UNIVERSIDADE DO VALE DO ITAJAÍ
CENTRO DE CIÊNCIAS TECNOLÓGICAS DA TERRA E DO MAR
CURSO DE CIÊNCIA DA COMPUTAÇÃO

CORREÇÃO AUTOMÁTICA DE ALGORITMOS NO ENSINO INTRODUTÓRIO DE PROGRAMAÇÃO

Informática na Educação

por

Fillipi Domingos Pelz

André Luis Alice Raabe, Dr.
Orientador

Itajaí (SC), Junho de 2011

UNIVERSIDADE DO VALE DO ITAJAÍ
CENTRO DE CIÊNCIAS TECNOLÓGICAS DA TERRA E DO MAR
CURSO DE CIÊNCIA DA COMPUTAÇÃO

CORREÇÃO AUTOMÁTICA DE ALGORITMOS NO ENSINO INTRODUTÓRIO DE PROGRAMAÇÃO

Área de Informática na Educação

por

Fillipi Domingos Pelz

Relatório apresentado à Banca Examinadora do Trabalho de Conclusão do Curso de Ciência da Computação para análise e aprovação.
Orientador: André Luis Alice Raabe, Dr.

Itajaí (SC), Junho de 2011

SUMÁRIO

LISTA DE ABREVIATURAS iv
LISTA DE FIGURAS v
LISTA DE TABELAS vi
RESUMO vii
ABSTRACT viii
1. INTRODUÇÃO 1
1.1 PROBLEMATIZAÇÃO 3
1.1.1 Formulação do Problema 3
1.1.2 Solução Proposta 3
1.2 OBJETIVOS 4
1.2.1 Objetivo Geral 4
1.2.2 Objetivos Específicos 4
1.3 METODOLOGIA 5
1.4 ESTRUTURA DO TRABALHO 7
2. FUNDAMENTAÇÃO TEÓRICA 8
2.1 INFORMÁTICA NA EDUCAÇÃO 8
2.1.1 Ensino de algoritmos e programação 8
2.2 CORREÇÃO AUTOMÁTICA DE ALGORITMOS 9
2.2.1 Análise dinâmica 10
2.2.1.1 Eficiência 11
2.2.1.2 Funcionamento 11
2.2.1.3 Cobertura por teste 12
2.2.2 Análise estática 12
2.2.2.1 Análise estrutural 13
2.2.2.2 Estilo e métricas de software 13
2.3 TRABALHOS SIMILARES 14
2.3.1 Primeira geração 15
2.3.2 Segunda geração 15
2.3.3 Terceira geração 16
2.3.4 Outros corretores 16
2.3.5 Trabalhos realizados no Grupo de Informática na Educação da UNIVALI 17
2.3.5.1 Portugol Núcleo e PortugolStudio 19
2.4 CONSIDERAÇÕES 25
3. ESTUDO PILOTO 27
3.1 CATALOGAÇÃO DE UM CONJUNTO DE QUESTÕES 27
3.2 DEFINIÇÃO DA MÉTRICA 29

4. DESENVOLVIMENTO 33
4.1 FERRAMENTAS UTILIZADAS 33
4.2 ALTERAÇÕES DO PORTUGOL 33
4.3 CORRETOR 40
4.3.1 Corretor Estrutural 40
4.3.2 Corretor Dinâmico 42
4.4 RESULTADOS 44
5. CONCLUSÕES 50
REFERÊNCIAS BIBLIOGRÁFICAS 51
APÊNDICE A - Lista de exercícios 56
APÊNDICE B – Ficha de Criérios 61

LISTA DE ABREVIATURAS

AST	Abstract Syntax Tree
HTML	HyperText Markup Language
IDE	Integrated Development Environment
TCC	Trabalho de Conclusão de Curso.
UNIVALI	Universidade do Vale do Itajaí.
XML	Extensible Markup Language

