UNIVERSIDADE FEDERAL DE SERGIPE

CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA

DEPARTAMENTO DE COMPUTAÇÃO

Uma abordagem para pontuação de exercícios de programação baseada em análise estática de código

Trabalho de Conclusão de Curso

Victor Souza Vieira

São Cristóvão – Sergipe

2021

UNIVERSIDADE FEDERAL DE SERGIPE

CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA

DEPARTAMENTO DE COMPUTAÇÃO

Victor Souza Vieira

Uma abordagem para pontuação de exercícios de programação baseada em análise estática de código

Trabalho de Conclusão de Curso submetido ao Departamento de Computação da Universidade Federal de Sergipe como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação.

Orientador(a): Professor Doutor Alberto Costa Neto

São Cristóvão – Sergipe

2021

Resumo

Com o aumento massivo de estudantes em cursos introdutórios de computação ocorrendo nas últimas décadas, o trabalho dos professores para corrigir seus execícios aumentou significativamente, como uma das soluções para este problema foram criados os analisadores automáticos de código fonte, estes analisadores podem ser divididos basicamente em três categorias: analisadores estáticos, analisadores dinâmicos e analisadores híbridos (utilizam tanto análise estática quanto dinâmica). Algumas das implementações de analisadores dinâmicos de código são conhecidas como *Online Judges* onde estes avaliam se durante a execução do código fonte todos os casos de teste foram ou não aprovados, caso aprovados o exercício receberá nota máxima caso contrário nota mínima, esta prática muitas das vezes acaba frustrando os alunos iniciantes, pois apenas é avaliado o resultado final do programa e não as partes que o compõem. Como solução, para que haja uma melhora nas notas destes estudantes, é proposta uma ferramenta baseada em análise estática de código, para a linguagem de programação *Python*, que irá pontuar com base em regras que podem ser definidas pelo professor o código escrito pelos alunos dando assim um melhor *feedback* para o resultado da avaliação.

Palavras-chave: Análise estática. Linguagem de programação *Python*. Código fonte.

Sumário

1 Introdução

Com o desenvolvimento constante das tecnologias da informação e comunicação (TIC), no que diz respeito à aprendizagem, configurações e métodos de avaliação vêm se tornando um imenso desafio (WESIAK; AL-SMADI; GÜTL, 2012). Em contrapartida, de acordo com (AMER; HAROUS, 2017), o rápido avanço tenológico mudou a forma como os professores ensinam e como os alunos aprendem, desta maneira fazendo com que os processos de ensino e aprendizagem sejam continuamente melhorados para que seja possível a ambas partes conviverem em harmonia.

Aprender programação não é uma tarefa trivial, pois, o ato de programar requer raciocínio e habilidades criativas para resolver problemas (RONGAS; KAARNA; KALVIAINEN, 2004), diversas ferramentas foram desenvolvidas ao longo do tempo para auxiliar professores e alunos em cursos introdutórios de programação. O autor também dividiu essas ferramentas em quatro categorias:

a) Ambientes de Desenvolvimento Integrado (IDE);

b) Visualização;

c) Ambientes Virtuais de Aprendizagem (AVA);

d) Sistemas para apresentação, gerenciamento e teste de exercícios, conhecidos também como *online judges*.

Nos cursos introdutórios de programação os discentes geralmente são apresentados a algum ambiente que permita que eles desenvolvam suas lógicas e escrevam programas, várias vezes estes ambientes irão permitir que os estudantes, editem, depurem, compilem e executem os programas que foram construídos. Por outro lado, todas essas informações podem acabar sobrecarregando os alunos diminuindo assim sua motivação e performance, podendo até levar ao abandono do curso (BRANDãO; RIBEIRO; BRANDãO, 2012).

O processo de avaliar os alunos de cursos introdutórios de programação é de grande

importância para que haja uma constatação do seu aprendizado ou não, porém, com o aumento da quantidade de alunos nestes cursos acaba sendo gerada uma sobrecarga maior nos professores que os avaliam (ARIFI et al., 2015). Com isso várias atividades que antes eram feitas de forma manual foram total ou parcialmente substituídas por métodos automatizados para que assim seja possível lidar com a demanda crescente de matriculados nestes tipos de curso (POŽENEL; FüRST; MAHNIč, 2015).

Os métodos automatizados de avaliação de código, especialmente os *Online Judges*, utilizam-se basicamente da análise de casos de teste, independente da qualidade do código escrito ou se o código está de acordo com o que foi ensinado, levando assim aos alunos em diversos casos aprenderem técnicas de codificação inadequadas (C et al., 2019). No capítulo 2 veremos que a análise de casos de teste pertence ao grupo de análise dinâmica de código.

