorna as saídas desejadas para as entradas fornecidas (ROMLI; SULAIMAN; ZAMLI, 2010). Seu principal objetivo é descobrir erros de execução em um programa. De acordo com Schneider e Jaques (2016) esta é a modalidade de avaliação predominante na maioria dos AVAs e STIs.

Na avaliação do aprendizado em programação, esse tipo de abordagem tem o propósito de garantir que o programa do aluno seja capaz de cumprir os objetivos propostos pelo professor no exercício. Para isso, é necessário que o código do aluno seja compilado, empacotado e executado, gerando os resultados corretos e atendendo as pós-condições

estabelecidas (SCHNEIDER; JAQUES, 2016).

Oliveira (2015) acrescentam que, na abordagem dinâmica, o funcionamento do programa é avaliado através de testes de caixa-preta e de caixa-branca. A testagem da caixa-preta, também chamada de teste funcional, é orientada a dados ou à entrada e saída e avalia o comportamento externo do programa, isto é, se a saída está correta de acordo com as entradas fornecidas. Já o teste da caixa-branca avalia o comportamento interno avaliando se os componentes de um programa funcionam.

A grande vantagem desse tipo de análise é a possibilidade de explorar outras vulnerabilidades que não poderiam ser encontradas apenas no código. No entanto, esse modelo de avaliação possui muitas desvantagens, apesar de seu amplo uso. Oliveira (2015) citam a dificuldade em tratar a variabilidade de soluções e fornecimento de feedback pouco detalhado (NAUDÉ; GREYLING; VOGTS, 2009). A primeira acontece quando há liberdade de escrita de código. A segunda é devida ao fato de a análise se basear apenas nos resultados da execução, isto é nas saídas fornecidas pelo programa, e não considerar a capacidade de programar de um aluno não contempla os processos realizados na construção de um programa, não se pode afirmar, com base apenas no resultado correto, se um aluno é bom programador ou não.

Naudé, Greyling e Vogts (2009) acrescentam que a análise dinâmica é sensível a pequenos erros, pois ainda que um aluno tenha realizado corretamente todos os passos para resolver um problema, um simples erro pode provocar falhas em todos os testes dinâmicos e prejudicar a avaliação do aluno, resultando em uma avaliação não fidedigna da aprendizagem. Nesse caso, a maior parte do processo de programação desenvolvido corretamente pelo aluno é anulada injustamente na avaliação dinâmica. No entanto, Oliveira (2015) destaca que a maior deficiência da análise dinâmica é sua a forte dependência de um modelo de solução padrão. Segundo os autores, o fato de um mesmo problema poder ser resolvido de formas diversas inviabiliza o uso da análise dinâmica.

2.2.6 Análise Dinâmico-estática

A análise dinâmico-estática ou estático-dinâmica é uma combinação das duas abordagens anteriores. De acordo com Romli, Sulaiman e Zamli (2010), esta é considerada a abordagem mais adequada para avaliar objetivos educacionais no domínio da programação. Segundo (OLIVEIRA, 2015) o principal benefício da análise dinâmico-estática é a redução das desvantagens de se aplicar apenas uma das duas estratégias. Por outro lado, a análise dinâmico-estática agrega as desvantagens das duas abordagens, ou seja, a forte dependência dos modelos de soluções e dos casos de teste, características herdadas da análise estática e dinâmica, respectivamente. Essas desvantagens são potencializadas em casos em que há ocorrências de uma ou mais das seguintes situações: alta variabilidade de soluções,

imprevisibilidade das saídas e demanda de muitos casos de testes.

2.3 PADRÕES DE CODIFICAÇÃO

A organização do código-fonte facilita os processos de desenvolvimento para que se possa escrever e manter código com qualidade (MARTIN; COPLIEN, 2008). Nesse contexto, um padrão de codificação é composto por um conjunto de regras (VERMEULEN et al., 2000) que estabelecem como o código deve ser escrito, descrevendo várias convenções usadas para implementações inteligíveis, consistentes e com qualidade. Embora algumas normas possam transcender qualquer linguagem de programação, os padrões de codificação são, em sua maioria, específicos para as linguagens a que se referem, como, por exemplo, os padrões criados pela Sun Microsystems / Oracle[1] (REDDY, 2000) e pela Google[2] (GOOGLE, 2019) para Java. Esses padrões definem as regras recomendadas da linguagem, não sendo, portanto, obrigatórios. Contudo, o uso de padrões de codificação é uma prática extensamente aceita no desenvolvimento de software (LAYTON; OSTERMILLER, 2019), sendo que uma empresa pode considerá-los como sugestões para criar seus próprios padrões, adotar apenas partes e/ou adaptar padrões existentes.

A necessidade da adoção de boas práticas, entre outras razões, reside no fato de que dificilmente um software é mantido durante a toda a sua vida pelo autor ou autores originais (MARTIN, 2002). Dessa forma, o uso de padrões de codificação contribui para reduzir a curva de aprendizado na transição dos integrantes de uma equipe, facilitando o entendimento do código do sistema por qualquer pessoa que conheça os padrões adotados e os siga, e permitindo que o código seja compartilhado de forma que qualquer membro da equipe possa entender rapidamente o código escrito pelos outros (MARTIN; COPLIEN, 2008).

Padrões de codificação também ajudam a aumentar a produtividade em um projeto, uma vez que a comunicação dentro da equipe de desenvolvimento tende a ser facilitada, pois entende-se que equipes de programadores que tenham uma forma padrão para nomear e formatar o código passem a compreendê-lo rapidamente, contribuindo para a criação de um ambiente colaborativo. Os benefícios dessas práticas são maior agilidade na retirada de bugs, atividades de validação e quanto ao uso da manutenabilidade[3] nos códigos-fonte (WILDT, 2015).

O escopo dos padrões de codificação é bastante amplo e pode variar de acordo com os objetivos, o contexto e o domínio da linguagem. Geralmente, cobrem as seguintes áreas (CHEN; CHEN; LEE, 2018):

[1] https://www.oracle.com/technetwork/java/codeconvtoc-136057.html
[2] http://google.github.io/styleguide/javaguide.html
[3] Capacidade/facilidade de prestar manutenção de software (PRESSMAN, 2009)

- **Nomenclatura**. Inclui convenções para nomear elementos como variáveis, constantes e funções, entre outros.
- **Organização de arquivos**. Inclui a convenção de nomenclatura de arquivos e como os arquivos serão organizados no sistema de arquivos.
- **Padrões para comentários**. Trata de estabelecer uma abordagem coerente para os comentários no intuito de melhorar o entendimento e até mesmo suportar a capacidade de gerar documentação a partir do código.
- **Convenções de codificação**. Aplicação coerente de convenções específicas em nível de código e a exclusão de formatações consideradas inadequadas para a qualidade do código.
- **Indentação**[4]. Preocupa-se com o uso consistente de espaços e linhas em branco, para a melhorar o alinhamento do código.
- **Quebras de linha**. Define tamanhos para o comprimento das linhas, estabelecendo um limite máximo aceitável para não prejudicar a legibilidade. Também elenca formas aceitáveis de como quebrar uma expressão quando esta não couber em uma única linha.

2.4 QUALIDADE DE SOFTWARE E MÉTRICAS DE CÓDIGO-FONTE

Qualidade é um conceito que se refere a atributos e características positivas que dizem respeito ao grau de perfeição e de conformidade a certo padrão. No entanto, a percepção da qualidade é relativa, pois depende de como ela é observada por um indivíduo, que usa como base diversos indicadores para medi-la, como por exemplo, quesitos como durabilidade e utilidade. Apesar da subjetividade deste conceito, em se tratando de software, existem vários atributos e indicadores aceitos universalmente como sendo características de um bom software e de um código fonte bem escrito.

A série de normas ISO/IEC 25000 (2014), também conhecida como Requisitos e Avaliação de Qualidade de Sistemas e Software (SQuaRE, Software product Quality Requirements and Evaluation), apresenta uma estrutura para a avaliação da qualidade de produtos de software, de forma a configurar-se como uma das normas mais importantes a esse respeito. Elas realizam uma junção e evolução das regras vigentes nas ISO/IEC 9126-1 (2001) e ISO/IEC 14598-1 (2001), descontinuadas após a publicação desta série. Dentro deste contexto, a ISO/IEC 25021 (2012), parte integrante da ISO/IEC 25000, tem o objetivo

[4] Ato de separar os níveis de codificação por espaços em branco

de auxiliar e padronizar o processo de caracterização e avaliação de qualidade de produtos de software, apresentando características e um guia para o uso dessas características para medição de qualidade de produtos de software.

Harman (2010) considera como código-fonte "qualquer descrição totalmente executável de um sistema de software". Portanto, nessa definição, código-fonte é concebido de modo a incluir código de máquina, linguagens de alto nível e representações gráficas executáveis de sistemas. Neste trabalho, o conceito de código-fonte será restrito a algoritmos escritos em uma linguagem de programação de alto nível.

Conceitualmente falando, qualidade de software e de código são duas disciplinas bem diferentes, cada uma delas está relacionada a diferentes aspectos e indicadores qualitativos do produto de software. A qualidade de software está em um nível mais elevado, faz parte da visão dos usuários e gestores, é o que eles vêm e obtêm a partir da utilização de um software. Alguns indicadores utilizados para medir a qualidade de um software incluem usabilidade e confiabilidade. Já a qualidade de código está em um nível mais baixo, faz parte da visão dos desenvolvedores, engenheiros, arquitetos e, em alguns casos, analistas e gerentes. Os indicadores da qualidade de código incluem, por exemplo, complexidade do código, duplicações de código, tamanho do código, entre outros.

Medir a qualidade de um código, no entanto, não é uma tarefa trivial, é uma atividade complexa e que envolve muitos indicadores (FENTON; PFLEEGER, 1998). Métricas de software têm como objetivo identificar, medir e consequentemente controlar os principais parâmetros que afetam o desenvolvimento de software. A análise das métricas de qualidade é necessária para assegurar que o código se mantenha aderente à sua arquitetura em toda a fase de desenvolvimento, reduzindo o esforço e o custo em alterações e manutenções, além de ser um fator motivador, ao permitir que o desenvolvedor esteja inserido em um ambiente que presa pelo alto padrão técnico.

De acordo com a (ISO/IEC 25021, 2012), métricas de código-fonte são uma subcategoria das métricas de software e são medidas obtidas através da análise do código-fonte, logo, enquadram-se na categoria de métricas internas. As métricas de código-fonte subdividem-se em dois grandes grupos: métricas de tamanho e complexidade, que medem características aplicáveis a qualquer programa, e métricas orientadas a objeto, que medem propriedades específicas de sistemas desenvolvidos sob esse paradigma.

Existe uma grande quantidade de métricas deste tipo. Para uma melhor visão e entendimento, os tipos e métricas utilizados neste trabalho são brevemente apresentados no Capítulo 5 desta dissertação.

3 TRABALHOS RELACIONADOS

Este capítulo reúne as abordagens, métodos, técnicas e ferramentas encontradas na literatura recente sobre estratégias tecnológicas de avaliações da aprendizagem baseadas na análise de códigos-fontes. Os conteúdos apresentados foram construídos com base nas Revisões Sistemáticas da Literatura (RSLs) realizadas por Ihantola et al. (2010), Ramos et al. (2015) e Souza, Felizardo e Barbosa (2016) nos estudos de Striewe e Goedicke (2014) e Oliveira (2015), na revisão bibliográfica conduzida por Oliveira, Mota e Oliveira (2015), nos Mapeamentos Sistemáticos da Literatura (MSLs) de Silva et al. (2015), Barbosa et al. (2015), Marcolino e Barbosa (2015), Oliveira et al. (2018) e Vendrame et al. (2019).

O conteúdo deste capítulo está alinhado com os objetivos específicos apresentados no Capítulo 1 e foi construído com vistas a contemplar as questões de pesquisa **RQ 1** e **RQ 2**. Na Seção 3.1, são destacadas as principais iniciativas no desenvolvimento de avaliadores automáticos. Na Seção 3.2, são apresentados os principais desafios da avaliação automática de programação. Na Seção 3.3 são reunidos requisitos para perspectivas futuras para o desenvolvimento de AVAs com suporte a correção automática de programação. Além disso, nessa seção é feita uma comparação entre outras abordagens de avaliação que são apoiadas por analisadores de código.

3.1 SISTEMAS E FERRAMENTAS

Esta seção aborda alguns sistemas e ferramentas tecnológicas existentes que são utilizadas no apoio ao ensino das disciplinas de programação, abordando suas principais características, pontos positivos e negativos. As ferramentas foram selecionadas com base em Silva et al. (2015), Ramos et al. (2015), Oliveira (2015), Ihantola et al. (2010), Oliveira, Mota e Oliveira (2015), Oliveira et al. (2018) e Vendrame et al. (2019). Uma vez que o foco do trabalho é analisar ferramentas de trabalhos mais recentes, foram excluídos desta avaliação softwares cujo ano da última publicação seja anterior a 2010.

3.1.1 Juízes On-line

As plataformas de JOs (KURNIA; LIM; CHEANG, 2002) são ferramentas empregadas em competições de programação que compilam, executam e testam códigos-fonte submetidos a elas, realizando a avaliação automática destes códigos e informando se o programa funcionou corretamente ou não (CHAVES et al., 2013; ALVES; JAQUES, 2014). Geralmente, essa avaliação gera respostas como: certo, saída errada, saída mal formatada,

erro de compilação, erro em tempo de execução, entre outros (CAMPOS; FERREIRA, 2004 apud SANTOS; RIBEIRO, 2011b).

Normalmente, essas plataformas fornecem um repositório de exercícios, onde o aluno pode resolver problemas em uma linguagem de programação suportada pela ferramenta (DAGOSTINI et al., 2018). Devido à sua praticidade, os JOs se tornaram não apenas exclusividade dos concursos e maratonas de programação e passaram a ser usados também no ensino de programação, durante aulas em laboratório, mostrando-se como uma alternativa para acompanhar turmas numerosas, uma vez que o uso destas ferramentas ajuda os professores a acompanharem individualmente seus alunos, e estes, por sua vez, podem obter retorno em tempo real sobre as correções de suas soluções implementadas.

Seguindo essa tendência, iniciativas foram realizadas no sentido de integrar JOs a AVAs como recurso de apoio a disciplinas de programação, a fim de automatizar o processo de submissão e avaliação de trabalhos práticos (SOUZA; BATISTA; BARBOSA, 2014). Nos tópicos subsequentes, serão abordadas as ferramentas mais recorrentes na literatura na última década.

3.1.1.1 ProgTest

O ProgTest (SOUZA; MALDONADO; BARBOSA, 2011) é um sistema web de apoio automatizado à avaliação de submissões de programas escritos em Java, desenvolvido para auxiliar as disciplinas de fundamentos de programação e teste de software. O ProgTest tem sido utilizado em cenários reais de ensino, em disciplinas oferecidas no ICMC/USP (SOUZA; BATISTA; BARBOSA, 2014). O ambiente pode fornecer um feedback apropriado para avaliar o desempenho dos alunos em relação a programação e teste (BARBOSA et al., 2008).

Para a avaliação de práticas de programação, juntamente com os programas também são submetidos seus respectivos casos de testes. O ProgTest compila os programas do aluno e submete-os à execução dos testes. Para isso, ferramentas de teste estão integradas ao ProgTest, fornecendo apoio à aplicação de critérios de teste (SOUZA; MALDONADO; BARBOSA, 2012) e tornando-o capaz de avaliar tanto a qualidade dos programas escritos pelos alunos quanto a qualidade dos casos de teste projetados para testar esses programas (SOUZA; ISOTANI; BARBOSA, 2015).

O processo do ProgTest consiste em compilar os programas do professor e do aluno, executar os critérios de testes funcional e estrutural em ambos e integrar técnicas de testagem a conceitos de programação. Segundo Oliveira (2015), o fato de o professor ter que fornecer os casos de teste juntamente com os modelos de soluções gera um aumento da carga de trabalho do professor, esse problema é ampliado pela variabilidade de soluções dos alunos.

3.1.1.2 MOJO

O Módulo de Integração com os Juízes Online (MOJO) (CHAVES et al., 2013) é um sistema de avaliação de práticas de programação que integra o conceito de JOs (KURNIA; LIM; CHEANG, 2002) ao Ambiente de Aprendizado Modular Orientado a Objetos (Moodle, Modular Object-Oriented Dynamic Learning Environment)[1] (MOODLE, 2020) para prover um ambiente para correção de atividades de programação. Ele consiste em um módulo de integração responsável pela comunicação e interação entre o Moodle e os Juízes Online, dessa forma, o Moodle é encarregado de fornecer a interface e o conjunto de funcionalidades necessárias à gestão e ao acompanhamento das atividades de programação, e a comunicação e interação com os Juízes Online (CHAVES et al., 2013) ficam a cargo do MOJO.

O MOJO visa auxiliar o professor no processo de elaboração, submissão e avaliação de questões de programação. Portanto, toda a experiência de uso do MOJO é focada na perspectiva do professor, perfil de usuário alvo do MOJO. Seu principal objetivo é diminuir consideravelmente a sobrecarga de trabalho do docente na correção e apresentação dos resultados das atividades desenvolvidas pelos alunos. Por ser centrado no professor, os resultados do JO são apresentados ao docente e este o repassa para os seus alunos, como forma de dar um feedback para os aprendizes (ALVES; JAQUES, 2014).

3.1.1.3 BOCA

O BOCA On-line Contest Administrator[2] (CAMPOS; FERREIRA, 2004; FRANÇA et al., 2011) é um sistema web para submissão de exercícios e correção on-line de código. É a plataforma atual usada nas competições de programação promovidas pela Sociedade Brasileira de Computação (SBC), como o Concurso de Programação da ACM (ICPC, ACM International Collegiate Programming Contest)[3] e as Olimpíadas Internacionais de Informática (IOI, International Olympiads in Informatics)[4], mas também pode ser usado em salas de aulas ou laboratórios para apoiar disciplinas regulares de programação na correção de trabalhos práticos (CAMPOS; FERREIRA, 2004). Dentre suas funcionalidades, destacam-se: submissão de exercícios com autenticação, controle de tempo e disponibilização de resultados em tempo real (CAMPOS; FERREIRA, 2004).

Conforme foi possível observar nos trabalhos analisados, o BOCA é direcionado a competições de programação, onde o aluno tem o objetivo de atestar seus conhecimentos práticos. Portanto, a ferramenta contempla o requisito do feedback imediato (FRANÇA;

[1] https://moodle.org/
[2] https://www.ime.usp.br/ cassio/boca/
[3] https://icpc.baylor.edu/
[4] https://ioinformatics.org/

SOARES, 2011). A inclusão de uma infraestrutura para prover o balanceamento de carga entre diversos servidores, conferiu ao BOCA a capacidade de execução em massa. De acordo com Campos e Ferreira (2004), o sistema tem o potencial de gerar interesse nos alunos através competição, o que pode ser considerado um benefício indireto. No entanto, o impacto da ideia de competição e da recompensa na motivação dos alunos a aprender é um tema bastante controverso. Enquanto muitos autores a defendem como uma estratégia efetiva para fomentar o aprendizado, outros a veem como prejudicial ao processo de aprendizagem (OLIVEIRA; BORGES, 2015). Essa falta de consenso reduz a possibilidade do BOCA de ser aplicável a contextos diversos, já que na literatura, o efeito positivo da competição na aprendizagem não é um tema pacífico.

3.1.1.4 BOCA-LAB

O trabalho de França e Soares (2011) apresentou um modelo de laboratório virtual baseado na integração do BOCA ao ambiente Moodle. Foi desenvolvida uma adaptação do BOCA, denominada BOCA-LAB para atender a necessidades específicas do Moodle, incluindo a extensão de algumas funcionalidades específicas para o trabalho em laboratórios, bem como a disponibilização de funcionalidades em forma de serviços. A integração desses dois ambientes foi realizada com o uso de Web Services (WSs).

Além do suporte ao Moodle, entre as novidades do BOCA-LAB estão: envio múltiplo de arquivos para compilação (OLIVEIRA; NOGUEIRA; OLIVEIRA, 2015), suporte on-line durante a competição, gerenciamento de times de alunos e juízes, proposição de problemas de programação, submissão e avaliação automática de soluções (FRANÇA; SOARES, 2011). Outra grande novidade apresentada em França et al. (2011) é arquitetura do modelo, que propicia uma infraestrutura para prover o balanceamento de carga entre diversos servidores, garantindo a alta disponibilidade de recursos computacionais para suportar a compilação e execução concorrente de grande quantidade de programas (FRANÇA; SOARES, 2013). França e Soares (2013) introduziu análise de similaridade nos programas submetidos para detecção de plágio entre as soluções.

Uma característica limitante do uso do BOCA-LAB é a carência de recursos com uma orientação pedagógica, como o mapeamento entre objetivos de ensino e dificuldades de aprendizagem. Do ponto de vista da correção automática, o fato da ferramenta ser restrita à verificação de entradas e saídas, não englobando outros mecanismos de análise do código, cerceia a exploração de outras abordagens, dentre as possibilidades disponíveis.

3.1.1.5 JOnline

JOnline (SANTOS; RIBEIRO, 2011a) é um JO didático para o ensino de programação no qual é possível acessar problemas para serem resolvidos e submeter códigos-fonte para a visualização do resultado (ALVES; JAQUES, 2014). De forma similar ao BOCA-LAB, a proposta do JOnline foi a de utilizar plataformas já existentes de JOs e estender suas funcionalidades, adicionando características educativas que auxiliem os discentes no aprendizado de programação (SANTOS; RIBEIRO, 2011b).

Dentre as novas funcionalidades agregadas, destacam-se a apresentação de dicas em língua portuguesa (ALVES; JAQUES, 2014), apresentação de casos de testes que geram resultados errados, organização de problemas por assunto e grau de dificuldade e votação do nível de dificuldade dos problemas pelos usuários (SANTOS; RIBEIRO, 2011b), além de programação colaborativa. Este último recurso permite que dois ou mais discentes possam desenvolver o seu código-fonte simultaneamente (SANTOS; RIBEIRO, 2011b).

Porém, apesar de adicionar funcionalidades didáticas aos juízes, auxiliando o aluno no processo de aprendizagem (ALVES; JAQUES, 2014), o JOnline apresenta a desvantagem de não estar integrado a um ambiente como, por exemplo, o Moodle que forneça outros importantes recursos (CHAVES et al., 2013).

3.1.1.6 PCodigo

O PCodigo (OLIVEIRA; NOGUEIRA; OLIVEIRA, 2015) é um complemento do AVA Moodle para apoiar a prática assistida de programação, com funcionalidades de execução em massa, representação de perfis em componentes de habilidades e análise de programas. O PCódigo foi desenvolvido na linguagem C, porém é aplicável a diferentes linguagens de programação. Integrado ao Moodle, a ferramenta aceita submissões de soluções de atividades de programação desenvolvidas por alunos, executa-as e emite relatórios de avaliação para professores (OLIVEIRA et al., 2017).