LISTA DE FIGURAS

Figura 1 - Metodologia utilizada. 5
Figura 2 - Módulos do Portugol Studio 19
Figura 3- Diagrama de classes da AST do Portugol. 22
Figura 4 - Exemplo de uma instância de AST que representa um programa escrito em Portugol. ... 24
Figura 5 – Visão geral do PortugolStudio versão 0.1. 25
Figura 6 - Gráfico de importância dos critérios definido pelos professores avaliadores. 30
Figura 10 - Gráfico de comparação das notas entregues pelos avaliadores e correção fictícia. 32
Figura 11- Classes para comunicação com o Portugol-Núcleo. 35
Figura 12 - Diagrama de classes do PortugolStudio 38
Figura 13 - Comparação das mudanças feitas na interface do usuário do Portugol Studio. 39
Figura 14 - PortugolStudio com corretor. 40
Figura 15 - Modelo de dados do corretor. 42
Figura 13 - Gráfico da média da importância dos critérios definido pelos dois avaliadores. 45
Figura 17 - Gráfico de comparação das notas entre avaliadores e corretor. 48

LISTA DE TABELAS

Tabela 1 - Propriedades das abordagens de correção, adaptado de Naudé (2007). 10
Tabela 2 - Geração do corretor x granularidade da correção, adaptado de Naudé (2007). 14
Tabela 3 - Exemplo de um programa escrito em Portugol. 23
Tabela 4 - Tipos de correção utilizados pelos corretores, adaptado de Naudé (2007). 25
Tabela 5 - Desvio padrão dos pesos dos critérios definidos pelos avaliadores. 30
Tabela 6 - Gabarito de correção. 31
Tabela 7 - Código fonte de integração do núcleo com o PortugolStudio 35
Tabela 8 - Código incluído para implementação do visitor. 36
Tabela 9 - Estrutura serializada. 41
Tabela 10 - Exemplo de XML de Questão. 43

RESUMO

PELZ, Fillipi. **Correção Automática de Algoritmos no Ensino Introdutório de Programação.** Itajaí, 2011. 41 f. Trabalho de Conclusão de Curso (Graduação em Ciência da Computação)–Centro de Ciências Tecnológicas da Terra e do Mar, Universidade do Vale do Itajaí, Itajaí, 2011.

O ensino de algoritmos é fundamental para as disciplinas iniciais dos cursos de computação (RAABE e DA SILVA, 2005). Aprender a programar não é simples, visto que há um alto o índice de evasão e relatos de dificuldades por parte dos alunos (LAHTINEN et. al, 2005). Raabe (2005) destaca que a abordagem mais utilizada no ensino de algoritmos é a resolução de problemas. Rahman e Nordin (2007) citam que devido a grande quantidade de tarefas que o professor comumente realiza, estas que vão desde a preparação do material à realização de pesquisas científicas, restringe-lo de passar por cada exercício entregue pelos seus alunos com a devida atenção. Um corretor automático pode auxiliar o papel do professor com um *feedback* ao aluno contendo uma nota e dicas para os possíveis erros. Neste trabalho foi proposto um método de avaliação de exercícios de algoritmos, este método foi utilizado por professores a fim de assemelhar suas correções e então compara-las com a nota entregue pelo corretor automático. Foi desenvolvido um corretor automático de algoritmos Portugol que avalia o funcionamento do programa escrito pelo aluno e as características estruturais do código que o aluno entrega para correção. A implementação do corretor consistiu em evoluir a ferramenta PortugolStudio que não havia sido utilizada em sala até então, a implantação da nova IDE PortugolStudio implicou na mudança da sintaxe Portugol lecionada em algoritmos 1 durante este TCC, esta nova sintaxe foi apontada por VALLE (2009) como mais adequada para a realidade da instituição por diminuir o impacto da migração da linguagem Portugol para as demais linguagens de programação lecionadas no restante do curso. Por fim foi realizada a coleta de respostas de exercícios dos alunos para comparação do corretor com os avaliadores, na comparação percebeu-se que o desempenho do corretor ficou abaixo do esperado, pela não utilização de mecanismos que calibram a nota do corretor, porem percebeu-se um potencial do corretor em ajudar os alunos com feedback para solução de algoritmos e a possibilidade deste em ajudar os alunos a desenvolver uma autoaprendizagem. Como trabalhos futuros identificou-se a melhora da acurácia do corretor, a implementação de mais mecanismos para dicas de soluções de exercício, a possibilidade de empacotar o PortugolStudio e um exercício como um objeto de aprendizado auto corrigível, por fim pode-se também comparar o desempenho de uma turma de controle comparada a uma turma que utilizou o corretor como ferramenta para prática de exercícios.

Palavras-chave: Informática na educação. Correção automática. Programação Introdutória.