Considere aqui um programa para calcular o n-ésimo termo da sequência de Fibonacci, é sabido que, existem formas iterativas e recursivas para escrever este programa e todas fornecerão o mesmo resultado. Agora imagine que o professor deseja que o aluno encontre a solução para este problema utilizando apenas a forma iterativa. Considerando este caso a análise dinâmica baseada em casos de testes irá falhar, pois, tanto as soluções iterativas quanto as soluções recursivas fornecerão o mesmo resultado (C et al., 2019). Seguindo o raciocínio do pensamento anterior, o aluno que desenvolveu o programa para calcular o n-ésimo termo da sequência de qualquer uma das duas formas e tenha sido aprovado em todos os casos de teste, irá tirar a nota máxima neste exercício, já o aluno que desenvolveu utilizando a técnica pedida pelo professor e por algum motivo seu código não foi aprovado em todos os casos de teste, receberá nota mínima. Dessa maneira não há um meio termo, ou o aluno recebe a nota máxima ou a nota miníma, independente da forma como o programa foi escrito, sendo assim, os alunos iniciantes em cursos introdutórios de programação, podem acabar sentindo-se frustrados por esta forma de avaliação binária e que não bonifica por seguir boas práticas de programação.

1.1 Objetivos

1.1.1 Gerais

O objetivo deste trabalho é propor uma ferramenta automática de análise estática de código para que seja possível, com base em regras que podem ser configuradas, avaliar a qualidade do código fonte desenvolvido auxiliando professores no processo de correção dos exercícios de programação ao mesmo tempo que provendo *feedback* para o aluno sobre os aspectos de qualidade do seu código fonte.

1.1.2 Específicos

São objetivos específicos deste trabalho:

- Realizar um mapeamento sistemático a respeito das ferramentas existentes de análise estática de código.
- Analisar as ferramentas encontradas e discutir seus pontos fortes e fracos, para assim, propor uma ferramenta melhor voltada para a realidade dos docentes e discentes da universidade.

1.2 Estrutura do documento

Para uma melhor navegabilidade e entendimento das divisões deste trabalho, o documento está dividido da seguinte maneira.

- Capítulo 1 - Introdução: é apresentada a situação problema que será abordada neste trabalho, suas causas e justificativas além dos objetivos deste documento;
- Capítulo 2 - Fundamentação teórica: neste capítulo temos abordados os principais conceitos referentes ao tema assim como, sua base teórica para contextualização;
- Capítulo 3 - Revisão sistemática: traz os resultados da revisão sistemática sobre a ferramenta proposta;
- Capítulo 4 - Ferramenta proposta.