Em extensão ao PCodigo original de Oliveira, Nogueira e Oliveira (2015), foi desenvolvido o PCodigo II (NEVES et al., 2017), uma evolução da proposta inicial, inovando na sua forma de mapear perfis, com a adição da representação de perfis de estudantes por métricas de código-fonte para uma análise multidimensional e quantitativa da aprendizagem de programação (OLIVEIRA et al., 2017). O PCodigo II mapeia soluções de programação em perfis de aprendizagem representados por métricas de software para quantificar esforço e qualidade de programação (OLIVEIRA et al., 2017). Ao todo, o PCodigo II implementa 348 métricas para análise de códigos em Linguagem C e caracterização de perfis de estudantes (NEVES et al., 2017). As principais funções do PCodigo II são execução em massa de exercícios, diagnóstico de dificuldades de aprendizagem e análise de plágios

(OLIVEIRA et al., 2017).

O PCodigo II provê uma gama de métricas de avaliação da aprendizagem de programação, porém essa vasta quantidade, por si só, não garante uma avaliação realista, pois depende da interpretação de um especialista para compreender o que todas essas métricas realmente podem dizer sobre a aprendizagem dos alunos, uma vez que nem todo professor detém esse conhecimento (OLIVEIRA et al., 2018). Essa análise torna-se ainda mais dificultada pela grande quantidade de métricas a serem visualizadas e correlacionadas, fazendo-se necessária a redução de dimensionalidade das variáveis, pois segundo Oliveira et al. (2018), para realizar análises estatísticas e previsões de desempenhos, é preciso um número bem menor de variáveis. Por esse motivo, Oliveira et al. (2018), criaram uma estratégia de seleção automática de métricas de software mais relevantes para auxiliar professores em suas tomadas de decisões de avaliação, como forma de estender a proposta do PCodigo II. No entanto, esse processo é feito externamente ao PCodigo II, demandando um tempo, esforço e conhecimento adicionais para ser realizado, o que pode, entre outras consequências, atrasar o feedback aos alunos, dadas as restrições comumente presentes nesses contextos.

3.1.1.7 Feeper

O Feeper[5] (ALVES; JAQUES, 2014; SCHNEIDER; JAQUES, 2016; FEEPER, 2020) é um ambiente virtual on-line integrado para auxiliar o processo de ensino/aprendizagem de programação, com ênfase no ensino superior. Dentre seu conjunto de funcionalidades-chave, além da correção automática, podem-se citar o feedback personalizado e instantâneo para o aluno (ALVES; JAQUES, 2014) e a forma de interação entre o professor e os alunos por trocas de mensagens registradas diretamente no código do aluno.

O ambiente possui um editor de códigos fonte on-line para que os alunos possam resolver os desafios disponibilizados pelo professor (FEEPER, 2020). De acordo com Schneider e Jaques (2016), através do Feeper o aluno pode praticar conceitos de Programação Orientada a Objetos (POO), utilizando a linguagem Java e submeter suas classes de código para serem validadas por um JO, através de uma base de testes pré-cadastrados (FEEPER, 2020).

O feedback fornecido pelo JO foi personalizado para produzir mensagens explicativas que indiquem para o aluno não só a ocorrência de um eventual erro de programação, mas que apontam em quais linhas exatas do código o erro foi cometido, explicando as possíveis causas do erro e orientando sobre como proceder para corrigi-lo, fornecendo ao aluno sugestões no formato de dicas de como melhorar o seu programa, para que possa chegar a uma solução satisfatória (SCHNEIDER; JAQUES, 2016).

[5] http://feeper.unisinos.br/

Conforme visto, proporcionar um feedback de qualidade para o aluno e o foco na interação deste com o professor são os pontos fortes do Feeper. O mecanismo de feedback, que combina técnicas de análise estática e avaliação dinâmica de código. Na visão do professor, o Feeper possibilita um maior detalhamento na correção e um mecanismo de cadastro de casos de teste pelo docente que automatiza a geração de casos de teste pela validação da assinatura das classes a serem submetidas, agilizando assim a realização dessa atividade pelo professor (SCHNEIDER; JAQUES, 2016).

3.1.1.8 The Huxley

O The Huxley[6] é um ambiente web que permite aos alunos submeterem código-fonte em diferentes linguagens de programação como respostas a exercícios de programação (JESUS et al., 2019b). A base de problemas possui centenas de exercícios categorizados por tópicos de programação e por nível de dificuldade (HUXLEY, 2020). Este nível varia de 1 a 10 e é calculado dinamicamente de acordo com a quantidade de estudantes que conseguem resolver (PAES et al., 2013).

A ferramenta possui um JO que dá uma resposta imediata ao estudante diante da sua submissão, sendo direcionado para conteúdo de suporte no caso de erros (JESUS et al., 2019c). O The Huxley aceita submissões em sete linguagens de programação, a saber: Python, C, C++, Java, Octave e Pascal (HUXLEY, 2020). Para verificar a resolução de um problema, o código submetido é confrontado com a base de testes do problema através de análise sintática do código e testes de aceitação (PAES et al., 2013). O feedback apresentado é contextualizado e direcionado a estudantes iniciantes. (JESUS et al., 2019a)

No trabalho de (JESUS et al., 2019b) foi adicionada a análise estática ao Th Huxley, fornecendo um analisador estático para especificação e checagem de restrições (requisitos) relacionadas às construções em Java e Python.

O The Huxley reúne características de um AVA que tem um JO como apoio à aprendizagem de programação. fornece um feedback detalhado. No entanto, conforme afirmam Jesus et al. (2018a), embora o The Huxley aceite submissões em várias linguagens, as mensagens são detalhadas somente para submissões na linguagem Python, havendo a necessidade de fazer extensões para obter o mesmo nível de detalhes para códigos escritos nas outras linguagens suportadas (JESUS et al., 2018b).

[6] https://www.thehuxley.com/

3.1.1.9 URIOnlineJudge

Um dos JOs mais conhecidos no Brasil, o portal URI Online Judge[7] (A.; LUCA, 2012) foi criado para servir de apoio e complemento de estudos para estudantes de Engenharias e Ciência da Computação, com foco no ensino de Algoritmos e Programação (BEZ; FERREIRA; TONIN, 2013). Desenvolvido no Departamento de Ciência da Computação da Universidade Regional Integrada, Brasil, o sistema possui várias das características pertinentes a um portal de programação, tais como: correção de códigos-fonte de forma automática e em tempo real, utilização de juízes especiais, alta disponibilidade, fórum e aceitação de soluções em diferentes linguagens de programação. Além destas características, o sistema possui um toolkit embutido para que os alunos possam testar entradas e saídas de suas implementações (DAGOSTINI et al., 2017) e permite a visualização do código-fonte diretamente pelo navegador (BEZ; TONIN; RODEGHERI, 2014).

O URI Online Judge está disponível gratuitamente na internet e aceita submissões em diversas linguagens de programação. Seu repositório de problemas está dividido em nove categorias, separadas por níveis de dificuldades (BEZ; TONIN; SELIVON, 2015) e cobrindo uma diversidade de subtópicos (DAGOSTINI et al., 2018). A plataforma possui um módulo que permite que os professores monitorem o progresso do curso (BEZ; FERREIRA; TONIN, 2013). Além disso, a ferramenta oferece um ambiente de interação e colaboração entre alunos, facilitando a troca de experiências, além de possibilitar que alunos iniciantes obtenham auxílio de estudantes mais avançados (CARVALHO; FERNANDES; GADELHA, 2016).

Do ponto de vista de um sistema online de apoio à aprendizagem, o URI Online Judge se apresenta como uma das ferramentas mais completas encontradas. No entanto, na parte de avaliação e correção do código, o sistema possibilita apenas a avaliação por meio da análise dinâmica, deixando de explorar outras dimensões da aprendizagem.

3.1.1.10 CodeBench

Desenvolvido na Universidade Federal do Amazonas (UFAM), o CodeBench é um JO desenvolvido como ferramenta de apoio a uma metodologia de ensino híbrido de programação (CARVALHO; FERNANDES; GADELHA, 2016). Outros objetivos são proporcionar aos professores melhor controle da classe durante a aplicação de avaliações presenciais, servir de ambiente para resolução de exercícios de programação e automatizar a correção dos exercícios de programação (CARVALHO; FERNANDES; GADELHA, 2016).

A estrutura e o conteúdo das mensagens apresentadas pelo CodeBench foi elaborado seguindo princípios de geração de feedback oriundos da psicologia (OLIVEIRA et al.,

[7] https://www.urionlinejudge.com.br

2019). Além disso, o CodeBench também permite a troca de mensagens entre alunos e professores de uma dada turma, bem como o compartilhamento de recursos didáticos por parte dos professores (UFAM, 2020).

3.1.1.11 Onlinejudge

O Onlinejudge[8] (JUDGE, 2020), é um JO para a submissão de códigos fontes adicionado ao Moodle, para classificar automaticamente tarefas de programação testando as submissões contra casos de teste personalizáveis, com suporte às linguagens C e C++. Para ampliar o leque de linguagens suportadas, a ferramenta pode ser integrada via WS ao sistema Ideone[9] (LABS, 2020), uma aplicação com compiladores e depuradores embutidos para mais de 60 linguagens de programação, permitindo a execução remota de código-fonte diretamente pelo navegador.

O Onlinejudge é open-source, porém o Ideone é um software comercial de código fechado, o que limita o uso integrado das duas ferramentas, permitindo a submissão gratuita de apenas 1000 códigos-fonte por mês. Além disso, o Ideone não aceita a submissão de vários códigos fonte por vez (FRANÇA; SOARES, 2013).

3.1.1.12 Codeboard

Codeboard[10] é um Ambiente Integrado de Desenvolvimento (IDE, Integrated Development Environment) baseado na Web para criação e compartilhamento de exercícios de programação. A plataforma realiza a análise, inspeção e avaliação automática de submissões com base em testes de unidade ou drivers de teste personalizados e permite o acompanhamento do progresso dos alunos usando estatísticas específicas do projeto. Suporta as linguagens de programação C, C++, Eiffel, Haskell, Java, Python. Além disso, possui a integração com plataformas educacionais como Coursera[11], edX[12] e Moodle.

3.1.1.13 run.codes

O run.codes[13] é um sistema de submissão e correção automática de exercícios de programação, mas também pode ser utilizado sem esse recurso, funcionando apenas como gestor de entrega de trabalhos comuns. Com suporte a diversas linguagens como Java, C/C++, Python, Pascal, R e Octave, as submissões podem ser realizadas em um arquivo

[8] https://github.com/hit-moodle/onlinejudge
[9] https://ideone.com/
[10] https://codeboard.io/
[11] https://www.coursera.org/
[12] https://www.edx.org/
[13] https://run.codes/

único ou em diversos arquivos compactados em um arquivo Zip com Makefile. No tocante à correção automática, há diferentes abordagens para a definição de casos de teste, como, por exemplo, definição de intervalos numéricos específicos nos quais a saída esperada para o resultado do programa deve estar compreendida, além de comparação de saídas textuais e comparação de arquivos binários. Também permite análise de plágio, verificando os níveis de similaridade entre os trabalhos submetidos, além de fornecer estatísticas e entregas e desempenho da turma.

3.1.2 Sites de competições e suporte à correção de código

Além dos sistemas propostos pela literatura, há várias iniciativas mundiais voltadas para disseminação do aprendizado de programação. Existem inúmeros exemplos de sistemas JOs disponíveis na Web, dentre esses, merecem menção: Code.org[14], CodeAcademy[15], CodeChef[16] , Programming Challenges[17], Code Avengers[18], HackerRank[19], Coderbyte[20] e o Khan Academy[21].

3.2 DESAFIOS DA AVALIAÇÃO AUTOMÁTICA DE PROGRAMAÇÃO

Nesta seção, em resposta à **RQ 1**, são reportadas as principais limitações das soluções atuais para o apoio à aprendizagem de programação com suporte à análise de código. Esse levantamento foi feito no intuito de desenhar o cenário presente para direcionar a condução deste trabalho e estabelecer as bases teóricas para fundamentar a construção da ferramenta pretendida.

Romli, Sulaiman e Zamli (2010) afirmam que os sistemas de automatização e execução de testes de código-fonte tem evoluído consideravelmente. Contudo, de acordo com Oliveira, Nogueira e Oliveira (2015), ainda há deficiências consideráveis com relação à efetividade dessas ferramentas, principalmente no que se refere a averiguar o alcance dos objetivos educacionais almejados. Para Ihantola et al. (2010), esses sistemas precisam evoluir no sentido de proporcionar um maior direcionamento a partir de uma perspectiva pedagógica.

[14] https://code.orgnbmv cvn
[15] https://codeacademy.com
[16] https://codechef.com
[17] https://programming-challenges.com
[18] https://codeavengers.com
[19] https://hackerrank.com
[20] https://coderbyte.com
[21] https://pt.khanacademy.org/

Segundo Oliveira et al. (2018), poucas soluções tecnológicas têm sido desenvolvidas para identificar dificuldades de aprendizagem e habilidades de programação a partir de informações de códigos-fontes escritos por alunos. De acordo com os autores, o desenvolvimento de estratégias tecnológicas nesse sentido tem sido um desafio para a informática na educação. Para Oliveira (2015), a maioria das propostas ainda não venceu os desafios do domínio da programação introdutória.

Conforme Insa e Silva (2018), a maioria dos SAAs existentes baseia-se na comparação dos resultados dos programas analisados com uma saída esperada previamente definida, tratando o código como uma caixa preta. Uma limitação dos sistemas que adotam essa estratégia é que eles permitem analisar apenas o comportamento observável externo dos programas (INSA; SILVA, 2015). Outra restrição deste tipo de abordagem é que usualmente elas classificam os programas analisados de forma binária, isto é, considerando a solução do aluno como inteiramente correta ou completamente errada. Raposo, Maranhão e Soares Neto (2019) mencionam que tais ferramentas limitam-se a informar se o aluno conseguiu, ou não, realizar as tarefas pedidas, sem apresentar informações qualitativas acerca do código criado pelos alunos.

De acordo com Ihantola et al. (2010), um dos principais problemas identificados nos SAA analisados é que a maioria das ferramentas é desenvolvida para atender às necessidades específicas de uma classe ou tarefa. Essa ênfase no comportamento do programa, respaldada por Ihantola et al. (2010), que afirmam que a maioria das ferramentas de avaliação está focada na funcionalidade dos programas, e não avaliando estilo ou desempenho, foi um dos motivadores para elaboração da proposta aqui apresentada.

O levantamento feito por Vendrame et al. (2019) aponta que há uma escassez de relatos sobre a utilização de teorias da aprendizagem na fundamentação da construção de ferramentas ensino-aprendizagem de algoritmos e programação, o que demonstra que as ferramentas estão sendo desenvolvidas sem esta preocupação. Na pesquisa em questão, apenas dois trabalhos reportaram o uso de teorias da aprendizagem, sendo elas o cognitivismo e o construtivismo. Essa falta de uma fundamentação pedagógica é corroborada por Oliveira, Nogueira e Oliveira (2015) e Ihantola et al. (2010).

França et al. (2011) afirmam que, embora o uso de tais ferramentas possa mitigar os problemas de natureza organizacional em práticas laboratoriais, não são suficientes para solucionar a dificuldade de acompanhamento e feedback. Segundo Farias e Nunes (2019), grande parte dos estudos que apresentam ferramentas ou ambientes para ensino de programação não surtem o efeito esperado na experiência de aprendizagem do estudante. Os autores atribuem esse fato à inobservância na construção dos cenários educacionais e metodologias ativas de ensino que tornem o aluno protagonista na construção do seu conhecimento.

Sendo assim, por haver uma dificuldade em adaptar as ferramentas atuais a contextos diversos, neste trabalho foi elaborada uma alternativa centrada na flexibilização de critérios avaliativos como estratégia para alcançar maior abrangência na avaliação. Outro diferencial é a combinação da avaliação estática e dinâmica do código em conjunto com a execução automática, convenções de codificação e métricas de código-fonte. Pois não foi encontrada nas fontes pesquisadas uma ferramenta que reunisse essas características.

3.3 PERSPECTIVAS FUTURAS

Esta seção trata dos caminhos de pesquisa futuros em relação à avaliação da aprendizagem baseada na análise assistida de código-fonte, no intuito de orientar o trabalho atual rumo à construção de uma ferramenta que venha a atender as expectativas reportadas nos trabalhos mencionados e, consequentemente, pavimentar o caminho para estudos vindouros.

Romli, Sulaiman e Zamli (2010) destacaram como caraterísticas mais importantes em um sistema de avaliação automática: realizar testagens suficientes, verificar se programas estão de acordo com a especificação, reconhecer erros sintáticos e semânticos e fornecer feedback imediato. Para Oliveira (2015), o principal desafio da análise automática e semi-automática é tratar a variabilidade de soluções em códigos-fontes (analise estática) e nas saídas geradas (dinâmica).

Para Oliveira (2015 apud MOREIRA; FAVERO, 2009), as principais motivações sobre avaliação automática no Brasil giram em torno de dois tópicos principais: como avaliar manualmente todos os exercícios dos estudantes para turmas grandes e como fornecer feedback em curto prazo. Oliveira (2015) também considera a geração relatórios com uma avaliação clínica e multidimensional um recurso importante. Conforme sugerido pela autora, o uso de métricas da Engenharia de Software e ferramentas de visualização de informação podem auxiliar nessa tarefa.

Para Paes et al. (2013), o feedback imediato é importante por que os estudantes precisam receber um retorno sobre as suas tentativas de resolução de problemas o mais rápido possível, pois o não recebimento desse retorno em tempo hábil gera um acúmulo de dúvidas no estudante, chegando a prejudicar o entendimento do próximo assunto a ser assimilado pelo não entendimento do assunto anterior.

Ihantola et al. (2010) recomenda que os SAAs devem estender os Sistemas de Gestão de Aprendizagem (LMSs, Learning Management Systems) existentes com a integração da avaliação automática para evitar que seja necessário reimplementar todos os recursos de gerenciamento de curso, e assim melhor atender às necessidades específicas da educação em Ciências da Computação. Como outro caminho possível, o autor argumenta que alguns

4.1.4 Segurança e Confiabilidade

Para contemplar a premissa apontada por Oliveira (2015), a saber: Desenvolver prevenção contra códigos maliciosos submetidos por alunos, foi feita uma separação entre a parte que faz a interface web (front-end) e o módulo que provê os serviços (back-end), cada uma sendo uma aplicação executando em servidores diferentes e que se comunicam por meio de requisições usando o protocolo Hypertext Transfer Protocol (HTTP). Esse isolamento reduz um eventual impacto na performance do sistema, caso haja um fluxo muito grande de requisições. Essa estrutura também protege o serviço de ter sua disponibilidade interrompida, prevenindo contra possíveis ataques de Denial Of Service (DoS) e Distributed Denial Of Service (DDoS) ou ainda impedir que dados não autorizados sejam acessados por meio da submissão de código malicioso pelo módulo aluno.

Do ponto de vista de invasões, as transferências de dados são feitas através do Secure Hypertext Transfer Protocol (HTTPS), a versão segura do protocolo de páginas HTTP. A comunicação com o banco de dados é aceita conexões seguras com autenticação e Secure Sockets Layer (SSL). O site da aplicação front-end foi configurado com um certificado Transport Layer Security (TSL) para estabelecer uma comunicação segura com o servidor, prevenindo a execução de scripts danosos ao sistema por programas mal intencionados. O TSL possibilita a segurança digital por meio da comunicação criptografada entre um site e um navegador.

4.1.5 Possibilidade de integração com outras plataformas

Para cobrir a questão levantada por Ihantola et al. (2010), o CodeTeacher foi concebido de forma a permitir a comunicação com outras plataformas de ensino, para que exercícios de programação sejam submetidos por meio de ambientes virtuais ou outras interfaces web. A primeira plataforma escolhida para integração foi o Google Classroom. Os detalhes dessa comunicação são descritos na Seção 4.4.

4.1.6 Características adicionais

Outros requisitos complementares foram especificados para o CodeTeacher. Dentre as demais funcionalidades disponibilizadas pelo sistema, destacam-se:

Suporte a internacionalização. O CodeTeacher possui suporte para aos idiomas português e inglês, este último sendo o idioma oficial da aplicação. Por padrão, o idioma é configurado automaticamente no primeiro acesso do usuário à plataforma, de acordo com a localidade (locale) da máquina cliente. Caso não seja encontrado o idioma correspondente, o idioma padrão permanece.

dos sistemas de avaliação podem se transformar em LMSs, mas para isso, os sistemas devem ser modulares o suficiente.

Após a leitura completa dos trabalhos consultados, foram estabelecidas as seguintes premissas para um software que vise a superar os desafios relatados, bem como contemplar as perspectivas futuras:

1. Flexibilidade quanto às modalidades de avaliação para tratar a variabilidade de soluções dos exercícios de programação (NAUDÉ; GREYLING; VOGTS, 2009; OLIVEIRA, 2015);
2. Feedback imediato (PAES et al., 2013), coerente, detalhado (BUYRUKOGLU; BATMAZ; LOCK, 2019), personalizado (ALVES; JAQUES, 2014; SCHNEIDER; JAQUES, 2016) e contextualizado;
3. Correção parcial (INSA; SILVA, 2018; INSA; SILVA, 2015);
4. Segurança, confiabilidade e proteção contra códigos maliciosos (OLIVEIRA, 2015)
5. Análise de plágios (OLIVEIRA; NOGUEIRA; OLIVEIRA, 2015);
6. Orientação a uma perspectiva pedagógica (IHANTOLA et al., 2010) e baseado em teorias da aprendizagem (VENDRAME et al., 2019);
7. Integração com AVAs, LMSs (IHANTOLA et al., 2010).

3.3.1 Análise comparativa de ferramentas

Durante a pesquisa e desenvolvimento deste trabalho, 11 ferramentas de avaliação foram utilizadas em caráter de teste para verificar a aderência aos princípios extraídos da literatura. As ferramentas foram analisadas comparativamente, dentro de uma apreciação crítica. As características-chave usadas na comparação foram as mesmas de Caiza e Alamo (2013). Na Tabela 3.1 as ferramentas testadas são comparadas com os requisitos estabelecidos.

Conforme pode-se observar na Tabela 3.1, percebeu-se que, embora todos os trabalhos citados contenham importantes contribuições para o apoio ao EAPC, nenhuma das ferramentas estudadas preenche todos os requisitos. Ao constatar dificuldades encontradas nas ferramentas de avaliação estudadas, neste trabalho propõe-se um ambiente de auxílio a correção automática de programas que seja capaz de reunir as características supramencionadas, além das funcionalidades próprias de um JO que são inerentes. O grande diferencial do presente trabalho em relação aos anteriores reside no método de avaliação.