ABSTRACT

Teaching algorithms is fundamental to the computer science course. (RAABE e DA SILVA,2005). Learning to program is not simple, introductory programming course having a high drop-out rates and claims by students (LAHTINEN et. al, 2005). Raabe (2005) cites the approach used in teaching algorithms is solving problems. Because of a range of tasks that usually the teacher performs, that go from making class content to scientific researches, this restricts the teacher from going through each exercise delivered by their students with proper attention (RAHMAN e NORDIN, 2007). An Automatic assessment can assist the teacher in feedback to the student with a grade. This study defines a tool to compare the assessment between assessors in order to approximate the results obtained by a mechanism of automatic correction with a correction performed by a teacher. In carrying out this undergraduate thesis one automated assessment system was developed to assess algorithms written in Portugol language that performs the functional evaluation of the student's program and structure of the algorithm. The development of the automated assessment system involved the improvement of the PortugolStudio IDE that haven't was been used in class room before this college thesis, because it was in stage of development by Lab of Solutions in Software. The PortugolStudio IDE uses a new syntax of Portugol language that was recommended by VALLE (2009) this led a realization of two collect data, because the first collect was made with another syntax of Portugol language. Finally we collected students' answers of the exercises and these were submitted to the assessment teachers and automated assessment system, after that the grade note was compared. The conclusion of this study is that the automatic correction developed failed in performance because it don't have mechanism to regulate the grade.

***Keywords**: Introductory programming. Automatic assessment. Algorithm.*

1. INTRODUÇÃO

O ensino de algoritmos é fundamental para as disciplinas iniciais dos cursos de computação como primeira etapa do desenvovimento do raciocínio lógico e para a pratica da programação (RAABE e DA SILVA, 2005).

Uma das etapas no aprendizado de programação é a elaboração de estruturas algorítmicas tais como laços e desvios condicionais. Após a demonstração e explicação destas estruturas o aluno interage com elas por meio de exercícios passados pelo professor a fim de praticar o que foi discutido em sala. Raabe (2005) destaca que a abordagem mais utilizada no ensino de algoritmos é a resolução de problemas pelos alunos.

O ambiente de estudos do aluno consiste em ferramentas para composição de códigos fonte onde ele consegue experimentar a linguagem de programação e verificar sua execução, essas ferramentas geralmente são integradas em um único ambiente de desenvolvimento, que geralmente contem um editor de texto, e um compilador.

O compilador realiza análises sintáticas e semânticas no código fonte do programa, e verifica se ele condiz com a estrutura da linguagem de programação. Os erros encontrados estão ligados com a má digitação do código fonte ou com a desatenção do programador.

A indicação de erros de imediato pelo compilador para o aluno pode ajudá-lo a gravar as palavras chaves da linguagem de programação e se o que foi escrito condiz com a estrutura base da linguagem, como abertura e fechamento de blocos, porém não é capaz de indicar ao aluno se ele está utilizando todas as estruturas exigidas pelo exercício, realizando os cálculos corretamente, utilizando nomes de variáveis adequados, identação do código, presença de comentários, etc.

Não há como o aluno saber de imediato se ele está resolvendo o exercício corretamente a não ser que o professor corrija o que foi elaborado pelo aluno e aponte o que o aluno deixou de fazer. Um corretor automático poderia apontar a visão do professor sobre o código fonte como, por exemplo, exigir a utilização de um laço de repetição para cumprir o exercício.

Duas abordagens são comuns para realização da correção automática, a estática e a dinâmica, onde a dinâmica verifica o resultado da execução do programa com um conjunto de entradas e saídas esperadas, já a abordagem estática faz verificações sem a execução do programa (RAHMAN e NORDIN, 2007).

A proposta deste trabalho é avaliar a acurácia destas técnicas de correção automática disponíveis na literatura, através do desenvolvimento de um corretor automático para algoritmos

escritos em Portugol , e compará-las com a correção manual realizada por professores e monitores da disciplina de algoritmos do curso de Ciência da Computação.

Para isso foi desenvolvido um corretor automático e definido um conjunto de questões que foram então realizados pelos alunos e corrigidas por um grupo de professores que atribuíram conceitos e notas a estas. A seguir os mesmos problemas e suas respectivas soluções foram submetidos à correção automática. Por fim foram comparados os resultados.

A correção de exercícios demanda muito tempo e dedicação dos professores e/ou dos monitores das disciplinas de Algoritmos I e II, o que sobrecarrega as pessoas envolvidas no ensino destas disciplinas. O corretor automático poderá viabilizar o aumento da quantidade de exercícios oferecidos aos alunos, e por fornecer ao aluno feedback da nota recebida, pode potencializar a autoaprendizagem.