2

Fundamentação teórica

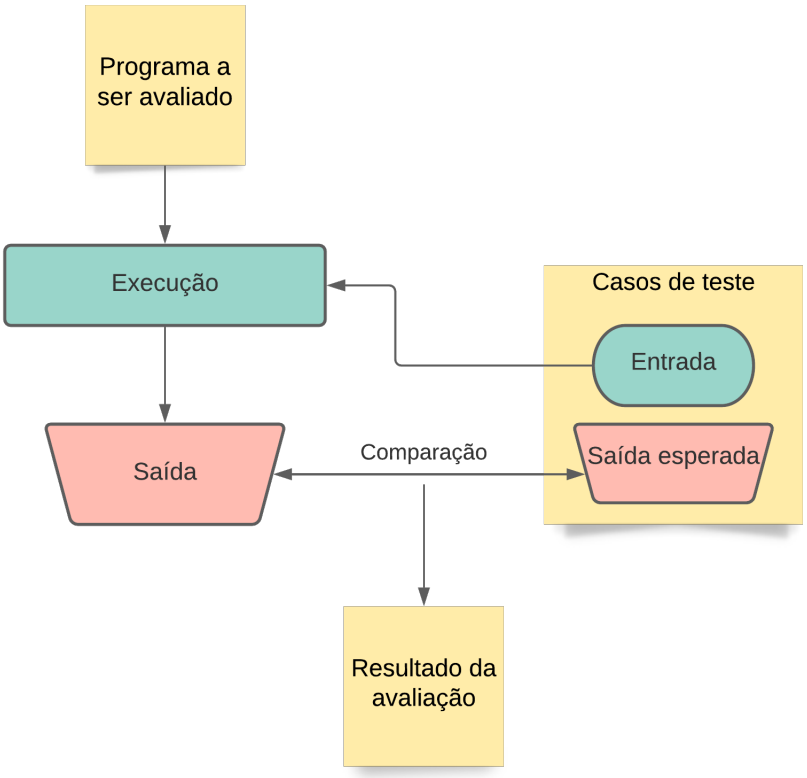
2.1 Análise dinâmica de código

Segundo (ARIFI et al., 2015) a análise dinâmica consiste em executar o programa utilizando como entrada uma sequência de casos de teste onde, para cada caso, existe uma saída esperada e uma saída fornecida pelo programa, no fim estas são comparadas e caso sejam iguais o caso de teste foi um sucesso caso contrário, insucesso. Um programa modelo é construído e tem suas saídas coletadas de acordo com um conjunto de casos de teste e depois os programas que executarem sobre estes casos de teste terão suas saídas comparadas ao do programa modelo (ARIFI et al., 2016). A Figura 1 mostra o processo de funcionamento da análise dinâmica.

Ainda de acordo com (ARIFI et al., 2016), por não possuir uma implementação muito complexa, a maioria dos sistemas de avaliação automatizados utilizam a análise dinâmica, porém vale ressaltar que existem certas fragilidades com esta abordagem, sendo a principal delas a impossibilidade de analisar o programa caso ele não compile ou apresente um ou mais erros durante sua execução.

Existem alguns pontos fracos em utilizar a análise dinâmica para avaliar alunos iniciantes, pois, geralmente eles costumam cometer erros como, esquecer de colocar ponto e vírgula ao final de uma sentença ou esquecer de fechar um parênteses que foi aberto. Como os sistemas de análise dinâmica são sensíveis a erro, cometer alguns dos erros citados anteriormente inviabilizaria a execução do programa, sendo atribuída nota zero à questão (ARIFI et al., 2015).

Figura 1 – Processo da análise dinâmica



Fonte: Autor

Um outro ponto fraco da análise dinâmica é o *feedback* gerado ao término da execução, onde basicamente seria:

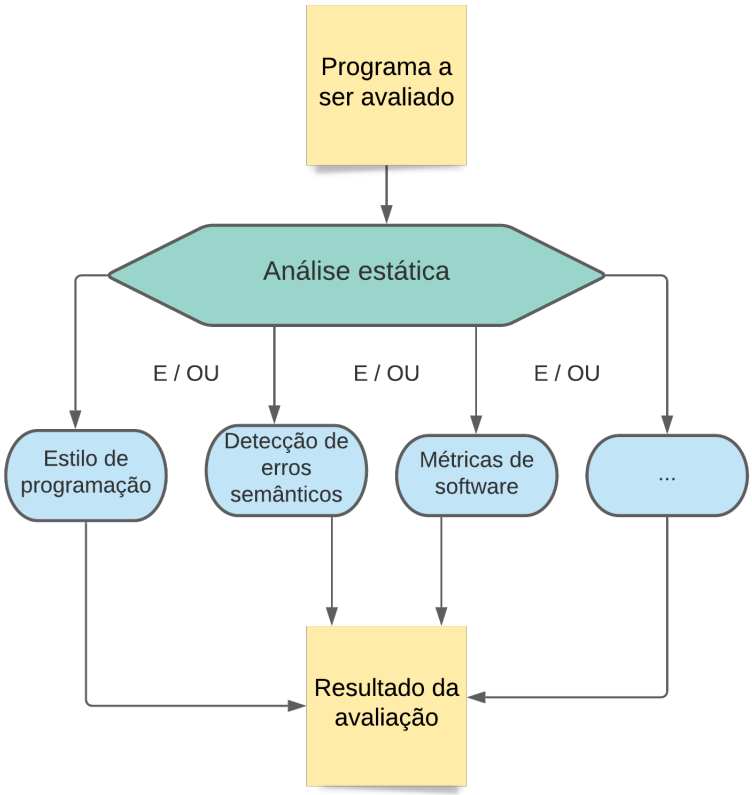
- "Correto": quando o programa foi executado sem erros e passou em todos os casos de teste;
- "Incorreto": quando o programa executou, mas não passou em todos os casos de teste;
- "Erro de compilação": quando o programa não teve sua compilação bem sucedida;
- "Erro de execução": quando ocorreu algum erro durante a execução do programa, por exemplo, divisão por 0;

Desta maneira os sistemas de avaliação que utilizam este tipo de análise de código são mais recomendados para alunos mais avançados em cursos de programação, pois eles certamente possuirão uma maior compreensão sobre o que estão escrevendo, além de já possuírem uma familiaridade maior com o desenvolvimento de programas.