Tabela 3.1 – Análise comparativa de ferramentas

Nome	**Estática**	**Dinâmica**	**Análise de Plágio**	**Métricas de software**	**Convenções de código**	**Licença**	**Linguagens suportadas**
URI Online Judge	Não	Sim	Sim	Não	Não	Gratuito	C/C++, C#, Java, Python, Pascal, Scala, Ruby, Lua, JavaScript, Kotlin, PSQL
Uva	Não	Sim	Não	Não	Não	GNU General Public License v3.0	C/C++, Java, Python, Pascal
Onlinejudge	Não	Sim	Não	Não	Não	GNU General Public License v3.0	C e C++
Ideone	Não	Sim	Não	Não	Não	Comercial	60+
PCodigo	Sim	Sim	Sim	Sim	Não	Fechado	Python
BOCA	Não	Sim	Não	Não	Não	GNU General Public License v3.0	C, Java
Progtest	Não	Sim	Não	Não	Não	GNU General Public License v3.0	Java
The Huxley	Sim (Java e Python)	Sim	Sim	Não	Não	Fechado	Python, C, C++, Java, Octave e Pascal
Codeboard.io	Sim	Sim	Não	Não	Não	-	C, C++, Eiffel, Haskell, Java, Python
run.codes	Sim	Sim	Não	Não	Não	-	Java, C/C++, R, Octave, entre outras
CodeBench	Sim	Sim	Não	Não	Não	-	C/C++, Python, Pascal

Fonte: Autor

4 CODETEACHER

O CodeTeacher é uma plataforma educacional unificada, focada em atividades práticas voltadas ao estudo de programação de computadores. Constitui-se de uma aplicação web que integra várias abordagens de avaliação automática de código-fonte, na qual o professor pode configurar a detecção automática de inconformidades no código dos alunos a partir da definição de critérios avaliativos pré-configurados, para garantir versatilidade e agilidade na avaliação. O aluno, por sua vez, pode submeter o seu trabalho, e automaticamente visualizar sua nota ao fim da submissão, recebendo um feedback instantâneo, coerente e personalizado.

A plataforma foi desenvolvida no intuito de aprimorar a experiência de avaliação para torná-la totalmente on-line e automática, disponível de forma ubíqua na internet, tendo como características principais a flexibilidade dos critérios avaliativos e o feedback individualizado, com o objetivo de ser um instrumento de avaliação de apoio amplo à prática assistida de programação. A sistemática de avaliação adotada apoia-se na combinação de análise estática, execução automática e testes, além da verificação de convenções de código e análise por métricas de código-fonte. O CodeTeacher mapeia estes conceitos e os integra em uma plataforma de monitoramento que avalia a qualidade da prática de programação dos alunos. Em sua versão atual, a ferramenta analisa códigos-fonte escritos na linguagem Java e funciona de forma integrada ao Google Classrom (GOOGLE, 2020a).

A seguir são detalhadas as principais características do CodeTeacher, conforme segue: na Seção 4.1 são relatados as motivações e os requisitos que deram origem à concepção do CodeTeacher, a Seção 4.2 discute aspectos de implementação do projeto. A Seção 4.3 dá uma visão geral do fluxo de funcionamento, adentrando nos aspectos de uso, bem como os dados, atores e as interações envolvidas no processo. A Seção 4.4 trata da integração a outras plataformas. Enfim, a Seção 4.5 relata limitações atuais do CodeTeacher e discute funcionalidades planejadas para versões futuras.

4.1 PREMISSAS E REQUISITOS

Conforme relatado no capítulo 3, no qual foram discutidos os trabalhos que nortearam a idealização da proposta aqui apresentada, o CodeTeacher foi concebido com base na literatura recente e relevante acerca dos tópicos de pesquisa aos quais ele visa contemplar, logo, o escopo da proposta foi elaborado a partir da investigação das dificuldades, deficiências e mazelas das ferramentas existentes, buscando identificar oportunidades de desenvolvimento de soluções que preencham satisfatoriamente as lacunas em aberto relata-

das nos trabalhos anteriores. A seguir, serão detalhadas as principais características do CodeTeacher, relacionando-as com os estudos que fundamentaram e justificaram a escolha do conjunto de funcionalidades que foram selecionadas como características primordiais. Cada uma dessas premissas, bem como os requisitos decorrentes delas, são apresentadas nas subseções a seguir.

4.1.1 Flexibilidade quanto às modalidades de avaliação

Para atacar o problema da variabilidade de soluções relatado por Oliveira (2015), foram definidos os seguintes requisitos:

Uso de máscaras na definição de critérios. É possível lançar mão de expressões regulares (JARGAS, 2016) para definição de elementos de código, como por exemplo, nomes de classes e membros de classes, interfaces, parâmetros de métodos e exceções. A inclusão deste recurso torna mais flexível a definição de critérios, uma vez que permite abranger uma maior variabilidade nos nomes dos elementos a serem buscados no código. Por exemplo, caso deseje-se averiguar a existência de uma determinada classe chamada "CarroConversivelImportado", pode-se definir o critério de que deve haver uma classe cujo nome inicie com o prefixo "Carro" ou que contenha "Conversivel", ou até mesmo que termine com "Importado". Para isso, o professor deve saber criar expressões regulares. Também é possível especificar se devem ser considerados rigorosamente os caracteres maiúsculos e minúsculos (case sensitive) ou se devem ser ignorados, enfim, as possibilidades são tantas quanto o uso de expressões regulares permitir.

Ponderação de critérios. Com um valor cada item avaliativo, é possível identificar, de forma mais detalhada, não somente quais as falhas de implementação no código de um aluno, como também qual a sua relevância diante dos objetivos de aprendizagem, de acordo com os pesos dados a cada critério.

4.1.2 Correção parcial

Para o problema descrito em Insa e Silva (2015), foram desenvolvidas as seguintes funcionalidades:

Independência de critérios. Essa abordagem permite avaliar o código mesmo que este não esteja totalmente correto, podendo realizar avaliações parciais. A análise arquivo-a-arquivo permite detectar erros de codificação específicos e de forma independente, prevenindo, por exemplo, que um erro de compilação comprometa toda a avaliação, permitindo uma avaliação mais precisa.

Criação e compartilhamento de configurações e de interpretações de

critérios. Professores devem ser capazes de reaproveitar configurações, publicá-las, compartilhá-las e derivar novas a partir de existentes. Essa possibilidade faz com que o professor perca menos tempo configurando critérios para cada atividade.

4.1.3 Feedback imediato, coerente, personalizado e contextualizado

Para suprir a demanda de oferecer um feedback rápido e adequado ao avaliar-se o desempenho do aluno Paes et al. (2013), foram especificados os seguintes requisitos:

Status de entrega de tarefas. As soluções submetidas pelos alunos seguem um fluxo padrão, que passa pelos estados:

1. Pendente. O aluno nunca acessou este envio. Os anexos não são retornados.
2. Enviado. Foi entregue ao professor.
3. Devolvido. Foi devolvido ao aluno.
4. Cancelada. O aluno optou por "cancelar o envio"da tarefa.

Os alunos são notificados sobre o estado de suas tarefas em tempo real, com as informações de data e hora da mudança de status. Ao fornecer informações instantâneas sobre a situação de sua resposta, o aluno toma ciência de suas falhas e acertos de forma imediata, tornando o aprendizado mais responsivo. Ao final da submissão, o aluno pode ter o resultado da avaliação, com sua respectiva nota quanto a abordagem por ele utilizada para montar a solução do problema proposto. Esse recurso pode ser desabilitado pelo professor, para evitar que o aluno adote um comportamento de tentativa e erro. Caso desabilitado, apenas uma submissão é permitida.

Para fornecer um feedback de qualidade, além de rápido, este deve fornecer as informações necessárias para a correta compreensão do erro. Para cobrir esse item, foram especificados os requisitos:

Personalização de mensagens. Nos relatórios resultantes da execução de um conjunto de critérios, são exibidas mensagens que contém uma descrição do erro. O professor pode complementar o feedbak com algum comentário direcionado para as peculiaridades de uma determinada avaliação. Essa funcionalidade dá características de avaliação semi-automática ao CodeTeacher.

Relatório explicativo de erros. O aluno recebe um relatório detalhado, com o tipo de erro e a possível causa. Quando aplicável, a mensagem vem acompanhada de sugestões para resolução do problema e a indicação das linhas de código problemáticas. Além do comentário adicional do professor, caso ele tenha inserido.

Visualização do código-fonte diretamente pelo navegador. Com esse recurso, a avaliação acontece totalmente no ambiente, sem que seja necessário baixar o código em separado e selecionar um diretório local para análise, caso o professor deseje revisar o código do aluno.

Disponibilização de um painel de indicadores educacionais no qual estatísticas de aprendizagem poderão ser exibidas para conhecer o histórico de programas escritos.

4.2 ARQUITETURA

Esta seção descreve as propriedades arquiteturais do CodeTeacher, são discutidos o modelo de arquitetura e a modularidade do sistema.

4.2.1 Modelo arquitetural

De acordo com Sommerville (2011), a arquitetura de um software é concernente a como o sistema está organizado estruturalmente. O desenho da arquitetura é o primeiro estágio do processo de projeto de software, pois descreve como o sistema está organizado em um conjunto de componentes de comunicação (SOMMERVILLE, 2011).

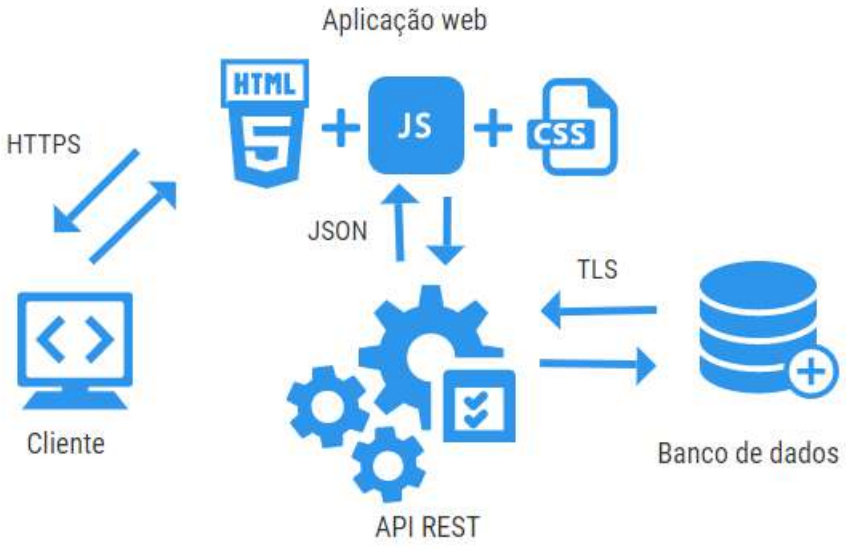
Conforme Santos, Segundo e Telvina (2017), o CodeTeacher foi originalmente desenvolvido como uma aplicação desktop e, neste trabalho, foi evoluído para uma nova versão, com o desenvolvimento de um serviço web e uma interface para sua disponibilização como serviço on-line. A especificação da arquitetura atual do CodeTeacher segue o modelo cliente-servidor e foi desenvolvida de forma desacoplada, seguindo o conceito de arquitetura modular.

No modelo cliente-servidor os computadores se comunicam através de uma rede, designando os papéis e as cargas de trabalho entre as máquinas que fornecem um recurso ou serviço, chamadas servidores, e aquelas requerentes dos serviços (clientes). Isto significa que há uma separação de interesse entre os dois papéis: o servidor se encarrega de organizar como os dados são armazenados internamente e responder as requisições, sem implicar em como processá-las na interface do usuário, pois essa é uma responsabilidade do cliente. Esta característica torna o funcionamento independente entre as partes.

Uma arquitetura modular pode ser definida como um sistema composto por um conjunto de módulos que pode ser modificado de forma independente, onde cada um dos componentes separados podem ser substituídos ou adicionados sem afetar os demais componentes do sistema (SOMMERVILLE, 2011). O modelo de arquitetura da Figura 4.1 mostra os componentes arquiteturais do CodeTeacher e os relacionamentos entre eles,

destacando os serviços disponibilizados pelos componentes e as regras de comunicação entre estes.

Figura 4.1 – Principais componentes estruturais do CodeTeacher



Fonte: Autor

O CodeTeacher está disponível como um sistema web, seguindo o modelo arquitetônico de software RESTful, que é uma implementação de um "web service"[1] utilizando o protocolo HTTP e os princípios Transferência de Estado Representacional (REST, Representational State Transfer). Em sistemas que seguem o estilo REST, a aplicação é vista como um conjunto de recursos que representam o seu estado. Esses recursos, quando acessados, transferem ou alteram o estado (conteúdo) da aplicação. Os recursos individuais são acessados e/ou alterados por meio de ações executadas pelos métodos disponíveis no protocolo HTTP.

A identificação do recurso solicitado é descrita pela URI Uniform Resource Identifier (URI) específica do serviço. As informações necessárias para o acesso ao recurso a ser consumido devem estar contidas no corpo da requisição. Geralmente, o cabeçalho carrega dados como credenciais de acesso, tokens de autenticação e detalhes do serviço são passados via parâmetro. A identificação do tipo de conteúdo que está sendo negociado é feita pelo campo cujo o nome é Content-type.

4.2.1.1 Implementação do Front-end e back-end

O front-end foi construído usando JavaScript como linguagem de programação e utilizando notações JavaScript Object Notation (JSON) para intercâmbio de dados.

[1] Solução utilizada na integração de sistemas e na comunicação entre aplicações diferentes

Dessa forma, a aplicação torna-se acessível de qualquer dispositivo compatível com essas tecnologias, não sendo necessária sua instalação prévia, bastando apenas possuir um dispositivo com um navegador web e acesso à internet. Para a criação das páginas web foram utilizados as tecnologias Linguagem de Marcação de Hipertexto, versão 5 (HTML5, HyperText Markup Language), Cascading Style Sheets (CSS3) e Javascript. A interface foi desenhada com auxílio do template Gentelella[2] de suporte do Bootstrap[3] e JQuery[4]. Como template engine, foi utilizado o Thymeleaf[5].

No back-end, o CodeTeacher foi escrito com a linguagem de programação Java, utilizando os recursos do ecossistema do Spring Framework[6]. O Sistema Gerenciador de Banco de Dados (RDBMS, Relational Database Management System) para o armazenamento dos dados e o MySQL[7], juntamente com o servidor web Apache[8], que recebe e processa as requisições da interface. Os dados mantidos e acessados pela plataforma são armazenados em servidores em nuvem, possibilitando maior segurança e ubiquidade.

As duas aplicações estão implantadas em servidores virtualizados distintos, ambos hospedados na infraestrutura de um provedor web de aluguel de máquinas na nuvem, com arquitetura elástica, permitindo que a ferramenta se adapte de acordo com a quantidade de usuários através do uso de múltiplos servidores; e tecnologia de escalonamento automático, possibilitando que o sistema conte com recursos de processamento adequados à demanda ou diminua esses recursos em um período de baixa atividade.

4.2.2 Organização Interna

O código-fonte do CodeTeacher está dividido em 3 projetos:

CodeTeacherCore: Contém o código das funcionalidades núcleo do CodeTeacher: as classes da interface de programação de aplicação (API, Application Programming Interface), incluindo suas dependências; as implementações das funcionalidades básicas; interfaces dos pontos de acesso e arquivos de configuração. Os outros projetos têm esse como dependência.

CodeTeacherWeb: Referente às funcionalidades do front-end, serve como camada de visualização, englobando as implementações dos pontos de acesso, páginas HTML, bibliotecas e arquivos de configuração necessários para implantação do serviço web.

CodeTeacherTests: Compreende toda a estrutura de testes automatizados, com

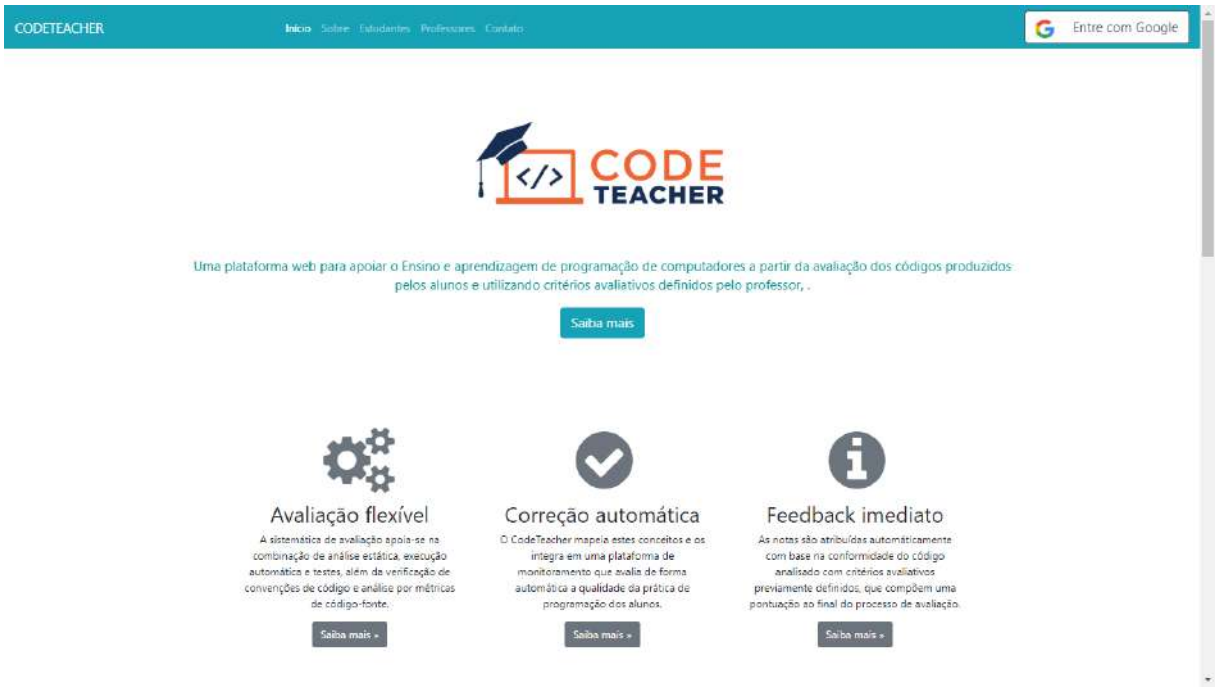
[2] https://colorlib.com/polygon/gentelella/
[3] https://www.getbootstrap.com/
[4] https://www.jquery.com/
[5] https://www.thymeleaf.org/
[6] https://spring.io/
[7] https://www.mysql.com/
[8] https://httpd.apache.org/

uma suíte de testes de unidade e uma bateria de testes de aceitação.

4.2.3 Interface visual

A interface web de apresentação do CodeTeacher possui uma implementação que segue o paradigma responsivo, isto é, possui a capacidade de adequar a resolução de suas páginas e reorganizar os componentes visuais de acordo com as dimensões da tela do dispositivo cliente, seja um computador convencional (PC), notebook, tablet ou smartphone. Tais características tornam o CodeTeacher adaptável para assim proporcionar uma experiência de navegação apropriada ao contexto de uso, conforme o espaço disponível para exibição permitir. A Figura 4.2 mostra a página inicial do CodeTeacher.

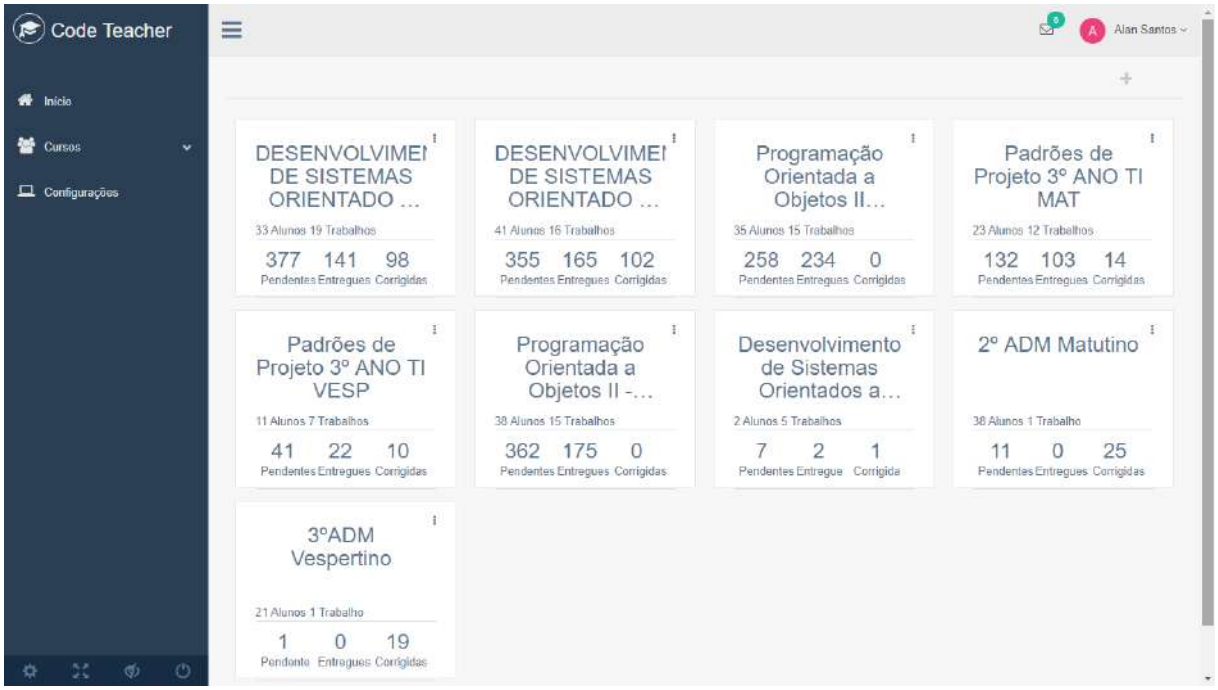
Figura 4.2 – Página inicial do CodeTeacher



Fonte: Autor

A visão do usuário é baseada no conceito de dashboard, exibindo um painel de indicadores com as informações mais relevantes para o usuário e um conjunto de ações possíveis já na página principal do sistema. Após o login, o usuário é encaminhado a uma página principal com a listagem das turmas ativas, que permite uma visualização global do status de exercícios, avaliações e conteúdo, além de fornecer opções para execução de fluxos comuns (Figura 4.3).

Figura 4.3 – Página principal de usuário do CodeTeacher



Fonte: Autor

4.3 FUNCIONAMENTO E USO

Basicamente, os critérios de avaliação automática do CodeTeacher podem ser agrupados nas seguintes categorias:

1. Compilação
2. Análise estática
3. Análise dinâmica
4. Verificação de convenções de código
5. Extração de métricas de código-fonte

Sendo assim, o CodeTeacher é composto por cinco grupos de critérios, os quais podem funcionar de maneira isolada. Com exceção da etapa de compilação, todos os demais grupos de critérios são opcionais e independentes entre si, ficando a cargo do avaliador o seu uso. Recomenda-se, contudo, o uso misto dos grupos de critérios, permitindo combinar seus efeitos para explorar as possibilidades de avaliação e alcançar uma análise mais abrangente.

Todas essas possibilidades são disponibilizadas para o usuário no momento da configuração dos critérios via interface visual. O nível de granularidade da avaliação e o peso dos itens avaliativos depende do professor/avaliador que estiver usando a ferramenta, sendo possível flexibilizar a análise considerando apenas alguns elementos na avaliação. Os detalhes de cada grupo de critérios são descritos nas subseções a seguir.