O grupo de Informática na Educação da Univali vem desenvolvendo e aprimorando um ambiente para apoio à aprendizagem de algoritmos, denominado Alice. O ambiente é um sistema tutor inteligente que busca personalizar o atendimento aos alunos. Neste sentido a proposta deste trabalho também poderá contribuir para que o Ambiente Alice possa disponibilizar sempre novos exercícios aos alunos, considerando seus desempenhos em problemas anteriormente propostos. Isso não é possível atualmente no Alice, pois a correção ainda depende de trabalho humano, seja o monitor da disciplina ou o professor, o que torna o retorno ao aluno e a disponibilização de novas questões menos eficiente.

Destaca-se também um esforço no grupo de Informática na Educação da Univali em aplicar a nova sintaxe do Portugol desenvolvida para diminuir o impacto na migração da aula de algoritmos para as aulas de programação onde linguagens como PHP, C e Java são utilizadas, havia reclamações de dificuldades dos alunos em realizar essa transação, como destacado por VALLE (2009). Por esse motivo viu-se a possibilidade de durante esse trabalho de conclusão de curso, colocar em funcionamento a ferramenta PortugolStudio, cujo é uma IDE que estava em faze de desenvolvimento no Laboratório de Soluções em Software este utiliza-se de um interpretador com a nova sintaxe do Portugol, sintaxe essa que possuí elementos mais próximos às linguagens utilizadas nas disciplinas subsequentes a Algoritmos 1.

Com isso, um esforço realizado nesse trabalho também foi a continuação do desenvolvimento e a documentação do PortugolStudio e de seu interpretador Portugol para coloca-los em uso na disciplina de algoritmos. Além de ter sido desenvolvido um corretor de algoritmos Portugol e realizada a comparação com a correção dos professores.

1.1 PROBLEMATIZAÇÃO

1.1.1 Formulação do Problema

Nos cursos da área de Computação, dentre as várias disciplinas lecionadas no inicio do curso, a de algoritmos e programação destacasse pela alta evasão e reclamações de dificuldade de aprendizado.

Vários estudos em Informática na Educação são direcionados para o ensino de Algoritmos e Programação, principalmente nas turmas iniciais de Ciência da Computação e áreas afins. Estes estudos apontam como principal ferramenta para ensino de programação a prática através de exercícios.

Turmas de Algoritmos e Programação do primeiro semestre, geralmente são extensas, e Lahtinen et al. (2005) afirma que, muitas vezes, não há recursos suficientes e os alunos sofrem com a falta de instrução individual.

Muitas vezes a correção dos exercícios não pode ser feita de imediato, o que impossibilita ao aluno descobrir sua nota e obter um feedback de seu aprendizado, antes da avaliação que adiciona sua nota ao seu currículo semestral.

1.1.2 Solução Proposta

Será comparada uma combinação de técnicas de correção automática com o modelo de correção exercido pelos docentes das disciplinas da área da computação e que lecionam ou lecionaram Algoritmos e Programação inicial.

Espera-se então que além do corretor identificar se um algoritmo está correto ou incorreto, também seja possível verificar quão correto este algoritmo está, próximo ao que um professor faria, entregando uma nota correspondente ao grau de acerto do exercício de algoritmos.

1.2 OBJETIVOS

1.2.1 Objetivo Geral

Desenvolver e avaliar a acurácia de um corretor automático de algoritmos Portugol, comparando-o com correções feitas por professores.

1.2.2 Objetivos Específicos

- Catalogar algoritmos de correção existentes e selecionar as técnicas de correção que serão utilizadas;
- Definir uma métrica para comparação da correção manual e da automática;
- Desenvolver um corretor automático;
- Integrar os mecanismos de correção ao ambiente Portugol Studio[1];
- Comparar os resultados do mecanismo de correção automática com o resultado da correção manual; e
- Avaliar os resultados obtidos e apontar pontos fortes e melhorias possíveis.

[1] Portugol Studio é uma ferramenta em desenvolvimento no Laboratório de Soluções em Software da UNIVALI o qual ainda não foi divulgado em publicações.