2.2 Análise estática de código

Diferentemente da análise dinâmica que é feita executando o programa para que seja possível a comparação com os casos de teste, a análise estática é realizada sem a necessidade dessa execução pois, esta, busca extrair informações e métricas a respeito do código escrito (ARIFI et al., 2015). A Figura 2 ilustra o funcionamento do processo da análise estática.

Figura 2 – Processo da análise estática



Fonte: Autor

As próximas subseções abordam alguns dos diversos métodos desta análise.

2.2.1 Análise do estilo de programação

Neste método, então, são utilizadas técnicas que tentem ao máximo explicitar o estilo que o programador escreve seus códigos o que torna esta análise útil também para auxiliar na verificação de plágio. Alguns fatores são considerados neste método, sendo alguns deles:

- Nomes de variáveis;
- Uso de constantes;

- Espaçamento entre linhas;
- Modularização do código;

2.2.2 Detecção de erros semânticos

Erros semânticos ocorrem quando existem sentenças que estão gramaticalmente corretas, mas que durante a execução do programa causarão algum erro. Alunos iniciantes tendem a cometer vários erros semânticos como por exemplo a divisão por zero ou alguma recursão infinita. A priori erros semânticos parecem não ter grande impacto num programa, mas imagine este problema num sistema financeiro por exemplo, a quantidade de dinheiro perdido pode ser incalculável.

Atualmente grande parte das IDEs trabalham com a pré-compilação do código fonte, isto é, executa a compilação enquanto o programador está codificando, para tentar minimizar ao máximo que estes erros venham a ocorrer.

2.2.3 Análise de métricas de software

A análise de métricas pode ser usada para auxiliar na verificação de confiabilidade e complexidade do programa. Esta análise cobre aspectos relacionados ao tamanho, complexidade e qualidade que podem ser medidos no código (ARIFI et al., 2015). São exemplos dos aspectos citados: frequência de comentários, quantidade de linhas de código, número de operadores e operandos para determinar quão complexa uma sentença pode ser, quantidade de declarações de variáveis e funções e assim por diante.

2.2.4 Análise de palavras-chave

O ponto deste método é que deve existir uma definição de palavras-chave pelo avaliador e em seguida a análise é feita através da busca dessas palavras no código-fonte, assim podendo determinar se elas foram ou não utilizadas.

2.2.5 Análise de similaridade estrutural e não estrutural

Para esta análise se faz necessária a existência de uma ou mais soluções para serem utilizadas como base de comparação com as soluções posteriormente fornecidas pelos alunos assim, podendo determinar níveis de similaridade entre elas e quanto mais próximas forem as soluções, maiores serão as notas atribuídas a elas.

2.3 Linguagem de programação Python

A linguagem de programação *Python* foi criada no ano de 1989 por Guido Van Rossum na Holanda, após ele procurar, sem êxito, uma linguagem de programação semelhante à linguagem ABC e que pudesse realizar *system calls* (chamadas de sistema) no sistema operacional Amoeba Lima (2021).

Python é uma linguagem de programação de alto nível, orientada a objetos, multiplataforma e interpretada e vem ganhando maior espaço no mercado no últimos anos, pois é uma linguagem relativamente mais fácil de aprender, possui uma comunidade gigantesca no mundo inteiro, inúmeras bibliotecas, nativas e de terceiros, para auxiliar os programadores nas suas tarefas além da sua grande utilização nas áreas de: ciência de dados, *machine learning* (aprendizado de máquina), inteligência articifial, automatização de processos, computação gráfica Roveda (2020).

Ainda segundo Roveda (2020) são vantagens da linguagem python:

- Comunidade grande de desenvolvedores
- Fácil aprendizado
- Diversos *frameworks*
- Portátil, extensível e multiplataforma
- Licença de uso público

A linguagem *Python*, também, permite ao programador escrever o mesmo requisito com uma menor quantidade de linhas de código quando comparado a linguagens de programação mais verbosas Devmedia (2020).

Em uma pesquisa realizada em 2020 pela ***StackOverflow Survey***[1], *Python* ficou em primeiro lugar como a linguagem que os desenvolvedores mais gostariam de utilizar.