4.3.1 Compilação

A fase de compilação usa internamente o compilador do Java Developer's Kit (JDK) instalado na máquina servidora da aplicação, que reporta potenciais erros encontrados durante o processo de compilação. Essa etapa é executada implicitamente toda vez que algum trabalho é avaliado, pois as demais análises são feitas a partir dos arquivos resultantes da compilação do código do aluno. A Figura 4.3 ilustra um exemplo de erro de compilação. Quando ocorrem erros de compilação (Compilation Errors), o usuário recebe como resposta a descrição do erro, com indicação das linhas do código-fonte que geraram os erros.

4.3.2 Análise estrutural

Para analisar estaticamente os elementos estruturais do código, foi utilizado o recurso da programação reflexiva, também conhecida como reflexão ou metaprogramação (HORSTMANN; CORNELL, 2000). Esse paradigma provê o uso de metainformações para examinar a estrutura e o estado de um programa e até mesmo alterar seu comportamento em tempo de execução. Como a linguagem Java suporta metaprogramação, foi explorado esse recurso da própria linguagem para detecção automática de inconformidades no código. A Figura 4.4 mostra uma visão geral do processo de análise estática no CodeTeacher.

O desenvolvimento desse mecanismo permite avaliar se a solução do aluno atende a todos os requisitos de assinatura definidos no exercício, pois é possível testar a assinatura de construtores, métodos de acesso, de modificação de atributos e demais métodos, verificando se há na solução do aluno uma classe apresente a estrutura definida pelo professor. Essa estratégia permite ainda analisar a aplicação de conceitos da POO como emprego de herança, polimorfismo e encapsulamento, entre outros (HORSTMANN; CORNELL, 2000).

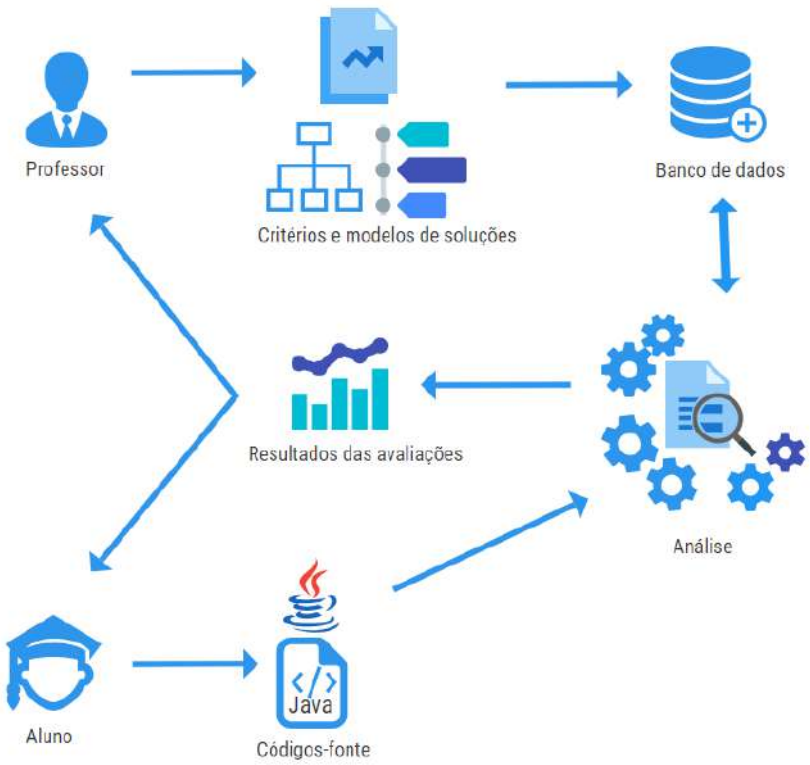
Para analisar o encapsulamento, podem ser verificados os níveis de visibilidade dos membros (atributos e/ou métodos) de uma classe, através da leitura de seus modificadores de acesso. A análise do uso de herança pode ser feita por meio da navegação pela hierarquia de classes implementada pelo aluno para detectar se determinada classe a ser herdada está presente. Para a análise do uso de polimorfismo pode-se, por exemplo, checar se determinada classe implementa ou não uma dada interface, de modo similar à verificação

da herança. O uso de polimorfismo pode ainda ser identificado e avaliado considerando a presença de métodos sobrecarregados e/ou sobrescritos.

É possível ainda analisar sintaticamente outros detalhes de implementação como a declaração correta dos atributos e métodos de uma classe. Por exemplo, pode-se configurar um critério que avalie se há, no escopo definido, algum método que retorne dados e/ou receba parâmetros de determinados tipos, bem como se esse método é de classe ou instância, não sendo necessário informar o nome do método a ser buscado, podendo-se lançar mão do uso máscaras na definição do critério.

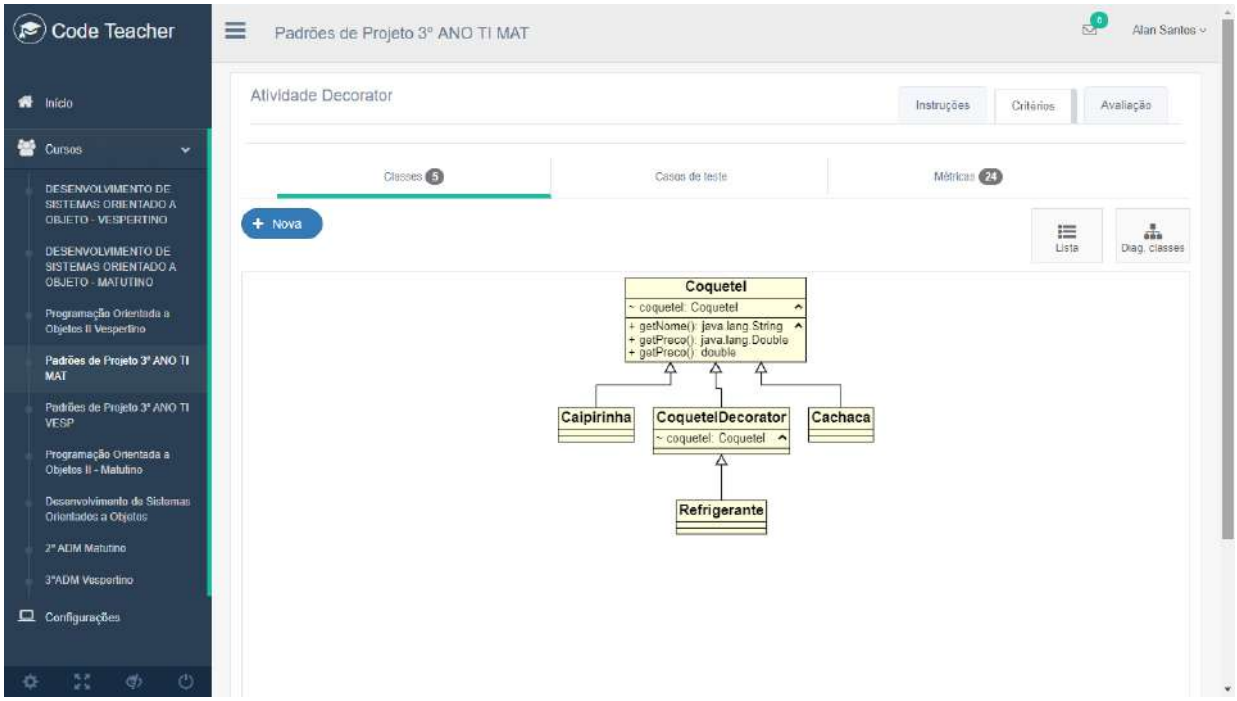
A Figura 4.5 mostra um exemplo de configuração de critérios de análise estática no CodeTeacher utilizado por um professor para realizar verificações estáticas nas soluções dos alunos. Nesse caso, o professor está interessado em descobrir se o aluno está implementando uma hierarquia de classes na qual a classe raiz é denominada "Coquetel", tendo como filhas as classes "Caipirinha", "Cachaca"e "CoquetelDecorator", esta última possuindo uma filha chamada "Refrigerante". A classe raiz deve possuir o atributo "coquetel"e implementar os métodos "getNome()"e "getPreco()", que serão herdados pelas classes filhas, conforme detalhado nas Figuras 4.6 e 4.7. A assinatura desses métodos devem respeitar as especificações do professor.

Figura 4.4 – Visão geral da análise estática no CodeTeacher



Fonte: Autor

Figura 4.5 – Configuração de critérios de análise estática no CodeTeacher



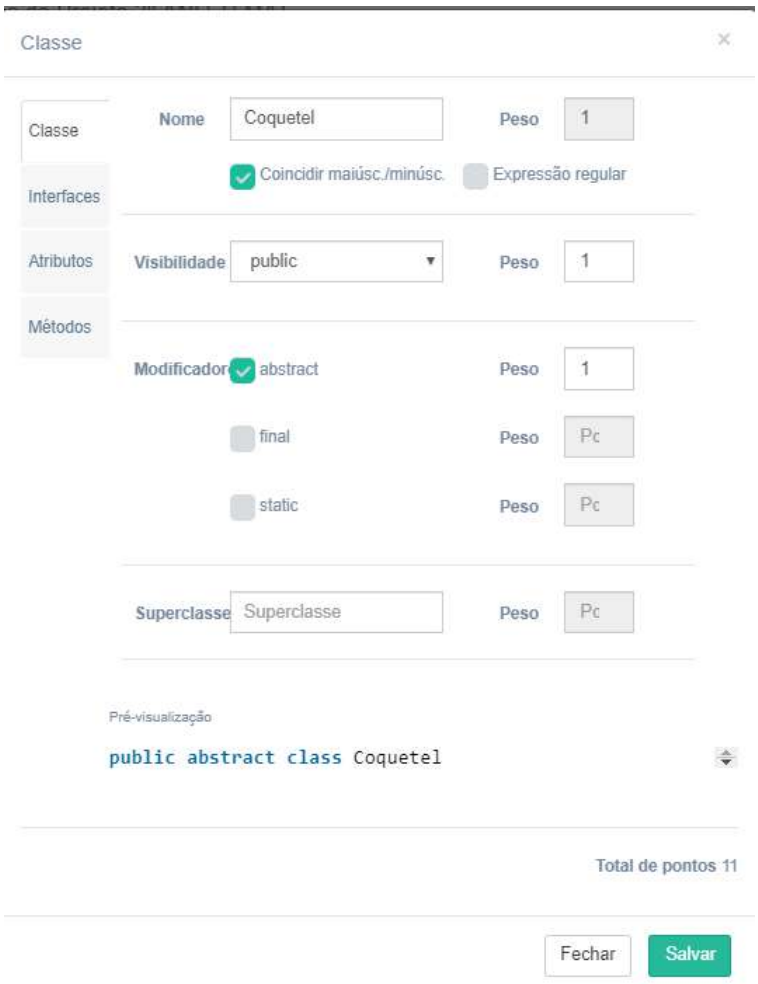
Fonte: Autor

4.3.3 Análise em tempo de execução (runtime)

No CodeTeacher, a análise dinâmica é baseada em pares de valores de entrada e saída para testar os serviços providos pelo programa analisado. Essa verificação é feita para checar a corretude do código a partir de sua testagem funcional. O professor modela um conjunto de casos a serem avaliados e submete-os para serem apreciados pela ferramenta e, através da combinação das entradas fornecidas e saídas previstas, é possível inferir a qualidade do programa quanto às suas funcionalidades, sendo possível com isso afirmar se o programa desenvolvido corresponde aos requisitos especificados pelo professor. O objetivo desse tipo de teste é simular o comportamento do programa em um ambiente ou cenário real. Um exemplo de uso dessa abordagem pode ser a checagem do retorno de métodos a partir da invocação com passagem de uma lista parâmetros pré-estabelecidos e da definição dos respectivos retornos esperados.

Outra possibilidade é a verificação da saída escrita pelo programa executado no console da aplicação, onde é possível comparar o resultado impresso pelo programa na saída padrão com um texto definido previamente pelo professor, esse texto pode, inclusive, ser parametrizável, funcionando como um template do resultado esperado. Há ainda opções para remover os espaços em branco, quebras de linha e tabulações da resposta, além da possibilidade de ignorar diferenças entre maiúsculas e minúsculas, para evitar que pequenos erros de digitação eventualmente cometidos pelo aluno prejudiquem sua

Figura 4.6 – Configuração de uma classe no CodeTeacher



Fonte: Autor

avaliação. O rigor e a tolerância a esses detalhes ficam a cargo do professor.

4.3.4 Análise de convenções de codificação

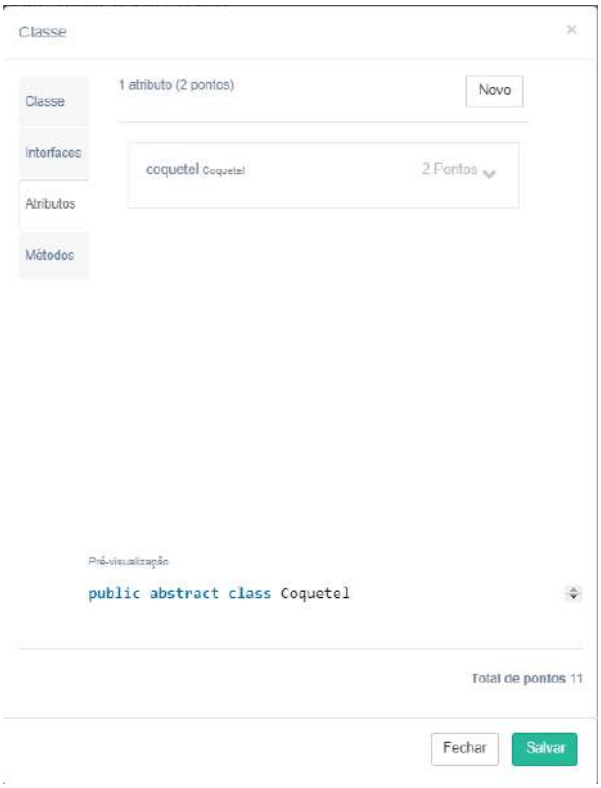
A Figura 4.8 mostra um exemplo de relatório de erros de convenções de codificação. Esta funcionalidade se enquadra no contexto da análise estática.

Para a verificação de convenções de código foi usado o Checkstyle (CHECKSTYLE, 2016), uma ferramenta para ajudar programadores Java a seguirem padrões de codificação. O Checkstyle analisa o código e fornece um conjunto de notificações em tempo real, acusando violações no padrão de codificação configurado, e indicando os trechos em que o código viola os padrões, automatizando assim o processo de verificação do código, a fim de que este seja aderente a um padrão de codificação.

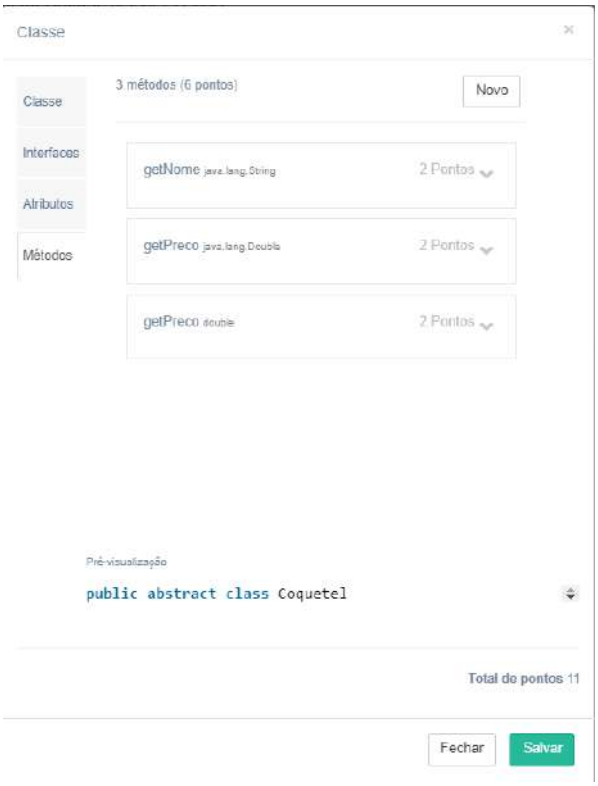
Apesar de haver outras ferramentas similares, o Checkstyle foi escolhido por ser melhor documentado e mais amplamente utilizado, pois atualmente existem plug-ins do

Figura 4.7 – Configuração de atributos e métodos

(a) Atributos



(b) Métodos



Fonte: Autor

Checkstyle para diversas IDE e editores de código. Outro critério norteador da escolha foi devido à sua facilidade de integração, pois, por ser escrito em Java, a implantação pôde ser feita empacotando o código como dependência do CodeTeacherCore, dispensando uma instalação separada.

4.3.5 Análise por métricas de código-fonte

O CodeTeacher dispõe de 50 métricas de código-fonte configuráveis, as quais o professor-avaliador pode lançar mão para compor o seu conjunto de critérios avaliativos em uma determinada atividade. Esta caraterística faz parte do conceito de análise estática. Ao cadastrar um critério, o usuário pode escolher as métricas que ele considera mais relevantes para a avaliação pretendida, estabelecer diferentes pesos para métricas selecionadas, definir intervalos de valores para as métricas escolhidas e desenhar a forma como os resultados serão agregados. Além disso, é possível criar métricas compostas a partir das disponíveis. Essas configurações podem ser usadas para avaliar, comparar e acompanhar os trabalhos dos alunos. Tais fatores podem ser enquadrados em uma escala de gradação definida pelo professor. Como exemplo, pode-se medir o nível de encapsulamento configurando

um percentual mínimo de membros encapsulados a ser alcançado. A Figura 4.9 mostra a janela de configuração de métricas.

A ferramenta-base escolhida para o cálculo de métricas foi o Checkstyle por possuir um conjunto considerável de métricas de código-fonte embutidas que possibilita configurar intervalos para os valores analisadas para medir a aderência do código a regras. Estas, por sua vez, também podem ser usadas para definir diversas métricas de código-fonte, checando violações de tamanho, complexidade, acoplamento, profundidade de aninhamento, entre outras.

O CodeTeacher encapsula o Checkstyle, adaptando as verificações existentes, isto é, o CodeTeacher coleta as observações apontadas pelo Checkstyle e converte-as em valores numéricos, registrando-os como não cumprimento a critérios. Essa adaptação foi necessária devido ao fato de o Checkstyle não ter sido projetado originalmente para calcular métricas, apenas para fazer os avisos de violação de conformidade com um conjunto de padrões de codificação previamente definido.

4.3.6 Processamento

O processamento dos critérios avaliativos no CodeTeacher se inicia após os alunos realizarem a submissão de soluções de programação na interface do aluno e consiste em calcular, para cada solução submetida para uma atividade de programação, os valores para os respectivos itens a serem avaliados. Esse processo é realizado em três fases: carregamento, coleta e análise. Na fase de carregamento, o código-fonte é copiado para a memória principal da instância da máquina servidora do CodeTeacher. Na coleta, são extraídos os dados necessários para fornecer os critérios especificadas na configuração. A fase de análise utiliza todas as informações obtidas na fase de coleta para gerar a árvore de resultados, com todos os critérios compostos, valores agregados e associações com os intervalos configurados. Em seguida, efetua-se o cálculo das estatísticas para os dados coletadas e os resultados são exibidos com base nas configurações cadastradas. A Figura 4.10 exemplifica o resultado geral de uma avaliação. As Figuras 4.11 e 4.12 apresentam a visualização de erros e notas, respectivamente.

4.4 INTEGRAÇÃO

Um dos requisitos especificados ao projetar-se o CodeTeacher foi a necessidade de integração com outras plataformas educacionais. Esta seção mostra como foi feita essa integração. Primeiro será discutido como o CodeTeacher foi projetado para dar suporte a outras plataformas e assim garantir a compatibilidade com outros sistemas e serviços.

Depois, será mostrado como foi implementada a comunicação com a plataforma Google Classroom.

4.4.1 Interfaces de comunicação

Para dar suporte a plataformas de terceiros, a comunicação pode ser feita de duas formas, diferindo na escolha da plataforma que servirá de interface principal. Na primeira abordagem, o LMS deve ser adaptado para fazer as chamadas aos recursos da API REST do CodeTeacher, nesse caso, todo o tratamento dos dados retornados pela API devem ser tratados na aplicação principal, assim o uso do CodeTeacher acontece de forma transparente para o usuário final. Essa forma de integração exige que a aplicação cliente (nesse caso, o AVA ou LMS que deseja consumir os serviços do CodeTeacher) seja autenticada a cada requisição, passando uma credencial via parâmetro.

Na segunda abordagem, um módulo de extensão deve ser criado no próprio CodeTeacher, de maneira a permitir a integração entre os ambientes. Este módulo de extensão deve permitir o acesso à funcionalidades disponibilizadas pelo LMS; usar estruturas específicas para registro dos dados; apresentar interfaces para submissão de soluções e para apresentação dos resultados, ambos a partir da interface do CodeTeacher. Ou seja, todo o processo é realizado no CodeTeacher, este sendo uma fachada para abstrair as funcionalidades da aplicação de terceiros. Obviamente, para que isso seja possível, a aplicação deve permitir que seus serviços sejam acessados por outas aplicações.

4.4.2 Google Classroom

O Google Sala de Aula (GC, Google Classroom)[9] foi escolhido como o primeiro LMS a ser usado como plataforma de apoio, pois oferece os recursos necessários para a gestão de alunos, registro de atividades e notas, entre outros aspectos administrativos das atividades educacionais. Faz parte da suíte de aplicativos Google for Education[10] (GOOGLE, 2020b), desenvolvidos para o fomento e utilização na educação.

A ferramenta GC tem como principais aspectos facilitadores do seu uso o fato de ser livre e gratuita, necessitando apenas ter uma conta no sistema para fazer uso dela. O GC facilita a entrada via login direto na plataforma e concentra diversas ferramentas e funcionalidades do Google, assim, facilitando atividades educacionais e a integração de diversas ferramentas online disponibilizadas pelo Google como: Gmail, Google Drive, Hangouts, Google Docs e Google Forms, entre outros.

[9] https://classroom.google.com/
[10] https://edu.google.com/

O Google for Education disponibiliza uma API para que os serviços que não são do Google possam aproveitar as ferramentas e a infraestrutura do GC, dessa forma, escolas e empresas de tecnologia podem adaptar os serviços providos pela plataforma. A Classroom API[11] é uma API para desenvolvedores criarem ferramentas que interagem com o GC e fazê-lo funcionar melhor de acordo com as suas necessidades. Para usar a Classroom API, os desenvolvedores precisam concordar com Termos de Serviço da Classroom API.

Com a API do Google, é possível realizar de forma programática muitas das ações que os professores e alunos podem fazer pela interface do usuário do GC. É possível adquirir a URL para os usuários efetuarem login e receber o retorno através da variável de URL enviada pelo Google. Daí em diante, o aplicativo ou serviço autorizado pode tratar esses dados da maneira que precisar, como: gravar os dados do usuário no banco de dados ou abrir sessões.

Antes de poder acessar os dados do GC, o aplicativo ou serviço interessado precisa solicitar a autorização do usuário. Para autorizar a permissão, a Classroom API utiliza o padrão OAuth, um padrão aberto para identificação, autenticação e autorização utilizado para fazer logon em aplicativos que desejam fazer uso de APIs. O fluxo de como cada etapa da autenticação acontece é explicado a seguir:

1. Na página inicial do CodeTeacher, o usuário clica em um botão para fazer login e é direcionado para uma página de autenticação do Google.

2. Nessa página o usuário faz autenticação e o Google pergunta se ele quer fornecer seus dados pessoais ao CodeTeacher.

3. O CodeTeacher solicita as permissões específicas necessárias, como nome de usuário, endereço de e-mail e foto do perfil, e o usuário pode aprovar ou rejeitar a solicitação do serviço.

4. Depois que o usuário concorda, o CodeTeacher recebe os dados pessoais do usuário e o redireciona para a página de bem-vindo.

5. Ao final, o Google ID obtido e o email do usuário logado são gravados na base de dados do CodeTeacher.

Sendo assim, o CodeTeacher forneceu a interface e o conjunto de funcionalidades necessárias à avaliação e ao acompanhamento das atividades de programação. As demais funcionalidades típicas para gestão da aprendizagem, como criação de turmas virtuais, lançamento de comunicados, criação de atividades, recebimento de trabalhos, organização de material e intermediação da comunicação entre professor e aluno, foram providas

[11] https://developers.google.com/classroom

pelo GC, porém, de forma customizada, diretamente na interface do CodeTeacher. Outra vantagem também, é que não há necessidade de os usuários do CodeTeacher se cadastrarem para obter acesso, essa parte fica na responsabilidade do provedor de autenticação do Google. Dessa forma, verifica-se a complementaridade entre os ambientes integrados, valorizando o conjunto de competências peculiares a cada um.

4.5 MELHORIAS

Conforme descrito na introdução deste capítulo, a versão atual do CodeTeacher foi desenvolvida com fins de validar a proposta deste trabalho, cujo enfoque é a multiplicidade de estratégias de avaliação. Portanto, nesta abordagem inicial, as funcionalidades foram priorizadas de forma a contemplar as premissas e requisitos encontrados na literatura. Sendo assim, algumas funcionalidades foram deixadas para serem desenvolvidas posteriormente. A seguir, são descritas algumas caraterísticas a serem incorporadas nas versões futuras.

Suporte a outras linguagens de programação. O CodeTeacher é escrito e lida apenas com programas Java, porém, esse estilo de correção automatizada é adaptável a qualquer linguagem, desde que seja implementado o devido módulo responsável por tratar as especifidades da linguagem em questão. No caso da análise dinâmica, esse módulo precisa apenas compilar e executar o código escrito. Outra alternativa seria converter o código escrito em outra linguagem para um formato intermediário (bytecode) que possa ser executado na Máquina Virtual Java (JVM, Java Virtual Machine).

Estender a funcionalidade de cadastro de exercícios a fim de facilitar e agilizar o desenvolvimento da atividade pelo professor bem como implementar uma funcionalidade que facilite a identificação de plágios entre as soluções dos alunos. (SCHNEIDER; JAQUES, 2016).

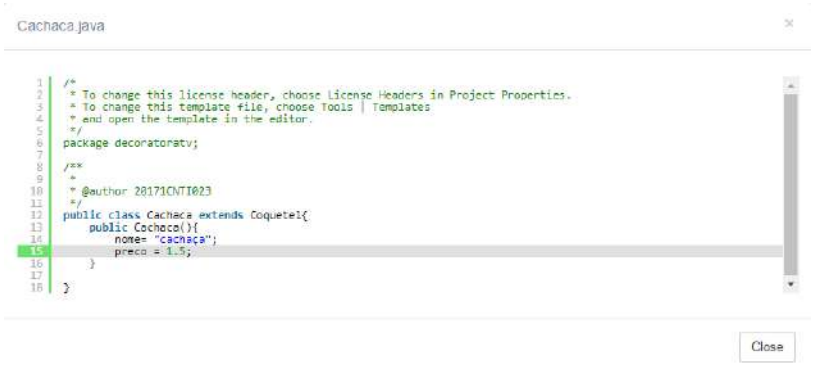
Alteração da estrutura do caso de teste para tornar o cadastro de dados de teste menos dependente da linguagem de programação utilizada e facilitar a execução de várias validações em cada caso de teste. (SCHNEIDER; JAQUES, 2016).

Outras melhorias são: **Geração automática de dados de teste** para a onerosidade da funcionalidade de cadastro de testes. **Recomendação de tópicos de estudo** para direcionar o aluno na resolução de suas dúvidas baseados nas deficiências encontradas na implementação do aluno. **Aplicação de conceitos de gamification** através de sistema de recompensa por badges e ranks. (DAGOSTINI et al., 2018). **Aperfeiçoamentos no quesito interface e usabilidade** e **Aplicação de políticas de segurança sobre a execução do código do aluno**. (SCHNEIDER; JAQUES, 2016).

Figura 4.8 – Exemplo de relatório de erros do CodeTeacher

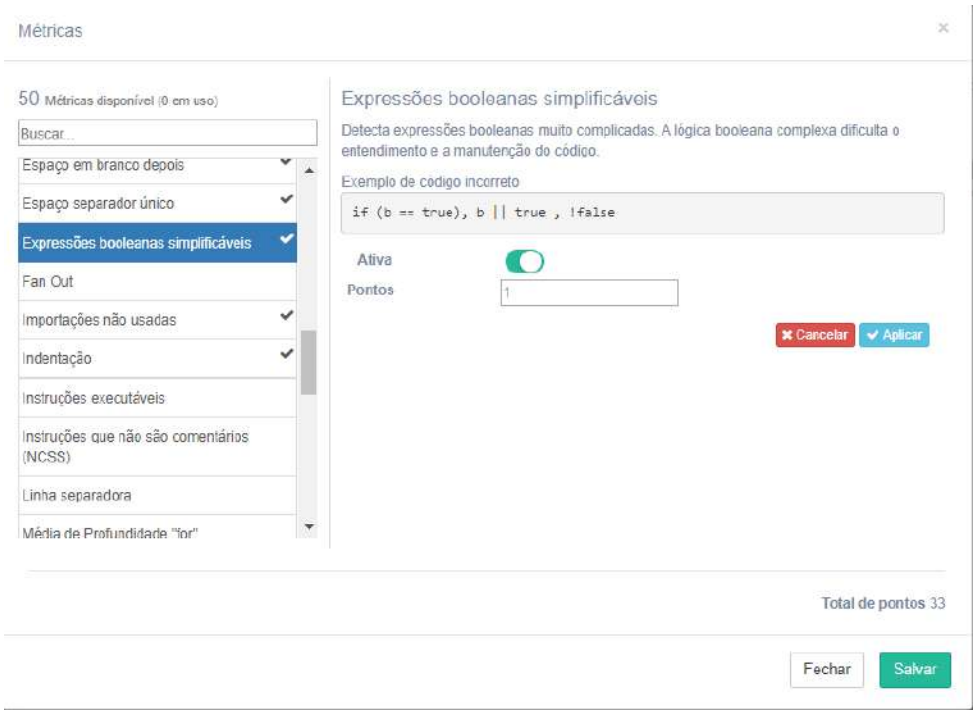
(a) Listagem de erros

(b) Indicação de linha de código problemática



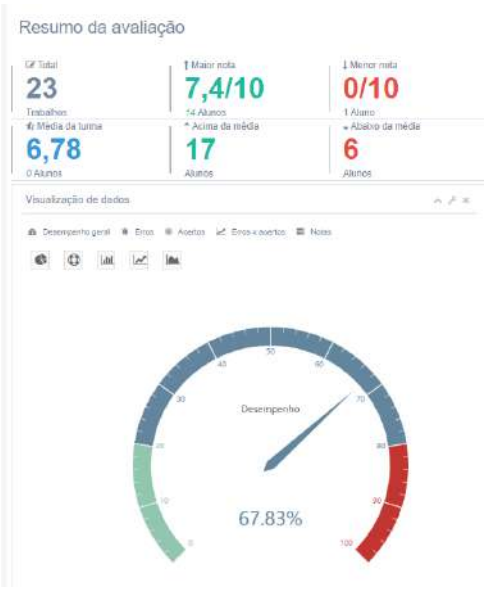
Fonte: Autor

Figura 4.9 – Configuração de critérios de métricas de código-fonte no CodeTeacher



Fonte: Autor

Figura 4.10 – Resultado geral de uma avaliação no CodeTeacher

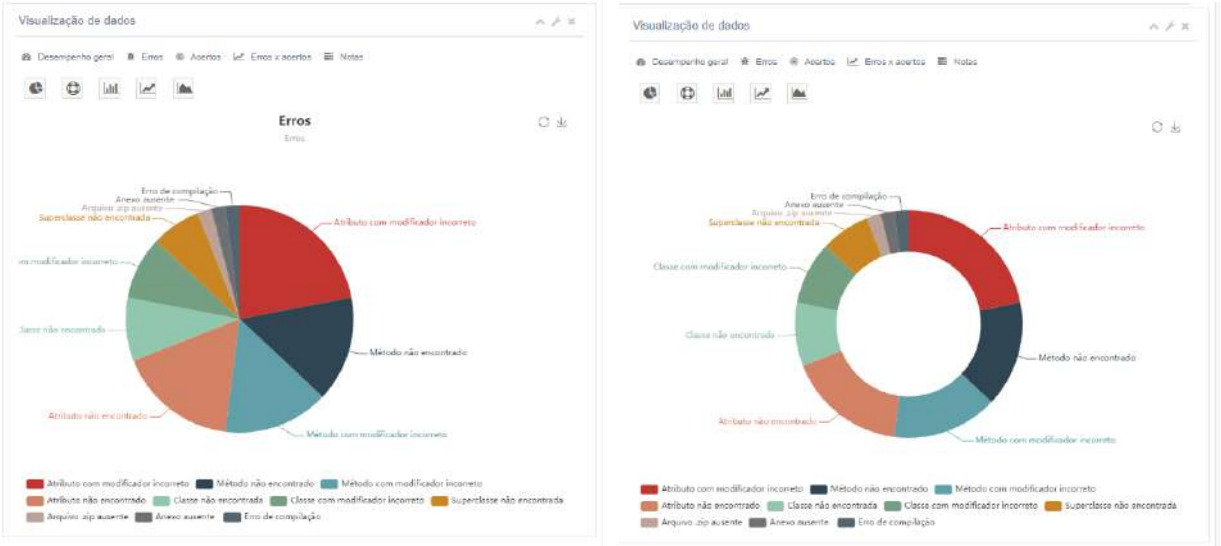


Fonte: Autor

Figura 4.11 – Visualização de erros

(a) Gráfico de pizza

(b) Gráfico de rosca

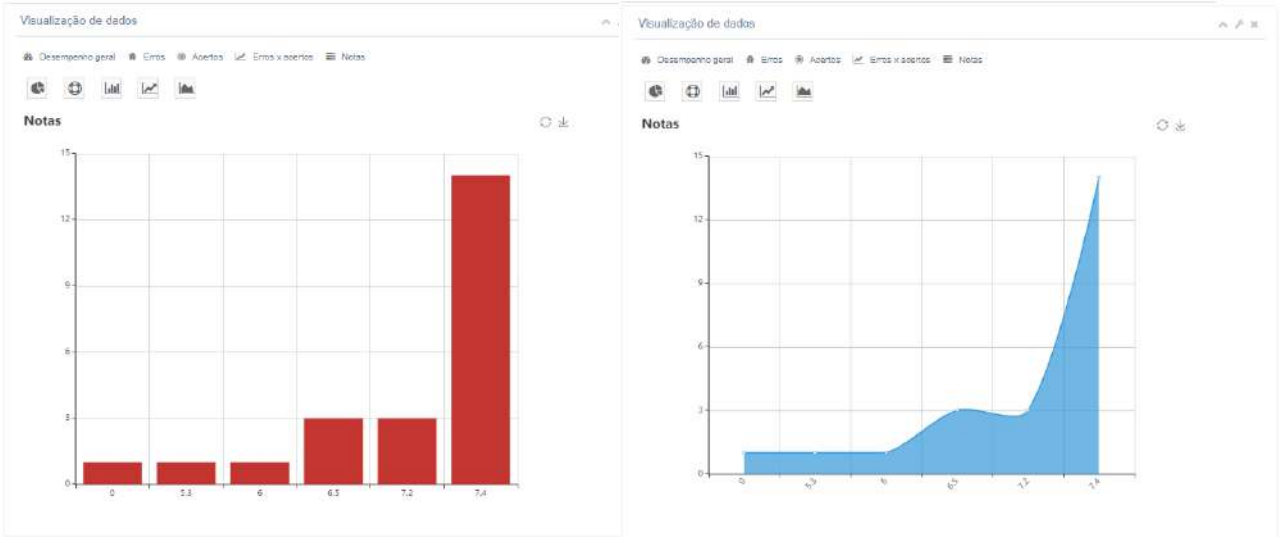


Fonte: Autor

Figura 4.12 – Visualização de notas

(a) Gráfico de barras

(b) Gráfico de área



Fonte: Autor

5 METODOLOGIA

Segundo MARCONI e LAKATOS (2017), “o Método Científico é o conjunto das atividades sistemáticas e racionais que, com maior segurança e economia, permite alcançar o objetivo – conhecimentos válidos e verdadeiros -, traçando o caminho a ser seguido, detectando erros e auxiliando as decisões do cientista.” Para Wazlawick (2014), em Computação, o método científico é importante por que a explicação dos dados é mais relevante do que a descrição da sua coleta, de forma que pesquisas nessa área não podem se ocupar apenas em descrever como os dados foram obtidos e processados, uma vez que a explicação dos dados possui maior valor científico.

Nesse sentido, este Capítulo contém os métodos e procedimentos adotados ao longo do desenvolvimento de todo este trabalho. O capítulo divide-se em três seções: A seção 5.1 apresenta a caracterização do trabalho do ponto de vista metodológico, identificando o tipo da pesquisa, bem como as opções metodológicas, de forma a categorizar este trabalho de acordo com critérios de classificação da pesquisa definidos na literatura, enquadrando-o de acordo com a sua natureza, objetivos e procedimentos técnicos. Na Seção 5.2 é detalhada a metodologia empregada no experimento conduzido neste estudo, apresentando uma análise sobre a investigação que foi realizada através da metodologia que foi aplicada. A Seção 5.3 apresenta os resultados coletados até o momento da escrita desta dissertação e discute as impressões do autor.

5.1 APRESENTAÇÃO DA METODOLOGIA

O foco deste trabalho está em instrumentalizar professores de programação com uma ferramenta de avaliação automática de códigos-fonte e verificar sua implicação na forma como os alunos aprendem. Desta forma, este trabalho se enquadra no grupo de trabalhos que Wazlawick (2014) classifica como trabalho original, pois busca "apresentar conhecimento novo a partir de observações e teorias construídas para explicá-las". A seguir, são relatadas as decisões metodológicas escolhidas como caminho para se chegar aos objetivos propostos.

Quanto à sua finalidade, a natureza deste trabalho caracteriza-se como pesquisa aplicada (WAZLAWICK, 2014), devido aos fins práticos do desenvolvimento do software pretendido. A pesquisa aplicada tem como objetivo a utilização de toda informação disponível para a criação de novas tecnologias e métodos, apontando para uma realidade que possui interesses específicos (PARANHOS; RODOLPHP, 2018). Esse tipo de pesquisa possui resultados palpáveis, buscando apresentar alternativas que ajudem a melhorar

ou transformar uma situação, fenômeno, sistema ou determinado aspecto do objeto (SCHWARTZMAN, 1979).

Quanto aos objetivos, uma pesquisa pode ser exploratória, descritiva ou explicativa (WAZLAWICK, 2014). No caso deste trabalho, a classificação fica bem caracterizada como uma pesquisa descritiva. Esse tipo de pesquisa pode ser entendida como um estudo de caso onde, após a coleta de dados, é realizada uma análise das relações entre as variáveis para uma posterior determinação dos efeitos resultantes em uma empresa, sistema de produção ou produto (PEROVANO, 2016). Por sua vez, a pesquisa também adquire um aspecto explicativo, pois além de registrar, analisar, classificar e interpretar os fenômenos sob investigação, o foco está na identificação de seus fatores determinantes (PRODANOV; FREITAS, 2013).

Como procedimentos metodológicos, pode-se citar a necessidade de pesquisa bibliográfica, uma vez que foi utilizado material já publicado, consistindo principalmente em livros e artigos científicos, para coletar os requisitos funcionais para desenvolvimento do software, através de revisões bibliográficas, análises, testes e estudos de ferramentas similares existentes, na busca e atribuição de conhecimentos sobre o uso do CodeTeacher como ferramenta de apoio à avaliação do desempenho estudantil, correlacionando o conhecimento adquirido com abordagens já trabalhadas por outros autores. Dentre as demais atividades realizadas durante o projeto, podem-se citar aquelas relativas ao desenvolvimento da aplicação, com a adoção de conceitos básicos da área de Engenharia de software (SOMMERVILLE, 2011).

Ainda quanto aos procedimentos técnicos, a necessidade de pesquisa experimental também foi detectada como parte da aplicação prática. Em particular, a pesquisa experimental consiste em submeter os objetos de estudo à influência de certas variáveis conhecidas pelo pesquisador e observar os resultados que as variáveis fornecem para o sujeito (GIL, 2008), em que caracteriza-se um conjunto de modelos de observação, descrição, investigação experimental e explanação teórica de fenômenos, fornecendo conhecimento que facilite a solução de problemas específicos (SILVA; MENEZES, 2005). Considerando que o ponto principal deste trabalho é identificar como a ferramenta impactou o desempenho dos estudantes, a pesquisa experimental se tornou a opção óbvia para o trabalho, frente à natureza da investigação.

A abordagem para o tratamento e coleta de dados do experimento foi quantitativa, pois demandou o uso de recursos e técnicas estatísticas, buscando traduzir em números os conhecimentos gerados pelo experimento. Para avaliar a efetividade da ferramenta desenvolvida, após o seu uso experimental por um grupo amostral representativo dos sujeitos pesquisados, os dados da pesquisa foram analisados com a medida por moda (Mo), assim como a média (μ).

Para organizar os dados coletados, foram utilizados recursos como índices, cálculos estatísticos e tabulação de dados. Os dados provenientes do experimento foram plotados em gráficos e tabelas para visualização dos resultados obtidos. Os gráficos gerados foram produzidos no próprio CodeTeacher, que foi usado como ferramenta de visualização da informação.

5.2 PLANEJAMENTO E EXECUÇÃO DO EXPERIMENTO

Esta seção descreve um experimento realizado para investigar o impacto do uso do CodeTeacher em um contexto real de EAPC. O interesse principal deste experimento foi atestar a efetividade do CodeTeacher como ferramenta de apoio ao processo de EAPC para suportar um método de avaliação que, em vez apoiar-se apenas na aplicação de testes para verificação da aprendizagem, seja baseada na análise de códigos-fonte para obtenção de um conjunto representativo de indicadores e, a partir destes, identificar características específicas do perfil de aprendizagem para reconhecimento automático mais preciso de dificuldades, habilidades e competências em programação.

O CodeTeacher foi aplicado experimentalmente em turmas de programação do Instituto Federal do Maranhão (IFMA) durante o segundo semestre do ano de 2019, para submissão de exercícios e posterior avaliação semi-automática dos códigos-fonte, levando em consideração padrões de correção definidos por professores. Para demonstrar a eficácia do uso do CodeTeacher nesse cenário, esta Seção descreve os experimentos realizados para avaliação semi-automática de exercícios de programação em linguagem Java, bem como os resultados alcançados.

O trabalho foi conduzido para validar uma das seguintes proposições:

H0 = CodeTeacher não auxilia aprendizes em programação a melhorarem a qualidade de seus códigos-fonte.

H1 = CodeTeacher auxilia aprendizes em programação a melhorarem a qualidade de seus códigos-fonte.

5.2.1 Características dos participantes

Neste experimento, o grupo de foco foi o curso de Técnico em Informática Integrado ao Ensino Médio. Na forma integrada, o aluno cursa o Ensino Médio juntamente com uma formação profissional. Os alunos desse curso compõem o universo desta pesquisa, pois:

1. É o único curso da instituição de ensino à qual o pesquisador é afiliado que oferta disciplinas de programação;
2. É um curso profissionalizante que visa a formação rápida e específica de programadores;
3. Propõe-se a formar profissionais aptos a atenderem as necessidades organizacionais, sendo direcionado a suprir as demandas do mercado local e mundial; e
4. O percurso formativo contempla um currículo multidisciplinar, agregando domínio técnico alinhado ao conhecimento de negócios.

Portanto, pelos motivos supracitados, este curso foi considerado adequado para a proposta deste trabalho. O CodeTeacher foi adotado como instrumento auxiliar de avaliação em duas turmas de uma disciplina de POO, na modalidade presencial. A disciplina centra-se no desenvolvimento de competências técnicas de programação sob o paradigma da Orientação a Objetos (HORSTMANN; CORNELL, 2000). Ao final da disciplina, o estudante deve ser capaz de implementar uma aplicação computacional num contexto de um sistema de informação, com base num determinado conjunto de especificações bem definidas. Ao todo, 80 alunos participaram do experimento, no qual cada um deles deveria enviar a resolução de problemas de programação previamente definidos e discutidos em sala de aula.

5.2.2 Pré-requisitos

Conforme visto nos capítulos anteriores, a grande maioria das pesquisas semelhantes envolve a aprendizagem introdutória, analisando trabalhos de estudantes neófitos. Devido à natureza e aos critérios avaliativos propostos nesta pesquisa, optou-se por trabalhar com alunos que, em tese, já possuem noções básicas de programação, e auxiliá-los na aquisição de novos conhecimentos, portanto, o foco deste experimento é averiguar não somente a funcionalidade dos programas desenvolvidos, como também a melhoria na qualidade dos códigos escritos, a partir da aplicação de boas práticas de programação.

5.2.3 Descrição dos dados

O conjunto de dados de entrada utilizado para o experimento foi constituído por:

- uma lista de competências;
- um grupo de indicadores;

- uma relação de critérios avaliativos;
- uma compilação de tópicos de ensino;
- um conjunto de enunciados de problemas de programação;
- uma coleção de modelos propostos como soluções de referência; e
- uma série de códigos-fonte submetidos por discentes como soluções aos enunciados, escritos na mesma linguagem de programação (linguagem Java).

O conjunto de enunciados de problemas de programação foi composto por três questões, propostos como atividades práticas. O conjunto estruturado de modelos de solução propostos como referência para resolução dos problemas, bem como o nível de dificuldade das questões foi fornecido por três especialistas, sendo dois professores e um tutor à distância. Foi elaborado um instrumento de medida ancorado em uma matriz com três competências, expressas em nove indicadores de aprendizagem. Cada um dos indicadores foi avaliado três vezes, gerando um conjunto de dados considerado adequado pela equipe executora do experimento para alcançar os resultados pretendidos.