[1] <https://insights.stackoverflow.com/survey/2020#technology-most-loved-dreaded-and-wanted-languages-wanted>

3

Revisão sistemática

3.1 Protocolo da revisão

Foi definido um protocolo de busca visando responder às seguintes questões:

- Q1 - Quais ferramentas de análise estática de código, voltadas para auxiliar professores no processo de correção de avaliações de exercícios de programação em turmas introdutórias existem?
- Q2 - Quais as linguagens de programação são suportadas pelas ferramentas encontradas?

As bases de conhecimento selecionadas para esta revisão foram: IEEE[1] sigla para *Institute of Electrical and Electronics Engineers*, a Scopus[2] e a Science Direct[3]. Para que fosse possível acessar os artigos completos destas bases foi utilizado o portal do periódicos CAPES[4].

Para a realização da consulta dos artigos, nas bases de conhecimento, algumas *strings* de busca foram criadas, pois além de facilitar a procura desses artigos elas também auxiliam no processo de filtragem deles. A Tabela 1 traz as *strings* utilizadas na pesquisa.

[1] Disponível em: <https://ieeexplore.ieee.org/>
[2] Disponível em: <https://www.scopus.com/>
[3] Disponível em: <https://www.sciencedirect.com/>
[4] Disponível em: <https://www-periodicos-capes-gov-br.ez20.periodicos.capes.gov.br/>

Tabela 1 – *Strings* de busca

Base	***String***	**Nº de resultados**
IEEE Xplore	(("All Metadata":Automatic Grading Of Programming Exercises) OR ("All Metadata":Static Analysis Of Metrics For Software Quality In The Academic Environment)) AND ("Index Terms":computer science education)	11
Scopus	("Automatic Grading Of Programming Exercises"OR "Static Analysis Of Metrics For Software Quality In The Academic Environment")	48
Sciente Direct	('Automatic Grading Of Programming Exercises') OR ('Static Analysis Of Metrics For Software Quality In The Academic Environment')	232

Fonte: Autor

Com o intuito de aumentar mais a filtragem dos 291 artigos encontrados, foram definidos critérios de inclusão e exclusão, para que assim fossem mantidos os artigos mais coerentes com o objetivo deste trabalho. Os critérios de inclusão definidos foram:

- Propor uma ferramenta de análise estática de código, configurável e voltada para avaliação de exercícios de programação.

Já os critérios de exclusão:

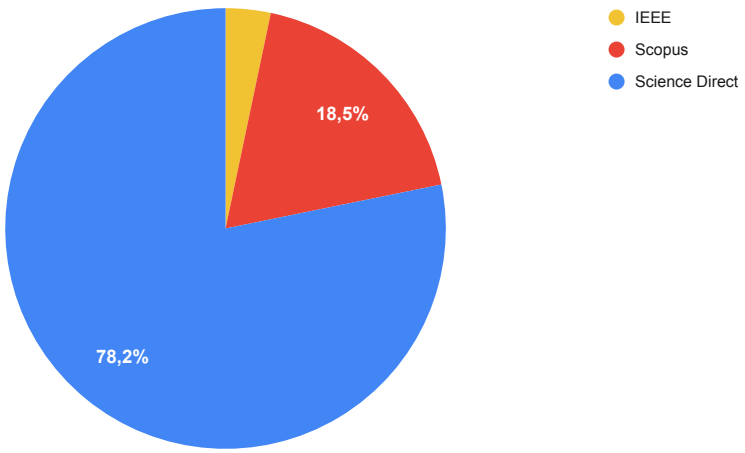
- Proposta de ferramenta que não utilize análise estática de código;
- Proposta que não deixe claro qual o método de análise;
- Proposta de ferramenta que não seja voltada para avaliação de exercícios de programação;
- Proposta que seja voltada para o ambiente profissional de desenvolvimento de software;

Após a coleta dos artigos científicos nas três bases citadas anteriormente foram aplicados os processos de inclusão e exclusão no título e resumo de cada um dos artigos encontrados. A próxima seção demonstra os resultados encontrados.