O conjunto de dados de saída extraídos após a aplicação de cada avaliação foi formado por:

- um conjunto de valores de medições obtidas através da análise dos códigos-fonte;

O conjunto de códigos submetidos pelos discentes consistiu em um total de 228 soluções, sendo, em média, 76 respostas para para cada questão. Todas as soluções enviadas foram escritas utilizando a linguagem Java.

5.2.4 Instrumento de avaliação

Para garantir uma perspectiva pedagógica, as medidas de desempenho extraídas foram mapeadas para os critérios avaliativos das atividades elaboradas, estes critérios, por sua vez, foram relacionados com os objetivos de aprendizagem da disciplina, contidos na matriz curricular do curso em questão. Tais objetivos foram convertidos em competências. Foi elaborado um conjunto de competências a serem verificadas, com base no perfil do egresso contido no projeto do curso. É importante salientar que o perfil de competências apresentado neste trabalho não pode ser tomado como o perfil real de referência dos alunos que atuam nas turmas. Ele representa um perfil baseado na percepção de importância dada a cada competência pelos especialistas para as atividades práticas propostas. Não havendo, portanto, uma busca por precisão absoluta na analise da capacidade técnica de cada aluno.

Haja vista a natureza complexa de cada competência, para a sua avaliação, foram definidos indicadores que informam os parâmetros de sua evidência, aqui chamados de indicadores de competência, os quais representam desdobramentos dos objetivos próprios de cada uma dessas competências, de forma a permitir a emissão de juízo de valor sobre o alcance dos objetivos. Por sua vez, os tópicos de ensino, isto é, os itens de conteúdo programático da disciplina a serem ministrados e cobertos pela avaliação, também foram associados aos critérios avaliativos. Nesse sentido, a distribuição de critérios foi agrupada conforme demonstra a Tabela 5.1, mostrando a relação entre os critérios avaliativos e cada um dos indicadores e entre estes e as competências, de modo a permitir uma avaliação global do desempenho do participante e uma interpretação de desempenho em cada uma das três competências.

Tabela 5.1 – Elementos da avaliação

Competências	**Indicadores**	**Critérios**	**Tópicos de ensino**
Abstrair entidades do domínio do problema, representa-las através da implementação de classes e utilizar objetos no desenvolvimento de uma aplicação.	• Define a estrutura básica de uma classe • Mapeia os dados do problema para atributos de classes • Utiliza o conceito de Classes. • Define atributos e métodos	Criação de uma classe a partir do conceito de um modelo como abstração de uma realidade. Esta classe deverá contemplar, obrigatoriamente, os seguintes membros: - Três atributos (no mínimo). - Métodos setters e getters. - Método construtor padrão - Dois outros métodos construtores - Um método para exibir os valores dos atributos da classe.	Classes e Objetos Atributos e Métodos Mecanismo de sobrecarga
Implementar mecanismos de ocultamento de dados através dos conceitos de visibilidade, encapsulamento e pacotes	• Agrupa classes relacionadas através do uso de pacotes; • Controla o acesso aos elementos de uma classe por meio dos modificadores default, protected, public e private	Criação de um sistema para gerenciar os funcionários de uma empresa. O projeto deverá seguir os princípios da orientação a objetos. O sistema deve guardar dados dos funcionários, como salário e nome, essas propriedades de funcionário deverão ser mantidas em objetos com atributos para armazenar seus valores, onde cada objeto corresponderá a um funcionário com seus respectivos dados. A implementação do projeto deverá respeitar a regra de encapsulamento.	Visibilidade e escopo de atributos e métodos:
Reutilizar código através da Herança e aplicar os conceitos de Sobreposição	• Implementar uma hierarquia usando herança • Empregar o conceito de classes abstratas • Sobrepor métodos	• Criar uma hierarquia de classes com 3 (três) níveis (no mínimo) • Sobrepor, pelo menos, 2 (dois) métodos	Herança, polimorfismo e Sobreposição de métodos

Fonte: Autor

Nas questões, foram propostas situações-problema devidamente contextualizadas na realidade das turmas, em articulação com a interdisciplinaridade, nas quais os alunos participantes deveriam exerceu o papel de solucionador, considerando todas as possibilidades de interpretação da situação para produzir um programa que atendesse ao que se pedia. As situações-problema foram estruturadas de tal forma a impulsionar o participante a mobilizar conhecimentos anteriormente construídos e reorganizá-los para enfrentar o desafio proposto pela situação. As questões possuíam níveis de dificuldade equivalentes, sendo uma variação dos exercícios vistos durante as aulas e envolvendo novas construções de sintaxe, por exemplo, pedindo para criar um programa que calcule a área de um triângulo depois que eles aprenderem a calcular programaticamente a área de um retângulo.

5.2.5 Valores de referência

Buscando diminuir o grau de subjetividade do processo avaliativo, foram definidas menções para reforçar o desenvolvimento das competências, sendo assim, foram registradas menções parciais e totais de todos os indicadores previstos para cada competência, tendo por pauta o atendimento aos seus indicadores.

A qualificação do desempenho, para cada aluno, foi expressa em termos de Faixas de Desempenho, construídas em intervalos onde se localizam os totais de pontos obtidos (notas), conforme descrito na Tabela 5.2. O instrumento também permitiu que o desempenho em cada uma das três competências fosse igualmente representado numa escala de 0 a 100.

Tabela 5.2 – Faixas de Desempenho e Intervalos de Notas

Faixas de Desempenho	**Intervalos de notas**
Insuficiente a Regular	[0, 40)
Regular a Bom	[40, 70)
Bom a Excelente	[70, 100]

Fonte: Autor

Para cada critério avaliativo, foi estabelecida uma medida de referência, ou seja um valor limite para aferir a aderência ou não do aluno ao item requerido. A checagem do atingimento ou não do alvo foi medida pela comparação do valor obtido pelo aluno naquele item em comparação com o limiar definido, funcionando como ponto de corte para decidir se o aluno contemplou ou não o requisito especificado na atividade. Em alguns casos, o valor aceitável foi definido através da configuração de intervalos, nos quais, dependendo do critério, foram escolhidos os valores de maior consenso obtidos na literatura.

Cabe enfatizar que não há um padrão absoluto para cada indicador, a ênfase neste aspecto do trabalho está justamente na necessidade de permitir ao usuário configurar seus

próprios intervalos para contemplar a variedade de distribuições dos valores em contextos diferentes. Portanto, a configuração aqui construída não tem a intenção de propor valores de referência absolutos, mas servir como ponto de partida para posterior aperfeiçoamento e adaptação de acordo com as necessidades específicas de seus usuários. Portanto, para os fins deste trabalho, a proposta inicial de configuração foi elaborada sem muito rigor, a título ilustrativo, buscando generalizar os valores.

5.2.6 Procedimentos

O experimento foi dividido em três etapas distintas, a saber: instrução, simulação e execução. Cada aula ministrada nessas etapas teve a duração de 50 minutos. Em cada etapa foi realizada uma avaliação de aprendizagem, sendo que cada uma se deu em uma modalidade distinta, porém de forma interconectada, variando de acordo com a função de cada prática avaliativa, compreendendo a avaliação diagnóstica, formativa e somativa. Essas avaliações foram executadas de forma conjugada, em um processo cíclico.

Para verificar a habilidade do CodeTeacher em avaliar a aprendizagem dos alunos, estes foram orientados a submeterem seus programas à ferramenta em cada uma das etapas. As atividades propostas envolviam implementar classes em Java, aplicar mecanismos de herança e polimorfismo e escrever um método principal para servir como ponto de entrada do programa para chamar métodos entre objetos. Em todas as avaliações, o professor forneceu uma descrição detalhada da tarefa e, a partir desta descrição, escrita em linguagem natural, foi criada a base de critérios avaliativos. A especificação das tarefas continha todas as instruções necessárias para sua realização (como o nome da classe principal responsável pela execução do programa, as entradas e saídas esperadas, o formato de saída do programa, o padrão de codificação adotado e o conjunto de métricas de código-fonte a serem consideradas na avaliação). Essas especificações foram convertidas em um conjunto de estruturado de regras cadastradas no CodeTeacher que definiram estruturalmente todos os critérios avaliativos empregados. Todas as avaliações foram desenvolvidas em laboratório, de forma individual.

A etapa de instrução consistiu em prover os alunos das informações, conceitos, teorias e técnicas necessárias para realizar as atividades que seriam propostas nas avaliações práticas. Essa etapa foi conduzida ao longo de três semanas, cada uma composta de quatro aulas, em dois encontros por semana. Logo na primeira semana, ao se iniciarem as atividades pedagógicas, foi conduzida uma avaliação diagnóstica para investigar os conhecimentos prévios dos alunos e detectar o nível de domínio da turma em relação aos fundamentos da programação, a fim de identificar pontos fortes e fracos, bem como a diversidade entre os alunos quanto ao conhecimento prévio sobre aos conceitos básicos considerados como pré-requisitos para as competências a serem desenvolvidas. O diag-

nóstico resultante dessa investigação inicial demonstrou uma heterogeneidade no nível de conhecimento, o que demandou o ajuste da abordagem pedagógica inicialmente planejada às necessidades específicas observadas na turma, pois detectou-se a necessidade de realizar estratégias pedagógicas diversificadas, com vistas a suprir essa defasagem inicial e propiciar o incremento da aprendizagem.

Ainda na primeira semana, os alunos foram introduzidos aos conceitos básicos da Orientação a Objetos por meio de demonstrações de como escrever código de acordo com esse paradigma. Na segunda semana, conceitos básicos foram aplicados através de atividades práticas no formato de tutoriais para guiar os primeiros passos dos alunos rumo a uma solução pronta para um problema exemplo. Finalmente, na terceira semana, os estudantes foram desafiados a proporem suas próprias soluções para situações-problema e tiveram que escrever seus programas e submetê-los à avaliação automática no CodeTeacher. Ao longo desse processo, o CodeTeacher forneceu feedback automatizado, com possibilidade de campos para anotação dos feedbacks do docente e ciência do aluno, considerando tanto o nível de atingimento dos indicadores quanto o desenvolvimento global da competência. Nessa etapa, não houve restrição da quantidade de submissões por aluno, sendo assim, os estudantes puderam enviar suas respostas, receber o feedback, corrigi-las e submetê-las novamente, até julgarem ter cumprido todos os requisitos da tarefa. Os status das submissões de todos os alunos foram registrados, assim como a quantidade de submissões, para identificar quais alunos empreenderam maior esforço para resolver a atividade, considerando a quantidade de tentativas.

A etapa de simulação, por sua vez, objetivou explorar a aplicação dos conhecimentos adquiridos na etapa de instrução e preparar os alunos para a etapa de execução. Essa etapa foi realizada em uma semana. Nesse período, foi aplicada uma avaliação formativa para a sondagem do processo educativo, na qual os alunos tiveram a oportunidade de exercitar suas competências e aplicar os conhecimentos teóricos trabalhados em sala de aula, identificando, analisando e propondo soluções para problemas. Essa etapa teve como principal motivação o acompanhamento do desempenho dos alunos, no intuito de identificar aqueles que ainda não haviam atingido o esperado e, portanto, necessitariam de ações de recuperação.

A etapa de execução também foi conduzida em uma semana. Os alunos foram instruídos a escreverem programas similares aos desenvolvidos na etapa anterior, no intuito de atestar se as competências foram desenvolvidas a contento, bem como se os próprios objetivos de aprendizagem haviam sido atingidos. Foi então aplicada uma avaliação somativa, com vistas a identificar os alunos aprovados. O diferencial desta etapa foi que os alunos puderam submeter suas soluções uma única vez.

5.3 RESULTADOS E DISCUSSÃO

Esta seção apresenta os resultados obtidos após a realização das análises dos dados extraídos. A seguir são relatados os achados do estudo, juntamente com as respectivas inferências feitas pelo pesquisador, no intuito de discutir os resultados e contribuir com a divulgação e publicação das descobertas.

5.3.1 Interpretação dos valores

A Tabela 5.3 sintetiza resultado geral da avaliação das tarefas através do CodeTeacher.

Tabela 5.3 – Resumo da avaliação

Etapa	Número de programas	Corretos (%)	Incorretos (%)	Erros de compilação (%)
Instrução	76	67	23	10
Simulação	78	78	13	9
Execução	80	90	9	1

Fonte: Autor

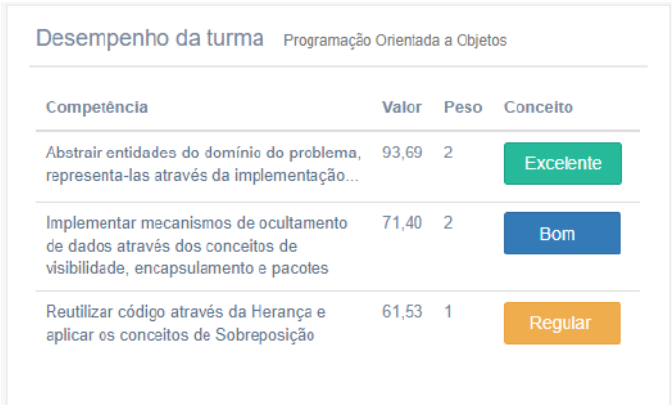
Na primeira etapa, dos 80 alunos, 4 não enviaram suas respostas, logo, foram avaliadas 76 tarefas, nas quais houve 10% de erro de compilação, 23% de programas incorretos e restante de 67% dos programas, como resultado correto do sistema. Essa primeira etapa foi a que apresentou a mais alta incidência de erros, o que demonstra dificuldades no inicio da aprendizagem, principalmente com relação à compreensão do que é requerido pelo exercício. Na segunda e terceira etapas, observou-se uma redução progressiva da quantidade de tarefas incorretas, o que sinaliza uma evolução dos alunos em resolver os problemas propostos. Na última etapa, por exemplo, o percentual de erro foi de apenas 1%.

As Figuras 5.1, 5.2 e 5.3 são imagens extraídas do próprio CodeTeacher e mostram os resultados do processamento, aplicado à configuração proposta e visualizada na ferramenta.

No CodeTeacher, o desempenho total da classe pode ser informado em vários formatos e uma dessas análises é fornecida na Figura 5.1, que mostra o desempenho geral da turma, obtida pela média de notas por trabalho, que é a média das notas associadas aos intervalos, ponderada pelos pesos dos critérios na configuração. Com isso, é possível demonstrar como interpretar os resultados das métricas do ponto de vista do professor, que conhece os requisitos gerais esperados da turma.

O feedback sobre o desempenho dos alunos nas diferentes avaliações também pode ser fornecido como vários gráficos. Uma dessas possibilidades é mostrada na Figura 5.2. Esta figura mostra a avaliação de dois dos alunos participaram das quatro avaliações, sendo que a avaliação zero corresponde à verificação diagnóstica feita no início da etapa 1,

Figura 5.1 – Competências da turma



Fonte: Autor

uma variação desse gráfico poderia incluir a comparação da evolução do desempenho de um aluno em relação ao histórico de progressão global da turma.

Figura 5.2 – Histórico comparativo de progressão

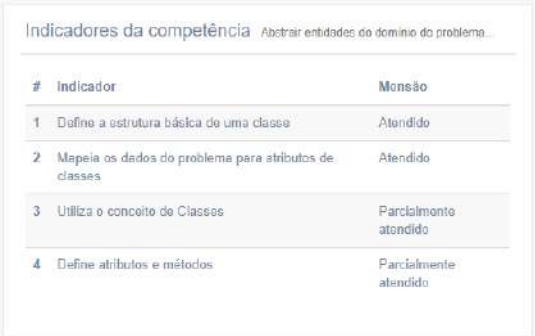
Fonte: Autor

A Figura 5.3 mostra os rótulos dos intervalos associados aos valores dos indicadores. Cada indicador foi marcado como "Atendido", "Parcialmente atendido"ou "Não atendido"e o cálculo da nota foi obtido a partir do percentual de itens atendidos. Os valores mostrados são as médias dos valores de cada medida para todas os critérios de cada indicador.

5.3.2 Análise estático-dinâmica

Nas análises estática e dinâmica, a maioria dos programas obteve a estrutura e o comportamento esperado. Daqueles que compilaram, mas não atenderam às especificações, a maioria dos erros foi relacionada à escrita de resultados no console da aplicação, devido

Figura 5.3 – Mensões dos indicadores



Fonte: Autor

às diferenças no texto impresso pelos alunos, principalmente relacionadas a problemas de acentuação, conforme a Tabela 5.4. Cabe ressaltar que apenas três alunos conseguiram contemplar todos os casos. Por tais motivos, esse conjunto de critérios não ofereceu os subsídios necessários para aferir o nível compreensão do tema por não ter proporcionado os resultados desejados.

Tabela 5.4 – Resumo da análise dinâmica

Etapa	**Entrada e Saída**	**Impressão na saída padrão**	**Acertos** (%)	**Erros** (%)
Instrução	77	53	71	29
Simulação	77	67	73	27
Execução	80	75	78	22

Fonte: Autor

5.3.3 Análise de convenções de código

Conforme pode ser observado na Tabela 5.5, na primeira etapa, a análise das convenções de código revelou baixa aderência no uso de padrões de nomenclatura e convenções de codificação da linguagem, isso indica que os alunos apresentaram dificuldade em seguir e aplicar boas práticas de programação. Com relação à estruturação e organização do código, foram detectadas várias ocorrências de problemas de indentação (43%), além de casos de espaços em branco em locais desnecessários, declarações não executáveis, blocos vazios, entre outros (35,8%). Altas taxas de erro envolvendo esse item demonstraram uma lacuna considerável no domínio de um tópico importante, fato que demandou uma intervenção no sentido de reduzir essa deficiência. Nas etapas seguintes, houve o melhoramento progressivo significativo desses indicadores, com o decréscimo perceptível dos valores a cada avaliação.

Tabela 5.5 – Resumo da avaliação das convenções de código

Critério	Instrução (%)	Simulação (%)	Execução (%)
Nomes de arquivos, classes, pacotes, variáveis, constantes e outros	7	4,3	1,2
Organização dos arquivos	6	13,1	14,6
Indentação	43	31,3	22,1
Comentários	23,4	11,3	1,7
Declarações de funções, variáveis e outras	19,3	11,6	3,7
Disposição das expressões	16,8	10,1	3,2
Política para espaços em branco e quebras de linha	35,8	15,3	11,5

Fonte: Autor

5.3.4 Análise por métricas de código-fonte

Como processo mental, escrever um programa implica estruturar o código produzido, independentemente da qualidade final. A complexidade dessa estrutura pode refletir o esforço cognitivo necessário para resolver a problema, dando indícios da sua dificuldade de resolução. Portanto, no tangente às métricas de código-fonte, foram extraídas métricas de esforço, dificuldade e complexidade.

5.3.4.1 Seleção de métricas

A seguir, são listadas as métricas consideradas relevantes para verificar a aprendizagem do aluno a partir de um código escrito.

A métrica Linhas de código que não são comentários (Non Commenting Source Statements - NCSS) determina a complexidade dos arquivos pela contagem das linhas de código executáveis (LEE, 2018). Para o cálculo do NCSS, não são consideradas apenas as instruções definidas na Especificação de Linguagem Java, também são incluídas todos os tipos de declarações. Em termos gerais, a métrica NCSS é obtida contando as linhas de código que não são comentários, de forma aproximadamente equivalente a contar os pontos-e-vírgulas e abertura de chaves (caracteres ";"e "{", respectivamente). Classes com NCSS muito grande são difíceis de ler e caras de manter. Um número NCSS grande geralmente significa que uma classe tem muitas responsabilidades e/ou funcionalidades que devem ser decompostas em unidades menores.

A métrica Complexidade Ciclomática (MCCABE, 1976) equivale basicamente ao número de condicionais. Em particular, a definição dessa métrica pode ser provada ao contar o número de desvios (ou estruturas condicionais) mais 1, ou seja, a complexidade é igual ao número de pontos de decisão somados de uma unidade para tornar o valor mais fácil de computar. Pontos de decisão são as seguintes estruturas: if, while, do, for, ?:, catch, switch e operadores && e || no corpo do código. Pela teoria, quando se trata de medir a qualidade do código por esse nível métrico, 10 é considerado um nível muito bom,

portanto, esse foi o limite especificado na configuração dessa métrica no critério definido no CodeTeacher.

A métrica NPATH (NEJMEH, 1988) calcula o número de possíveis caminhos de execução por meio de uma função (método). Ela leva em consideração o aninhamento de instruções condicionais e expressões booleanas de várias partes (A && B, C || D, E ? F :G e suas combinações). A métrica NPATH foi projetada para evitar problemas da métrica de complexidade ciclomática, como o nível de aninhamento dentro de uma função (método).

A métrica Complexidade de Expressões Booleanas (Boolean Expression Complexity) verifica o número de operadores booleanos (&&, ||, &, | e $\hat{}$) em uma expressão em relação a um limite especificado. Também restringe os operadores booleanos aninhados a uma profundidade especificada. Essa métrica se justifica pelo fato de que muitas condições levam a códigos difíceis de ler, depurar e manter.

5.3.4.2 Análise das métricas selecionadas

Na avaliação, foram identificados problemas de excesso de instruções, expressos no valor da métrica NCSS. Também foram detectadas estruturas condicionais em demasia e alta quantidade de comparações (Complexidade Ciclomática, Complexidade NPATH e Complexidade de Expressões Booleanas), além de problemas quanto ao uso correto de operadores e expressões lógicas, estruturas de controle condicional e de repetição. Esses achados demonstraram uma discrepância entre a dimensão do problema e a complexidade da solução, denunciando uma inabilidade dos alunos que obtiveram altos valores nessas métricas em criarem programas sucintos para resolverem problemas simples de programação, escrevendo mais código do que o necessário para chegar a uma solução.

Tabela 5.6 – Resumo da avaliação por métricas de código-fonte

Métrica	Instrução (%)	Simulação (%)	Execução (%)
Complexidade de Expressões Booleanas	13	13	1,3
Complexidade Ciclomática	35	23	17
Non Commenting Source Statements - NCSS	28	16	0
Complexidade NPath	31	22	7

Fonte: Autor

5.3.5 Ameaças à validade

As seguintes ameaças à validade estão presentes nesta pesquisa:

Quantidade de questões. Foram aplicadas apenas três questões para aplicar as avaliações. Acredita-se que uma maior quantidade de questões pode gerar resultados distintos daqueles aqui exibidos, pois, quanto mais procedimentos e instrumentos de avaliação forem planejados, maior a probabilidade de cobrir mais tópicos.

Quantidade de especialistas. Nesta pesquisa, a equipe docente foi composta de apenas três especialistas, tal quantidade pode não ser representativa, tendo em vista que o universo de especialistas é bastante vasto para o contexto do ensino de programação, podendo agregar profissionais da área da Pedagogia e Psicologia, por exemplo.

Intervalos de classificação. Os valores dos critérios usados em relação ao valor de atingimento dos indicadores obtidos podem não ter sido os intervalos ideais, de forma que outros intervalos podem gerar resultados distintos daqueles aqui exibidos;

Viés subjetivo na avaliação. A análise dos códigos dos discentes foi realizada em comparação com modelos soluções de referência. Devido às diversas possibilidades de evidência dos indicadores das competências, é possível que a adoção de mais soluções da gere resultados distintos daqueles aqui exibidos.

5.3.6 Conclusões

O conhecimento resultante das práticas avaliativas assistidas relatadas neste capítulo permitiu suportar duas importantes frentes para o desenvolvimento da aprendizagem: acompanhar o progresso do aluno e, ao mesmo tempo, orientar a ação educativa do docente. Na primeira frente, o sistema ajudou os alunos a assumirem um papel ativo em seu processo de aprendizagem, ao prover feedback adequado e clarificar os pontos fortes e o que ainda precisava ser alcançado, conduzindo os alunos ao pleno desenvolvimento das competências, em uma abordagem de autoestudo.

Na segunda frente, o sistema permitiu aos docentes concentrarem seus esforços na adequação entre o que foi planejado e a aplicabilidade de seu plano de trabalho, tendo em vista as características da turma, pois, sob o ponto de vista da otimização do trabalho do professor, este experimento forneceu o resultado da avaliação com menos esforço e forneceu as informações desejadas relacionadas à avaliação da programação. Embora ainda haja um certo esforço para configurar os critérios, este trabalho foi considerado pequeno, quando comparado ao esforço necessário para avaliar muitas tarefas sem uma ferramenta que possa automatizar esse processo.

Juntas, essas frentes impulsionaram o protagonismo do aluno como sujeito de seu processo educativo e aprimoraram a prática docente, na medida em que possibilitaram, a cada um desses atores, os subsídios para a reflexão sobre a sua própria atuação na cena pedagógica. Propiciando momentos de feedback em que docente e aluno puderam diagnosticar a aprendizagem já realizada e os pontos de dificuldades, e puderam juntos realizar correções de rumo ou adoção de novas estratégias para melhorar o desempenho do aluno no curso, de modo a reorganizarem a ação educativa e traçarem planos para atingir os objetivos.

Assim, a avaliação permeou todo o ato educativo, trazendo por centralidade o protagonismo do aluno e por premissa a objetividade e simplicidade na sua execução, facilitando para ambos a compreensão sobre o processo avaliativo, possibilitando um equilíbrio entre a formação massificada de alunos, e a necessidade de controle de evolução do aprendizado individual.

6 CONCLUSÕES

Este capítulo finaliza esta dissertação de mestrado e apresenta: as considerações finais da pesquisa; as limitações encontradas; as contribuições alcançadas; e as possibilidades de trabalhos futuros.

A Seção 6.1 traz uma discussão e análise mais aprofundada dos resultados experimentais, que responde às questões de pesquisa, visto que todos os experimentos foram conduzidos a fim de responder tais questões.

A Seção 6.2 contém os principais aspectos e contribuições de forma a finalizar o trabalho apresentado, recapitulando o que era esperado do trabalho através dos objetivos inseridos inicialmente e relatando o que foi conseguido.

A Seção 6.3 apresenta as limitações identificadas e problemas que surgiram durante o desenvolvimento do trabalho e quais as consequências do mesmo.

A Seção 6.4 apresenta as perspectivas futuras, que representam oportunidades de expansão do trabalho apresentado, bem como, novos projetos que puderam ser vislumbrados a partir do desenvolvimento da pesquisa.

6.1 CONSIDERAÇÕES FINAIS

Este trabalho apresentou uma abordagem para auxiliar o EAPC utilizando um SAA como ferramenta de apoio pedagógico. O SAA utilizado, denominado CodeTeacher, foi desenvolvido especificamente para esse propósito. Este sistema compila e executa automaticamente programas submetidos por alunos e avalia-os com base no esquema de verificação definido. Ao ser desenvolvido um SAA, e através de sua disponibilização para o público em geral, espera-se que com este trabalho haja uma contribuição efetiva para o EAPC e que o sistema seja aproveitado pelas pessoas interessadas, gerando assim, mais casos de uso que venham a contribuir para o amadurecimento da proposta.

A utilização do CodeTeacher se aplica a quaisquer contextos de ensino-aprendizagem de programação em Java, desde os níveis iniciais até os níveis superior e de pós-graduação. Como potenciais beneficiários da solução tecnológica aqui proposta, podem-se citar professores e alunos de todos os níveis da área de Computação. O sistema pode ser adotado por instituições de ensino, públicas ou privadas (universidades, institutos, faculdades, escolas, entre outros) como instrumento auxiliar no processo de ensino-aprendizagem de programação em seus cursos de Computação, podendo usufruir das vantagens de proporcionar experiências inovadoras de aprendizagem aos seus alunos. Também podem ser público-alvo

empresas de treinamento que ofertem cursos de programação nas modalidades à distância e presencial, bem como empresas de recrutamento para a aplicação de testes seletivos online.

6.1.1 Verificação da Hipótese de Pesquisa

Para responder a Questão de Pesquisa **RQ 1** (Quais as maiores limitações dos principais SAAs?), foram discutidas as características principais dos SAAs no Capítulo 3, destacando-se a importância do feedback, ou seja, a devolutiva da submissão com um relatório de desempenho suficientemente informativo, assim como a necessidade de versatilidade na avaliação, além de requisitos como segurança e confiabilidade.

Para responder à **RQ 2** (Qual a eficácia da avaliação por análise automática de código-fonte?), foram apresentadas no Capítulo 2, os fatores superlativos das estratégias mais empregadas na correção de códigos-fonte de alunos. Com isso, foi possível constatar a complementaridade das abordagens utilizadas para alcançar uma avaliação mais abrangente e flexível, visto que nenhuma das soluções analisadas se mostrou aderente a todos os itens elencados como necessários para uma avaliação efetiva.

Para responder à **RQ 3** (Quais os benefícios da combinação proposta?), foram descritos no Capítulo 5 os procedimentos para realização de um experimento, no qual o SAA proposto foi adotado experimentalmente em uma turma de computação com base em conceitos básicos de Programação Orientada a Objetos. Havia 80 alunos em sala de aula que enviaram suas tarefas. O experimento foi considerado satisfatório ao demonstrar uma melhoria na capacidade de detectar deficiências de aprendizagem.

Para responder à **RQ 4** (Quais as limitações do uso concomitante das estratégias adotadas?), através do experimento relatado no Capítulo 5, foi possível notar algumas restrições, como o fato do aluno ter que escrever um programa que siga a nomenclatura de classes, atributos e métodos e cujos testes estejam restritos a um número limitado de pares de entrada e saída esperados, além de que a saída impressa deve estar no formato pré-definido. No futuro, essas restrições podem ser removidas para tornar o sistema mais versátil. Uma outra limitação é o fato de a ferramenta estar restrita às configurações do professor e não ser capaz de fazer verificações extras. Por exemplo, em uma análise manual podem ser encontrados projetos de alunos com arquivos desnecessários, pois alguns trabalhos podem conter classes pertencentes a outras atividades. Tais arquivos, embora não comprometam o funcionamento do programa, apontam uma falta de cuidado na organização do projeto, contudo, a verificação automática não encontrou esses arquivos e, portanto, passariam despercebidas pela análise. Para contrapor essa deficiência, métricas de tamanho, com Linhas de Código, Tamanho de Arquivo, Quantidade de classes, entre outras podem ser usadas de forma combinada para ajudar a contornar essa limitação.

6.2 CONTRIBUIÇÕES ALCANÇADAS

A principal contribuição desta dissertação está no desenvolvimento de um SAA inovador, com um conjunto de ferramentas para a automação da avaliação de códigos-fonte, com ênfase nos estudos e na seleção de critérios, o que permite a análise de código-fonte de acordo com a percepção de qualidade da indústria de software, através de um método de avaliação semi-automática, sem a necessidade de investir grandes esforços na concepção e correção de exercícios. Este resultado permitiu alcançar o objetivo geral estabelecido no Capítulo 1, a saber: Alcançar potencialidades de coleta automática de dados relativos ao desempenho discente em programação e extrair informações suficientes para compor uma nota final, para subsidiar uma avaliação automática da aprendizagem de programação de computadores.

Além dessa, são também contribuições desta pesquisa:

1. Revisão bibliográfica sobre as abordagens de correção automática de código-fonte, onde foi realizada uma análise comparativa dos diferentes métodos e ferramentas propostos pela literatura. Este resultado foi perseguido no intuito de contemplar o objetivo específico: "Fazer um levantamento do estado da arte acerca dos sistemas, ferramentas e técnicas empregados na análise de código-fonte para suportar o EAPC", elencado na Seção 1.5, Subseção 1.5.2.

2. A avaliação da proposta utilizando um cenário real de EAPC para detectar problemas de aprendizagem e identificar as partes mais críticas de uma atividade avaliativa. Essa realização foi conseguida no intento de alcançar o objetivo específico: "Realizar análises utilizando as informações extraídas e usar ferramentas de visualização de informação para comparar soluções de programação em detalhes".

3. Uma análise detalhada, em relação ao comportamento e estrutura dos códigos dos programas escritos pelos estudantes, o que representa um avanço em comparação a literatura relacionada, ao ampliar o número de estratégias de avaliação consideradas e propor uma abordagem que visa diminuir as contradições das análises dos critérios. Através desse tipo de abordagem, torna-se possível auxiliar o trabalho de avaliação de professores de programação para ajudá-los a compreenderem as dificuldades de aprendizagem de seus alunos. O objetivo específico que norteou esta busca foi: "Produzir relatórios que ajudem a detectar deficiências de aprendizagem para uma abordagem personalizada, oportunizando maior clareza sobre o rendimento discente individual e coletivo".

4. Elaboração de um método quantitativo que utiliza análise semi-automática de código como suporte para detecção de dificuldades de aprendizagem de programação, com

o uso de critérios de interesse consideradas úteis nos experimentos realizados com participantes, de acordo com o que foi especificado no objetivo específico: "Propor um instrumento de apoio à avaliação diagnóstica, formativa e somativa da aprendizagem de alunos na prática da programação de computadores".

Cabe ressaltar que este projeto produziu um fruto que permitiu alcançar o resultado de contribuir cientificamente, dando origem à publicação de um artigo científico aceito como short paper em uma conferência internacional com extrato qualis B1. O artigo, intitulado “Collection and Analysis of Source Code Metrics for Composition of Programming Learning Profiles”, foi apresentado no 18th ICALT 2019 (IEEE International Conference on Advanced Learning Technologies). O ICALT é uma conferência internacional anual sobre aprendizagem avançada em tecnologias de aprendizagem e aprimoramento tecnológico, organizada pela IEEE Computer Society e pelo IEEE Technical Committee on Learning Technology, que no ano de 2019 foi realizada em Maceió (AL), entre os dias 15 e 18 de julho. Também há uma outra publicação em andamento, em fase de submissão na Revista Novas Tecnologias em Educação (RENOTE), periódico com qualis A4 (extrato 2019), ISSN: 1679-1916, no qual os resultados parciais desta dissertação são relatados e discutidos.

Como produto direto desta pesquisa, foi efetuada a solicitação junto ao Instituto Nacional da Propriedade Industrial (INPI) de um processo requerendo a entrada de Registro de Programa de Computador, protocolado com a assessoria do Núcleo de Inovação Tecnológica da Universidade Estadual do Maranhão (NIT-UEMA), sob o número de processo: 512020000110-6. Caracterizando assim este trabalho como um projeto de inovação, cujo software produzido poderá, com um investimento razoável, ser lançado no mercado para ser explorado comercialmente.

6.3 LIMITAÇÕES

Devido ao grande número de ferramentas e serviços de apoio à aprendizagem de programação disponíveis e às restrições de tempo, tornou-se impraticável realizar uma descrição detalhada de todas. Portanto, não foi possível incluir no Capítulo 3 análises de outras ferramentas e serviços relacionados, agregando outras comparações, que talvez fossem interessantes para dar um direcionamento diferente ao trabalho.

Com relação ao experimento realizado, por mais que este tenha sido desenhado para ser generalizável, os testes foram realizados com dados de um contexto específico, com características peculiares de um período, conteúdos, turmas e estratégias de ensino particulares. Em outros contextos poderão ser usadas didáticas distintas, com uso de materiais diferentes, por exemplo. Assim, há a possibilidade de haver a necessidade de refazer o processo de avaliação para esses contextos específicos.

Uma outra limitação está na escolha dos critérios e na seleção de configuração dos itens avaliativos, pois a relação entre o valor das medições e a aprendizagem dos alunos necessita de aperfeiçoamento para uma análise mais precisa, pois este aspecto também está sujeito às particularidades do ambiente de ensino-aprendizagem, de forma que novos trabalhos devem ser conduzidos nesse sentido.

6.4 TRABALHOS FUTUROS

Como trabalhos futuros, pretende-se proceder na condução de novos estudos de caso para atestar a viabilidade desta abordagem, bem como validar seu funcionamento. Os relatos destas experiências poderão ser publicados para que, dessa forma, também haja contribuição como fonte bibliográfica para os pesquisadores de EAPC.

Espera-se, portanto, que em um futuro próximo, o CodeTeacher possa ser instrumento de trabalho de professores, auxiliando-os na compreensão das dificuldades de aprendizagem de seus alunos e na tomada de decisões de avaliação que reorientem as ações de ensino em favor da aprendizagem.

Novas questões de pesquisa surgiram após a conclusão da investigação apresentada neste trabalho. Uma possibilidade de investigação se dá na adoção de uma maior quantidade de critérios de avaliação. Por fim, avaliações envolvendo um número maior de alunos e outros tipos de conteúdos programáticos também deverão ser conduzidas.

REFERÊNCIAS

A., T. N.; LUCA, B. J. Uri online judge: a new classroom tool for interactive learning. In: FECS. *Proceedings at WORLDCOMP'12*. [S.l.], 2012. p. 242–246. Citado na página 38.

ALA-MUTKA, K. A survey of automated assessment approaches for programming assignments. *Computer Science Education*, v. 15, p. 83–102, 06 2005. Citado 5 vezes nas páginas 18, 23, 24, 25 e 26.

ALMEIDA, E. et al. Ambap: Um ambiente de apoio ao aprendizado de programação. 02 2020. Citado na página 23.

ALVES, F.; JAQUES, P. Um ambiente virtual com feedback personalizado para apoio a disciplinas de programação. In: *XXV Simpósio Brasileiro de Informática na Educação (SBIE 2014)*. [S.l.]: Sociedade Brasileira de Computação- SBC, 2014. Citado 5 vezes nas páginas 31, 33, 35, 36 e 43.

AURELIANO, V. C. O.; TEDESCO, P. C. d. A. R. Ensino-aprendizagem de programação para iniciantes: uma revisão sistemática da literatura focada no sbie e wie. SBC, 2012. Citado na página 22.

BARBOSA, A. et al. Um mapeamento sistemático sobre analisadores de código em disciplinas de programação. In: *IV Congresso Brasileiro de Informática na Educação*. [S.l.: s.n.], 2015. p. 1235. Citado na página 31.

BARBOSA, E. et al. Integrated teaching of programming foundations and software testing. In: . [S.l.: s.n.], 2008. p. S1H–5. Citado na página 32.

BEZ, J. L.; FERREIRA, C.; TONIN, N. Uri online judge academic: A tool for professors. 08 2013. Citado na página 38.

BEZ, J. L.; TONIN, N.; RODEGHERI, P. Uri online judge academic: A tool for algorithms and programming classes. In: THE 9TH INTERNATIONAL CONFERENCE ON COMPUTER SCIENCE EDUCATION (ICCSE 2014). [S.l.]: Iccse, 2014. p. 149–152. Citado na página 38.

BEZ, J. L.; TONIN, N.; SELIVON, M. Uri online judge academic: Integração e consolidação da ferramenta no processo de ensino/aprendizagem. In: *XIV Workshop de Educação em Computação (WEI)-SBC 2015*. Recife, PE: Sociedade Brasileira de Computação - SBC, 2015. Citado na página 38.

BRASSCOM. *Relatório Setorial de TIC 2019*. [S.l.], 2019. Disponível em: <https://brasscom.org.br/relatorio-setorial-de-tic-2019/>. Acesso em: 05 jan. 2020. Citado na página 14.

BUYRUKOGLU, S.; BATMAZ, F.; LOCK, R. Semi-automatic assessment approach to programming code for novice students. In: . [S.l.: s.n.], 2016. Citado na página 24.

BUYRUKOGLU, S.; BATMAZ, F.; LOCK, R. Improving marking efficiency for longer programming solutions based on a semi-automated assessment approach. *Computer Applications in Engineering Education*, v. 27, 01 2019. Citado 4 vezes nas páginas 16, 19, 24 e 43.

CAIZA, J.; ALAMO, J. D. Programming assignments automatic grading: Review of tools and implementations. In: . [S.l.: s.n.], 2013. p. 5691–5700. ISBN 9788461626618. Citado na página 43.

CAMPOS, C.; FERREIRA, C. Boca: um sistema de apoio para competiç oes de programação. In: XII WORKSHOP DE EDUCAÇÃO EM COMPUTAÇÃO (WEI), 12., 2004, Salvador, BA. *Anais do Congresso da SBC*. Salvador, BA: SBC, 2004. Citado 3 vezes nas páginas 32, 33 e 34.

CARTER, J. et al. How shall we assess this? *SIGCSE Bulletin*, v. 35, p. 107–123, 12 2003. Citado na página 24.

CARVALHO, L.; FERNANDES, D.; GADELHA, B. Juiz online como ferramenta de apoio a uma metodologia de ensino híbrido em programação. In: *Anais do XXVII Simpósio Brasileiro de Informática na Educação (SBIE 2016)*. Uberlândia/MG: SBC, 2016. p. 140. Citado na página 38.

CASTRO, T.; FUKS, H. Sistematização da aprendizagem de programação em grupo. In: WORKSHOP DE INFORMÁTICA NA ESCOLA (WIE), 22., 2011, Aracaju, SE. *Anais do Simpósio Brasileiro de Informática na Educação - SBIE*. Aracaju, SE: SBC, 2011. ISSN 2176-4301. Citado na página 22.

CASTRO, T. H. et al. Utilizando programação funcional em disciplinas introdutórias de computação. In: *Workshop de Educação em Computação (WEI)-SBC 2002*. Florianópolis: Sociedade Brasileira de Computação - SBC, 2002. Citado na página 14.

CHAVES, J. et al. Uma ferramenta baseada em juízes online para apoio às atividades de programação de computadores no moodle. *RENOTE*, v. 11, 12 2013. Citado 3 vezes nas páginas 31, 33 e 35.

CHAVES, J. et al. Mojo: Uma ferramenta para integrar juízes online ao moodle no apoio ao ensino e aprendizagem de programação. *HOLOS*, v. 5, p. 246, 01 2014. Citado na página 16.

CHAVES, J. O. et al. Mojo: Uma ferramenta de auxílio à elaboração, submissão e correção de atividades em disciplinas de programação. In: WORKSHOP DE EDUCAÇÃO EM COMPUTAÇÃO (WEI), 23., 2013, Maceió, AL. *Anais do Congresso da Sociedade Brasileira de Computação*: Cidades inteligentes: Desafios para a computação. Maceió, AL: SBC, 2013. Citado na página 33.

CHECKSTYLE. *CHECKSTYLE - code style analysis tool for Java*. 2016. Disponível em: <http://checkstyle.sourceforge.net>. Acesso em: 31 out. 2019. Citado na página 57.

CHEN, H.-M.; CHEN, W.-H.; LEE, C.-C. An automated assessment system for analysis of coding convention violations in java programming assignments*. *Journal of Information Science and Engineering*, v. 34, p. 1203–1221, 09 2018. Citado 3 vezes nas páginas 16, 19 e 28.

DAGOSTINI, J. et al. Uri online judge blocks: Construindo soluç oes em uma plataforma online de programação. In: . [S.l.: s.n.], 2018. p. 168. Citado 3 vezes nas páginas 32, 38 e 62.

DAGOSTINI, J. et al. Incentivando a aprendizagem de algoritmos através do uri online judge forum 2.0. In: . [S.l.: s.n.], 2017. Citado na página 38.

EBRAHIMI, A. Novice programmer errors: language constructs and plan composition. *International Journal of Human-Computer Studies*, v. 41, n. 4, p. 457 – 480, 1994. ISSN 1071-5819. Disponível em: <http://www.sciencedirect.com/science/article/pii/S107158198471069X>. Citado na página 16.

ELNAFFAR, S. Using software metrics to predict the difficulty of code writing questions. In: . [S.l.: s.n.], 2016. p. 513–518. Citado na página 16.

FARIAS, F.; NUNES, I. Aprendizagem ativa no ensino de programação: Uma revisão sistemática da literatura. In: VIII CONGRESSO BRASILEIRO DE INFORMÁTICA NA EDUCAÇÃO (CBIE 2019), Brasília, DF. *Anais dos Workshops do VIII Congresso Brasileiro de Informática na Educação (WCBIE 2019)*. Brasília, DF, 2019. p. 377. Citado na página 41.

FEEPER. *FEEPER: Uma ambiente web para apoio ao ensino de programação*. 2020. Disponível em: <http://feeper.unisinos.br/>. Acesso em: 30 de jan. de 2020. Citado na página 36.

FENTON, N.; PFLEEGER, S. *Software Metrics: A Rigorous Practical Approach*. [S.l.: s.n.], 1998. XII. ISBN 0534954291. Citado na página 30.

FERREIRA, F. Z. et al. El o e tri: Estimando a habilidade dos estudantes em uma plataforma online de programação. *RENOTE*, v. 17, p. 11–20, 07 2019. Citado na página 18.

FILHO, R. et al. A evasão no ensino superior brasileiro. *Cadernos De Pesquisa*, v. 37, p. 641–659, 12 2007. Citado na página 15.

FONSECA, S. et al. Adaptação de um método preditivo para inferir o desempenho de alunos de programação. In: . [S.l.: s.n.], 2019. Citado 3 vezes nas páginas 14, 22 e 23.

FRANÇA, A. et al. Um sistema orientado a serviços para suporte a atividades de laboratório em disciplinas de técnicas de programação com integração ao ambiente moodle. *RENOTE*, v. 9, 07 2011. Citado 5 vezes nas páginas 14, 15, 33, 34 e 41.

FRANÇA, A.; SOARES, J. M. Sistema de apoio a atividades de laboratório de programação via moodle com suporte ao balanceamento de carga. In: *XXII Simpósio Brasileiro de Informática na Educação*. Aracaju-SE: SBC, 2011. Citado na página 34.

FRANÇA, A. B.; SOARES, J. Sistema de apoio a atividades de laboratório de programação via moodle com suporte ao balanceamento de carga. *Revista Brasileira de Informática na Educação*, v. 21, 08 2013. Citado 2 vezes nas páginas 34 e 39.

GIL, A. C. *Metodos e Técnicas de Pesquisa Social*. 6. ed. São Paulo: Atlas, 2008. Citado na página 67.

GIRAFFA, L.; MORAES, M.; UDEN, L. Teaching object-oriented programming in first-year undergraduate courses supported by virtual classrooms. In: _____. *Springer Proceedings in Complexity*. [S.l.: s.n.], 2014. p. 15–26. Citado na página 15.