3.2 Resultados encontrados

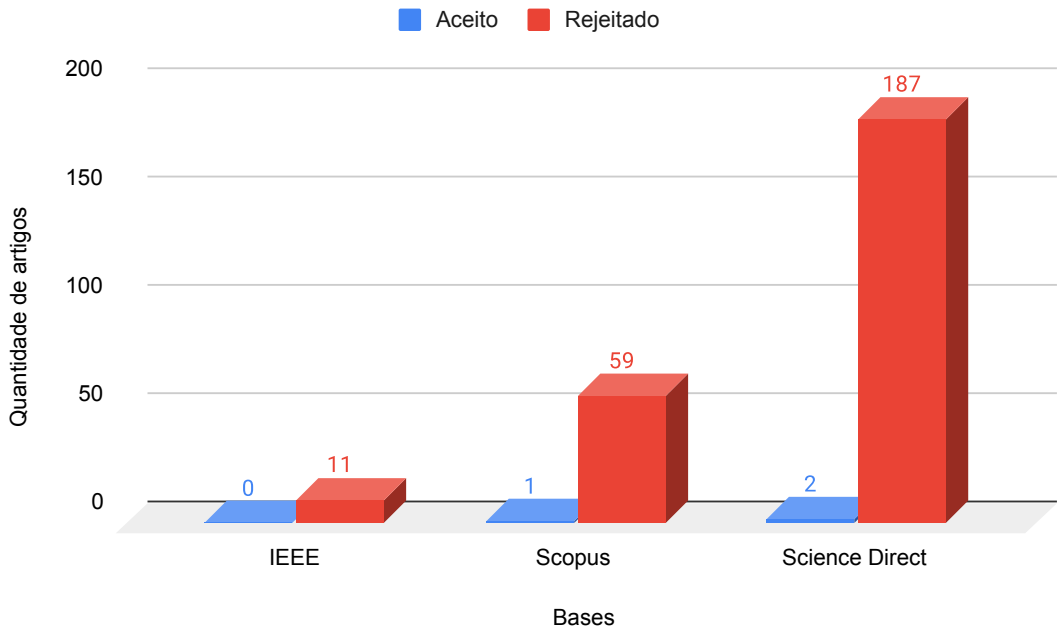
Aplicando os critérios de inclusão e exclusão nos 291 artigos encontrados, nas 3 bases de conhecimento, foram considerados aceitos 3, duplicados 31 e rejeitados 257. A Figura 3 diz respeito à distribuição percentual dos artigos encontrados por base, a Figura 4 exibe a quantidade de artigos encontrados por base, já a Figura 5 traz os anos em que foram lançados os artigos que foram aceitos no critério de inclusão.

Figura 3 – Distribuição percentual dos artigos encontrados por base de conhecimento



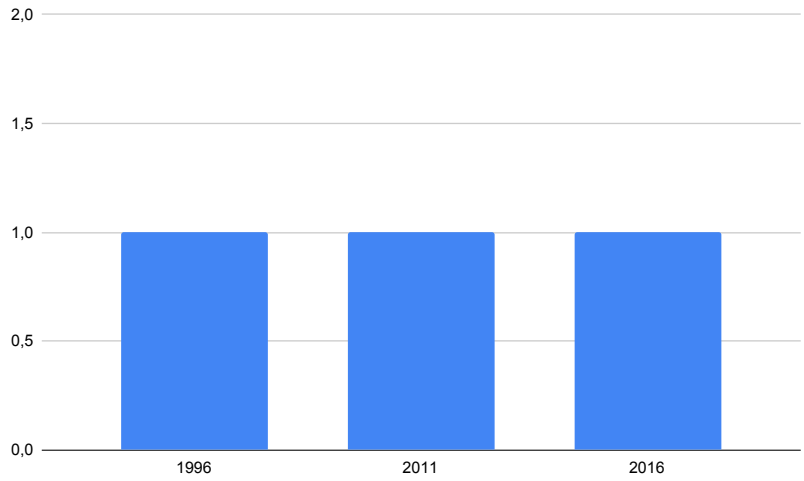
Fonte: Autor

Figura 4 – Distribuição quantitativa de artigos aceitos e rejeitados por base de conhecimento



Fonte: Autor

Figura 5 – Quantidade de artigos aceitos lançados por ano



Fonte: Autor

3.3 Análise dos resultados

As próximas subseções dizem respeito aos artigos que foram selecionados no critério de inclusão, as duas primeiras propostas fogem um pouco do critério, mas foram consideradas visto que, seu conteúdo é de fato relevante para este trabalho.