GOOGLE. *Google Java style guide*. 2019. Acesso em: 31 out. 2019. Disponível em: <https://google.github.io/styleguide/javaguide.html>. Citado na página 28.

GOOGLE. *Classroom*. 2020. Disponível em: <https://classroom.google.com/>. Acesso em: 13 fev. 2020. Citado na página 45.

GOOGLE. *Google For Education*. 2020. Acesso em: 11 fev. 2020. Disponível em: <https://edu.google.com/>. Citado na página 60.

GUPTA, S.; GUPTA, A. E-assessment tools for programming languages: A review. In: *ICITKM*. [S.l.: s.n.], 2018. p. 65–70. Citado na página 24.

HARMAN, M. Why source code analysis and manipulation will always be important. In: *10th IEEE International Working Conference on Source Code Analysis and Manipulation (SCAM 2010)*. [S.l.: s.n.], 2010. p. 12. Citado na página 30.

HORSTMANN, C. S.; CORNELL, G. Core java 2. vol.1: Fundamentos. In: _____. [S.l.]: Makron Books, 2000. Citado 2 vezes nas páginas 54 e 69.

HUXLEY, T. *Huxley*. 2020. Disponível em: <https://www.thehuxley.com/>. Acesso em: 31 de jan. de 2020. Citado na página 37.

IHANTOLA, P. et al. Review of recent systems for automatic assessment of programming assignments. In: *Proceedings of the 10th Koli Calling International Conference on Computing Education Research, Koli Calling'10*. New York, NY, USA: ACM, 2010. p. 86–93. ISBN 978-1-4503-0520-4. Citado 6 vezes nas páginas 31, 40, 41, 42, 43 e 48.

INSA, D.; SILVA, J. Semi-automatic assessment of unrestrained java code: A library, a dsl, and a workbench to assess exams and exercises. In: *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education*. New York, NY, USA: ACM, 2015. (ITiCSE '15), p. 39–44. ISBN 978-1-4503-3440-2. Disponível em: <http://doi.acm.org/10.1145/2729094.2742615>. Citado 5 vezes nas páginas 19, 24, 41, 43 e 46.

INSA, D.; SILVA, J. Automatic assessment of java code. *Computer Languages, Systems Structures*, v. 53, p. 59 – 72, 2018. ISSN 1477-8424. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1477842417301045>. Citado 2 vezes nas páginas 41 e 43.

ISO/IEC 14598-1. *Software Engineering - Product evaluation – Part 1: General, International Organization for Standarization*. [S.l.], 2001. Citado na página 29.

ISO/IEC 25000. *Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — Guide to SQuaRE*. [S.l.], 2014. Citado 2 vezes nas páginas 19 e 29.

ISO/IEC 25021. *Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — Quality measure elements*. [S.l.], 2012. Citado 2 vezes nas páginas 29 e 30.

ISO/IEC 9126-1. *Software engineering – product quality – part 1: Quality model*. [S.l.], 2001. Citado na página 29.

JACKSON, D. Semi-automated approach to online assessment. In: . [S.l.: s.n.], 2000. v. 32, p. 164–167. Citado na página 24.

JARGAS, A. *EXPRESSOES REGULARES: UMA ABORDAGEM DIVERTIDA*. 5. ed. NOVATEC, 2016. Disponível em: <https://books.google.com.br/books?id=12tA2_TUlIgC>. Citado na página 46.

JESUS, G. et al. Análise dos erros mais comuns de aprendizes de programação que utilizam a linguagem python. In: . [S.l.: s.n.], 2018. p. 1751. Citado 4 vezes nas páginas 15, 22, 23 e 37.

JESUS, G. et al. Avaliação de uma abordagem para auxiliar a correção de erros de aprendizes de programação. In: . [S.l.: s.n.], 2018. p. 1. Citado na página 37.

JESUS, G. et al. Análise dos erros mais comuns de aprendizes de programação que utilizam a linguagem python. In: COMPUTER ON THE BEACH, 2019, Florianópolis, SC. *Anais do Computer on the Beach 2019*. Florianópolis, SC, 2019. p. 406. Citado 2 vezes nas páginas 22 e 37.

JESUS, G. S. et al. Code umpire: uma abordagem para avaliação automática e unificada de restriç oes em código-fonte de aprendizes de programação. In: COMPUTER ON THE BEACH, 2019, Florianópolis, SC. *Anais do Computer on the Beach 2019*. Florianópolis, SC, 2019. p. 307. Citado 3 vezes nas páginas 16, 25 e 37.

JESUS, G. S. et al. Eme: Uma ferramenta para auxiliar a correção de erros de aprendizes de programação. In: COMPUTER ON THE BEACH, 2019, Florianópolis, SC. *Anais do Computer on the Beach 2019*. Florianópolis, SC, 2019. p. 287. Citado na página 37.

JUDGE, O. *Online Judge*. 2020. Disponível em: <https://onlinejudge.org/>. Acesso em: 13 fev. 2020. Citado na página 39.

KOYYA, P.; LEE, Y.; YANG, J. Feedback for programming assignments using software-metrics and reference code. *ISRN Software Engineering*, v. 2013, p. 1–8, 01 2013. Citado na página 16.

KURNIA, A.; LIM, A.; CHEANG, B. Online judge. *Computers Education*, v. 36, p. 299–315, 08 2002. Citado 3 vezes nas páginas 18, 31 e 33.

LABS, S. R. *Sphere Research Labs – IDE ONE*. 2020. Disponível em: <https://ideone.com/>. Acesso em: 30 jan. 2020. Citado na página 39.

LAYTON, M. C.; OSTERMILLER, S. J. *Gerenciamento Ágil De Projetos Para Leigos*. 2. ed. [S.l.]: Altabooks, 2019. 432 p. Citado na página 28.

LEE, C. C. *JavaNCSS - A Source Measurement Suite for Java*. 2018. Disponível em: <http://www.kclee.de/clemens/java/javancss/>. Acesso em: 31 out. 2019. Citado na página 78.

LEMOS, M. A. d.; BARROS, L. N. d.; LOPES, R. d. D. Uma biblioteca cognitiva para o aprendizado de programação. In: *Workshop de Educação em Computação (WEI)-SBC 2003*. [S.l.]: Sociedade Brasileira de Computação - SBC, 2003. Citado na página 23.

LOPES, B. et al. Método de ensino de programação mediada por simulação: Um estudo de caso no curso técnico integrado em informática. In: V CONGRESSO BRASILEIRO DE INFORMÁTICA NA EDUCAÇÃO. *Anais do XXVII Simpósio Brasileiro de Informática na Educação (SBIE 2016)*. Uberlândia/MG, 2016. p. 340. Citado na página 22.

MARCOLINO, A.; BARBOSA, E. Softwares educacionais para o ensino de programação: Um mapeamento sistemático. In: *IV Congresso Brasileiro de Informática na Educação*. [S.l.: s.n.], 2015. Citado 2 vezes nas páginas 22 e 31.

MARCONI, M. d. A.; LAKATOS, E. M. *Fundamentos de metodologia científica*. 8. ed. São Paulo: Atlas, 2017. 256 p. Citado na página 66.

MARTIN, R. C. *Agile Software Development - Principles, Patterns, and Practices*. 1. ed. [S.l.]: Prentice Hall, 2002. Citado na página 28.

MARTIN, R. C.; COPLIEN, J. O. *Clean Code - A Handbook of Agile Software Craftsmanship*. 1. ed. [S.l.]: Prentice Hall, 2008. Citado na página 28.

MCCABE, T. J. Mccabe, a complexity measure. *IEEE Transaction on Software Engineering*, v. 2, p. 308–320, 01 1976. Citado na página 78.

MEIRELLES, P. R. M. *Monitoramento de métricas de código-fonte em projetos de software livre*. Tese (Doutorado) — Instituto de Matematica e Estatística, Universidade de São Paulo, 2013. Citado na página 19.

MOODLE. *Moodle - Open source learning platform*. 2020. Disponível em: <https://moodle.org/>. Acesso em: 13 fev. 2020. Citado na página 33.

MOREIRA, M.; FAVERO, E. Um ambiente para ensino de programação com feedback automático de exercícios. p. 66075–110, 01 2009. Citado na página 42.

NAUDÉ, K.; GREYLING, J.; VOGTS, D. Marking student programs using graph similarity. *Computers Education*, v. 54, p. 545–561, 02 2009. Citado 2 vezes nas páginas 27 e 43.

NEJMEH, B. A. Npath: A measure of execution path complexity and its applications. *Commun. ACM*, Association for Computing Machinery, New York, NY, USA, v. 31, n. 2, p. 188–200, fev. 1988. ISSN 0001-0782. Disponível em: <https://doi.org/10.1145/42372.42379>. Citado na página 79.

NEVES, A. et al. Pcodigo ii: O sistema de diagnóstico da aprendizagem de programação por métricas de software. In: . [S.l.: s.n.], 2017. p. 339. Citado na página 35.

OLIVEIRA, A.; MOTA, L.; OLIVEIRA, A. Uso de ambientes virtuais de aprendizagem como suporte ao ensino de programação: uma revisão sistemática. *Interfaces Científicas - Exatas e Tecnológicas*, v. 1, p. 9, 10 2015. Citado na página 31.

OLIVEIRA, A. S. et al. Uma proposta para ensino semipresencial de programação apoiada por ambiente virtual de aprendizagem e juiz on-line. In: COMPUTER ON THE BEACH, 2019, Florianópolis, SC. *Anais do Computer on the Beach 2019*. Florianópolis, SC, 2019. p. 756. Citado na página 22.

OLIVEIRA, E. W.; BORGES, M. R. d. S. O efeito da competição e da recompensa na motivação à aprendizagem. In: *Simpósio Brasileiro de Sistemas Colaborativos (SBSC)*. Salvador, BA: Sociedade Brasileira de Computação - SBC, 2015. Citado na página 34.

OLIVEIRA, J. et al. Mensagens estendidas de feedback em um juiz online para alunos de introdução à computação: resultados preliminares. In: . [S.l.: s.n.], 2019. p. 329. Citado na página 39.

OLIVEIRA, M. et al. Análise de aprendizagem a partir de códigos-fontes e um proposta de seleção automática de métricas de avaliação. In: . [S.l.: s.n.], 2018. p. 369. Citado 6 vezes nas páginas 16, 18, 25, 31, 36 e 41.

OLIVEIRA, M. et al. Mapeamento automático de perfis de estudantes em métricas de software para análise de aprendizagem de programação. In: . [S.l.: s.n.], 2017. p. 1337. Citado 2 vezes nas páginas 35 e 36.

OLIVEIRA, M.; NOGUEIRA, M.; OLIVEIRA, E. Sistema de apoio à prática assistida de programação por execução em massa e análise de programas. In: *XIV Workshop de Educação em Computação (WEI)-SBC 2015*. Recife, PE: Sociedade Brasileira de Computação - SBC, 2015. Citado 5 vezes nas páginas 34, 35, 40, 41 e 43.

OLIVEIRA, M. de. As tecnologias de análise de aprendizagem e os desafios de prever desempenhos de estudantes de programação. In: *Anais do V Workshop de Desafios da Computacao aplicada a Educacao*. Porto Alegre, RS, Brasil: SBC, 2015. p. 80–89. Disponível em: <https://portaldeconteudo.sbc.org.br/index.php/desafie/article/view/9164>. Citado 13 vezes nas páginas 15, 16, 24, 25, 26, 27, 31, 32, 41, 42, 43, 46 e 48.

PAES, R. et al. Ferramenta para a avaliação de aprendizado de alunos em programação de computadores. In: . [S.l.: s.n.], 2013. Citado 8 vezes nas páginas 14, 15, 18, 19, 37, 42, 43 e 47.

PALMEIRA, L. B.; SANTOS, M. P. *Evasão no bacharelado em ciência da computação da universidade de Brasília: análise e mineração de dados*. Brasília: [s.n.], 2014. Monografia do curso de Ciência da Computação da UnB–Universidade de Brasília. Citado na página 14.

PARANHOS, L. R. L.; RODOLPHP, P. J. *Metodologia da pesquisa aplicada à tecnologia*. 1. ed. São Paulo: SENAI-SP Editora, 2018. Citado na página 66.

PEREIRA, F. et al. Early dropout prediction for programming courses supported by online judges. In: _____. [S.l.: s.n.], 2019. p. 67–72. ISBN 978-3-030-23206-1. Citado 2 vezes nas páginas 14 e 22.

PEREIRA, F.; OLIVEIRA, E. Teixeira de; FERNANDES, D. Predição de zona de aprendizagem de alunos de introdução à programação em ambientes de correção automática de código. In: *XXVIII Simpósio Brasileiro de Informática na Educação (SBIE 2017)*. Recife, PE, Brasil: Sociedade Brasileira de Computação - SBC, 2017. p. 1507. Citado na página 19.

PEREIRA, F. D. *Uso de um método preditivo para inferir a zona de aprendizagem de alunos de programação em um ambiente de correção automática de código*. Dissertação (Mestrado em Ciência da Computação) — Universidade Federal do Amazonas, Manaus, 2018. Citado na página 23.

PEROVANO, D. G. *Manual de metodologia da pesquisa científica*. 1. ed. Curitiba: InterSaberes, 2016. 384 p. Citado na página 67.

PETERSEN, A.; SPACCO, J.; VIHAVAINEN, A. An exploration of error quotient in multiple contexts. In: *Proceedings of the 15th Koli Calling Conference on Computing Education Research*. [S.l.: s.n.], 2015. p. 77–86. Citado na página 19.

PRESSMAN, R. *Engenharia de Software - 7.ed.* McGraw Hill Brasil, 2009. ISBN 9788580550443. Disponível em: <https://books.google.com.br/books?id=y0rH9wuXe68C>. Citado na página 28.

PRODANOV, C. C.; FREITAS, E. C. *Metodologia do trabalho científico: Métodos e Técnicas da Pesquisa e do Trabalho Acadêmico*. 2. ed. Rio Grande do Sul: Feevale, 2013. Citado na página 67.

QUEIROZ, M.; DANTAS, A. Neurociência e o ensino de programação: Uma revisão sistemática da literatura. In: . [S.l.: s.n.], 2018. p. 1033. Citado na página 16.

RAABE, A.; SILVA, J. Marques Carvalho da. Um ambiente para atendimento as dificuldades de aprendizagem de algoritmos. p. 2326–2335, 07 2005. Citado na página 23.

RABÊLO JÚNIOR, D. et al. Cosmo: Um ambiente virtual de aprendizado com foco no ensino de algoritmos. In: . [S.l.: s.n.], 2018. Citado 2 vezes nas páginas 14 e 15.

RAHMAN, K.; AHMAD, S.; NORDIN, M. J. The design of an automated c programming assessment using pseudo-code comparison technique. 01 2007. Citado 2 vezes nas páginas 18 e 26.

RAMOS, V. et al. A comparação da realidade mundial do ensino de programação para iniciantes com a realidade nacional: Revisão sistemática da literatura em eventos brasileiros. In: *IV Congresso Brasileiro de Informática na Educação*. [S.l.: s.n.], 2015. p. 318. Citado na página 31.

RAPKIEWICZ, C. E. et al. EstratÉgias pedagÓgicas no ensino de algoritmos e programaÇÃo associadas ao uso de jogos educacionais. *RENOTE*, v. 4, 12 2006. Citado na página 16.

RAPOSO, A.; MARANHÃO, D.; SOARES NETO, C. Análise do modelo bkt na avaliação da curva de aprendizagem de alunos de algoritmos. In: *XXVI Simpósio Brasileiro de Informática na Educação (SBIE 2019)*. [S.l.: s.n.], 2019. p. 479. Citado 4 vezes nas páginas 15, 22, 23 e 41.

REDDY, A. Java coding style guide. In: _____. [S.l.]: Sun Microsystems, 2000. Citado na página 28.

ROCHA, P. et al. Ensino e aprendizagem de programação: Análise da aplicação de proposta metodológica baseada no sistema personalizado de ensino. *RENOTE*, v. 8, 03 2010. Citado 4 vezes nas páginas 14, 15, 22 e 23.

ROMLI, R.; SULAIMAN, S.; ZAMLI, K. Automatic programming assessment and test data generation a review on its approaches. In: *Information Technology (ITSim), 2010, International Symposium*. [S.l.: s.n.], 2010. v. 3, p. 1186 – 1192. Citado 4 vezes nas páginas 26, 27, 40 e 42.

SAIKKONEN, R.; MALMI, L.; KORHONEN, A. Fully automatic assessment of programming exercises. In: . [S.l.: s.n.], 2001. v. 33, p. 133–136. Citado na página 24.

SANTOS, F. A. de O.; FONSECA, L. Collection and analysis of source code metrics for composition of programming learning profiles. In: *2019 IEEE 19th International Conference on Advanced Learning Technologies (ICALT)*. [S.l.: s.n.], 2019. v. 2161-377X, p. 173–175. ISSN 2161-3761. Citado 2 vezes nas páginas 16 e 19.

SANTOS, F. A. de O.; SEGUNDO, P.; TELVINA, M. Codeteacher: Uma ferramenta para correção automática de trabalhos acadêmicos de programação em java. In: . [S.l.: s.n.], 2017. p. 1152. Citado 2 vezes nas páginas 18 e 49.

SANTOS, F. A. de O.; SOARES SEGUNDO, P.; TELVINA, M. Codeteacher: Uma ferramenta para correção automática de trabalhos acadêmicos de programação em java. In: _____. [S.l.: s.n.], 2019. p. 148–157. ISBN 9788572477024. Citado na página 16.

SANTOS, J. C. d. S.; RIBEIRO, A. Uma proposta de um juiz online didático para o ensino de programação. p. 332–341, 01 2011. ISSN 2175-5876. Citado na página 35.

SANTOS, J. S. C.; RIBEIRO, A. de R. L. Jonline: proposta preliminar de um juiz online didático para o ensino de programação. In: XXII SIMPóSIO BRASILEIRO DE INFORMÁTICA NA EDUCAÇÃO. *Anais do XXII SBIE - XVII WIE*. Aracaju-SE: SBC, 2011. Citado 2 vezes nas páginas 32 e 35.

SCHNEIDER, G.; JAQUES, P. Combinando técnicas de análise estática e avaliação dinâmica para avaliação de código em ambientes de aprendizagem de programação. *Revista Brasileira de Computação Aplicada*, v. 8, 04 2016. Citado 7 vezes nas páginas 24, 26, 27, 36, 37, 43 e 62.

SCHWARTZMAN, S. *Pesquisa acadêmica, pesquisa básica e pesquisa aplicada em duas comunidades cientificas*. 6. ed. São Paulo: não publicado, 1979. Citado na página 67.

SILVA, E. L.; MENEZES, E. M. *Metodologia da pesquisa e elaboração de dissertação*. 4. ed. Florianópolis: UFSC, 2005. Citado na página 67.

SILVA, P. et al. Um mapeamento sistemático sobre iniciativas brasileiras em ambientes de ensino de programação. In: *IV Congresso Brasileiro de Informática na Educação*. [S.l.: s.n.], 2015. p. 367. Citado na página 31.

SOMMERVILLE, I. *Engenharia de software*. PEARSON BRASIL, 2011. ISBN 9788579361081. Disponível em: <https://books.google.com.br/books?id=H4u5ygAACAAJ>. Citado 3 vezes nas páginas 19, 49 e 67.

SOUZA, D.; BATISTA, M.; BARBOSA, E. Avaliação de qualidade de um ambiente de apoio ao ensino de programação. *RENOTE*, v. 12, 12 2014. Citado 2 vezes nas páginas 15 e 32.

SOUZA, D.; FELIZARDO, K.; BARBOSA, E. A systematic literature review of assessment tools for programming assignments. In: . [S.l.: s.n.], 2016. p. 147–156. Citado 2 vezes nas páginas 24 e 31.

SOUZA, D.; ISOTANI, S.; BARBOSA, E. Teaching novice programmers using progtest. *International Journal of Knowledge and Learning*, v. 10, p. 60–77, 01 2015. Citado na página 32.

SOUZA, D.; MALDONADO, J.; BARBOSA, E. Progtest: An environment for the submission and evaluation of programming assignments based on testing activities. In: *Software Engineering Education and Training (CSEE T)*. [S.l.]: 2011 24th IEEE-CS Conference on, 2011. p. 1–10. Citado 2 vezes nas páginas 16 e 32.

SOUZA, D. M. de; MALDONADO, J. C. M.; BARBOSA, E. F. Aspectos de desenvolvimento e evolução de um ambiente de apoio ao ensino de programação e teste de software. In: *Anais do 23º Simpósio Brasileiro de Informática na Educação*. Rio de Janeiro: SBC, 2012. ISSN 2316-6533. Citado na página 32.

STRIEWE, M.; GOEDICKE, M. A review of static analysis approaches for programming exercises. In: KALZ, M.; RAS, E. (Ed.). *Computer Assisted Assessment. Research into E-Assessment*. Cham: Springer International Publishing, 2014. v. 439, p. 100–113. ISBN 978-3-319-08657-6. Citado na página 31.

TOBAR, C. et al. Uma arquitetura de ambiente colaborativo para o aprendizado de programação. In: *XII SBIE-2001 - Simpósio Brasileiro de Informática na Educação*. [S.l.: s.n.], 2001. Citado na página 16.

UFAM, U. F. d. A. *Um Sistema Juiz Online para Cursos de Computação*. 2020. Disponível em: <http://codebench.icomp.ufam.edu.br/>. Acesso em: 31 de jan. de 2020. Citado na página 39.

VENDRAME, B. et al. Mapeamento sistemático sobre ferramentas digitais online para o ensino-aprendizagem de algoritmos e programação no ensino superior. In: COMPUTER ON THE BEACH, 2019, Florianópolis, SC. *Anais do Computer on the Beach 2019*. Florianópolis, SC, 2019. Citado 5 vezes nas páginas 14, 16, 31, 41 e 43.

VERMEULEN, A. et al. *The Elements of JavaTM Style*. [S.l.]: Cambridge University Press, 2000. Citado na página 28.

VIANA, G.; PORTELA, C. O uso de softwares educativos para introdução de lógica de programação no ensino de base e superior. *Informática na educação: teoria prática*, v. 22, 05 2019. Citado na página 22.

WATSON, C.; LI, F. Failure rates in introductory programming revisited. *ITICSE 2014 - Proceedings of the 2014 Innovation and Technology in Computer Science Education Conference*, 06 2014. Citado na página 14.

WAZLAWICK, R. S. *Metodologia de pesquisa para Ciência da Computação*. 2. ed. Rio de Janeiro: Elsevier, 2014. Citado 2 vezes nas páginas 66 e 67.

WEBER, G.-W.; STEINLE, F. Elm-pe: An intelligent learning environment for programming. In: . [S.l.: s.n.], 1996. Citado na página 22.

WEINBERG, G. *The Psychology Of Computer Programming*. [S.l.: s.n.], 1971. v. 18. Citado na página 22.

WILDT, D. *eXtreme Programming : Práticas para o dia a dia no desenvolvimento ágil de software*. 1. ed. [S.l.]: Casa do Código, 2015. 162 p. ISBN 9788555191060. Citado na página 28.