3.3.1 *A software system for grading student computer programs*

No artigo produzido por (JACKSON, 1996), o autor discute sobre como as avaliações devem incluir julgamentos sobre a qualidade das soluções produzidas pelos alunos nos cursos de computação. O autor então propõe uma ferramenta de análise híbrida (que combina análise estática e análise dinâmica) capaz de avaliar os seguintes critérios:

- Corretude da solução apresentada
- Eficiência da solução
- Complexidade
- Análise de estilo de código contendo:
  - Percentual de linhas em branco
  - Percentual de linhas comentadas
  - Espaços por linhas
  - *Gotos*

  - Percentual de indentação
  - Tamanho de funções
  - Caracteres por linha
  - Palavras-chave utilizadas

- Taxa de acerto em casos de teste

A ferramenta proposta por (JACKSON, 1996) embora não utilize apenas análise estática, foi inclusa nos artigos aceitos. O motivo desta inclusão se deu pelo fato da ferramenta estar bem particionada entre as análises e os pontos de avaliação de cada uma delas, o autor também utilizou pesos entre cada um dos itens da lista acima descrita, tornando assim possível uma avaliação mais justa dos exercícios dos alunos.

3.3.2 *Ability-training-oriented automated assessment in introductory programming course*

A ferramenta proposta por (WANG et al., 2011), AutoLEP é bastante robusta no quesito de suas análises e feedback, novamente se trata de uma ferramenta de análise híbrida, mas diferentemente da proposta anterior, esta, normaliza os códigos fornecidos pelos estudantes e as soluções fornecidas pelo professores para assim poder remover variações entre os códigos e tornar o processo de análise mais simples. O AutoLEP foi projetado para fornecer testes suficientes para que seja possível testar todas as possibilidades de determinados problemas, verificar se os programas atendem as especificações ou não e conseguir funcionar mesmo quando os programas possuem erros sintáticos e semânticos, para que desta forma seja possível fornecer um feedback mais preciso, assim como uma pontuação condizente com as soluções aprensentadas.

3.3.3 *Semantic similarity based evaluation for C programs through the use of symbolic execution*

A proposta apresentada no trabalho de (ARIFI; ZAHI; BENABBOU, 2016) traz como centro a análise estática e o uso de grafos de fluxo de controle para que seja possível realizar comparações entre uma solução fornecida pelo avaliador com outra solução fornecida pelo aluno que está sendo avaliado. O grande desafio deste trabalho foi conseguir lidar com a questão relacionada ao conjunto de soluções existentes para algum determinado exercício, para contornar o problema, uma solução inovadora foi alcançada, sendo esta a comparação de programas de acordo com sua execução semântica como medida para verificar a similaridade entre as soluções fornecidas.

Não foi possível ainda obter mais detalhes sobre este trabalho, pois a plataforma do *scopus* não permitiu a leitura na íntegra do documento.

4

Ferramenta proposta

Para que seja possível auxiliar professores de cursos introdutórios de computação no processo correção de exercícios de programação, a ferramenta proposta irá utilizar a análise estática de código para que assim seja possível aliviar a carga de trabalho que seria corrigir inúmeros exercícios de programação indo além da correção básica oferecida pelos juízes *online*.

4.1 Descrição do analisador estático

Para que seja possível extrair métricas dos exercícios enviados pelos alunos, será necessário que, para cada exercício de programação, exista pelo menos uma solução fornecida pelo professor e um conjunto de regras seja pré-definido, juntamente com os pesos para cada regra. Desta forma o analisador será capaz de fornecer as métricas de acordo com os dados informados.

4.2 Métricas que poderão ser consideradas para análise

Utilizando como inspiração o trabalho de (JACKSON, 1996), algumas das métricas citadas no seu trabalho serão utilizadas como por exemplo a quantidade de linhas de código no escopo uma função. Outras métricas ainda serão consideradas para a composição da ferramenta.

4.3 Linguagem de programação suportada pela ferramenta

Inicialmente, a linguagem de programação que será suportada pela ferramenta é a *Python*, visto que está sendo amplamente utilizadas em cursos introdutórios de programação, assim tornando o processo mais simples, já que haverá vários cenários para possíveis testes da ferramenta na Universidade Federal de Sergipe.

4.4 Linguagem de programação na qual a ferramenta será desenvolvida

Por conveniência a linguagem de programação escolhida para o desenvolvimento do analisador estático será também a *Python*, já que esta linguagem possui uma gama enorme de ferramentas que auxiliarão no desenvolvimento, sem contar na comunidade que cresce a cada dia.

4.5 Planejamento das atividades

Nesta seção serão apresentadas as fases do projeto Tabela 2 juntamente com o cronograma de trabalho Quadro 1.

Tabela 2 – Fases do projeto

1 - Estudo dos processos da análise estática
2 - Métricas de análise de código
3 - Implementação da ferramenta
4 - Validação da ferramenta
5 - Escrever o TCC
6 - Apresentar o TCC

Fonte: Autor

As próximas etapas do trabalho, são resumidas em estudar com maior enfase os processos envolvendo a análise estática de código, métricas de análise de código, implementação da ferramenta, validação da ferramenta, por fim, escrever e apresentar o TCC.

O cronograma será dividido da seguinte maneira, duas semanas para estudar mais sobre os processos da análise estática, duas semanas para estudar sobre métricas de análise, cinco semanas para o desenvolvimento da ferramenta, duas semanas para validação, três semanas para escrita do documento e uma semana para apresentação. O Quadro 1 ilustra este cronograma.

Quadro 1 – Cronograma

	Janeiro				Fevereiro				Março				Abril				Maio			
Fase	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
1		x	x																	
2				x	x															
3						x	x	x	x	x	x	x								
4													x	x	x					
5																x	x	x		
6																			x	

Fonte: Autor

Referências

AMER, H.; HAROUS, S. Smart-learning course transformation for an introductory programming course. In: *2017 IEEE 17th International Conference on Advanced Learning Technologies (ICALT)*. [S.l.: s.n.], 2017. p. 463–465. Citado na página 4.

ARIFI, S.; ZAHI, A.; BENABBOU, R. Semantic similarity based evaluation for c programs through the use of symbolic execution. In: . [s.n.], 2016. v. 10-13-April-2016, p. 826–833. Cited By 3. Disponível em: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-84994560588&doi=10.1109%2fEDUCON.2016.7474648&partnerID=40&md5=a90ebcef0a9b12e65ec0d2d645092630>. Citado na página 16.

ARIFI, S. M. et al. Automatic program assessment using static and dynamic analysis. In: *2015 Third World Conference on Complex Systems (WCCS)*. [S.l.: s.n.], 2015. p. 1–6. Citado 4 vezes nas páginas 5, 7, 9 e 10.

ARIFI, S. M. et al. Automated fault localizing and correction in dynamically analyzed programs. In: *2016 4th IEEE International Colloquium on Information Science and Technology (CiSt)*. [S.l.: s.n.], 2016. p. 587–592. Citado na página 7.

BRANDãO, L. de O.; RIBEIRO, R. da S.; BRANDãO, A. A. F. A system to help teaching and learning algorithms. In: *2012 Frontiers in Education Conference Proceedings*. [S.l.: s.n.], 2012. p. 1–6. Citado na página 4.

C, S. V. et al. Assessment of quality of program based on static analysis. In: *2019 IEEE Tenth International Conference on Technology for Education (T4E)*. [S.l.: s.n.], 2019. p. 276–277. Citado na página 5.

DEVMEDIA. *Guia Completo de Python*. [S.l.], 2020. Disponível em: <https://www.devmedia.com.br/guia/python/37024>. Citado na página 11.

JACKSON, D. A software system for grading student computer programs. *Computers Education*, v. 27, n. 3, p. 171–180, 1996. ISSN 0360-1315. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0360131596000255>. Citado 3 vezes nas páginas 15, 16 e 17.

LIMA, G. *Python: A origem do nome*. [S.l.], 2021. Disponível em: <https://www.alura.com.br/artigos/python-origem-do-nome/>. Citado na página 11.

POžENEL, M.; FüRST, L.; MAHNIč, V. Introduction of the automated assessment of homework assignments in a university-level programming course. In: *2015 38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. [S.l.: s.n.], 2015. p. 761–766. Citado na página 5.

RONGAS, T.; KAARNA, A.; KALVIAINEN, H. Classification of computerized learning tools for introductory programming courses: learning approach. In: *IEEE International Conference on Advanced Learning Technologies, 2004. Proceedings.* [S.l.: s.n.], 2004. p. 678–680. Citado na página 4.

ROVEDA, U. *O que é Python, para que serve e por que aprender?* [S.l.], 2020. Disponível em: <https://kenzie.com.br/blog/o-que-e-python/>. Citado na página 11.

WANG, T. et al. Ability-training-oriented automated assessment in introductory programming course. *Computers Education*, v. 56, n. 1, p. 220–226, 2011. ISSN 0360-1315. Serious Games. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0360131510002241>. Citado na página 16.

WESIAK, G.; AL-SMADI, M.; GÜTL, C. Towards an integrated assessment model for complex learning ressources: Findings from an expert validation. In: *2012 15th International Conference on Interactive Collaborative Learning (ICL)*. [S.l.: s.n.], 2012. p. 1–7. Citado na página 4.