1.3 Metodologia

Durante a elaboração deste Trabalho de Conclusão de Curso, percebeu-se a necessidade de modificar a metodologia descrita na pré-proposta, em função da nova sintaxe do Portugol, e pelo fato do PortugolStudio não estar finalizado durante a primeira etapa deste TCC. A decisão de implementar o corretor na nova sintaxe do Portugol, implicou na quebra da coleta e correção manual das provas realizadas pelos alunos em duas etapas, a piloto, realizada no TCCI e a final durante o TCCII.

O novo escopo do TCC está demonstrado pela Figura 1, sendo que o TCC I consistiu em elaborar o processo de coleta e análise comparativa englobando: 1. Análise de algoritmos de correção e trabalhos similares; 2. Elaboração de gabarito de correção; 3. Definição de problemas e classes de problemas; e 4. Validação do modelo de correção. O TCC II consistiu na implementação e validação do corretor automático passando pelas etapas: 1. Finalização do PortugolStudio; 2. Codificação do corretor; 3. Coleta de provas com a nova sintaxe; e 4. Comparação das correções de professores e a correção realizada pelo software.

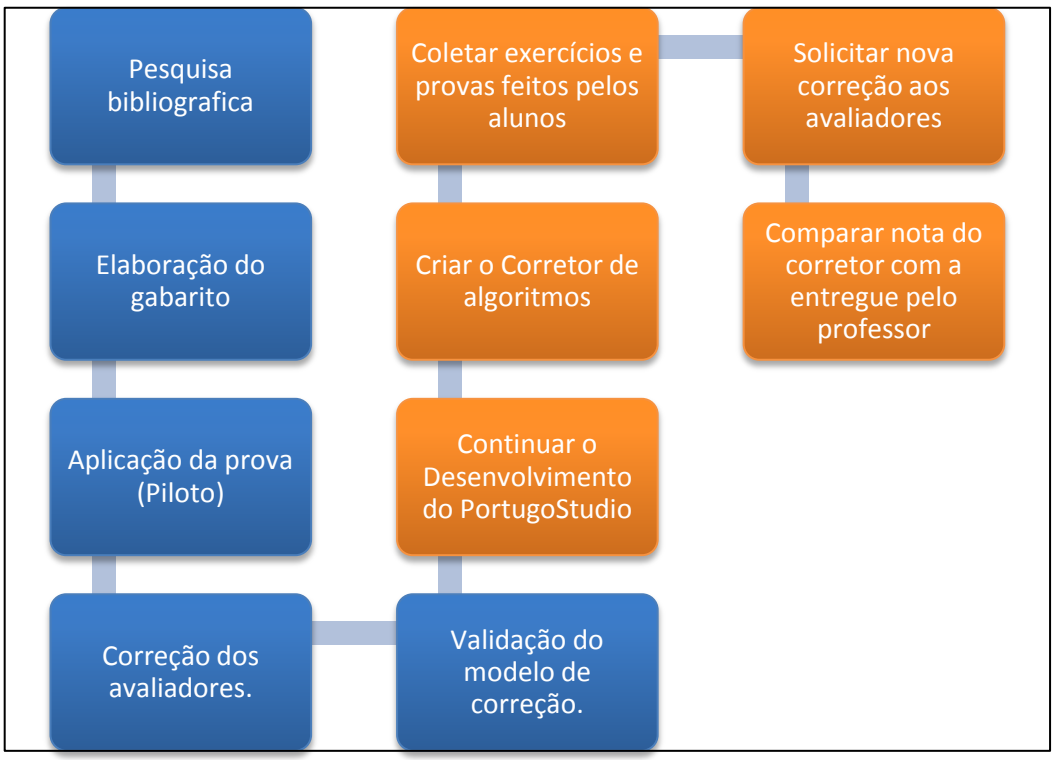

Figura 1 - Metodologia utilizada.

Inicialmente os exercícios e correções manuais seriam realizados somente no TCC I, para no TCCII apenas comparar os valores da correção manual com os valores retornados pelo corretor automático, porém, com a utilização da nova sintaxe do Portugol, os alunos do primeiro período em vigência juntamente com a elaboração deste TCC, estavam estudando através da sintaxe antiga do Portugol, impossibilitando a utilização de suas respostas no corretor automático, para testa-lo. Visto isso, fez-se necessário uma nova coleta de respostas, já utilizando a nova sintaxe do Portugol, e uma nova correção dessas provas também foi necessária por parte dos avaliadores.

Para a elaboração do TCC I, realizaram-se estudos na bibliografia para localização de classificações de problemas algorítmicos lecionados em Algoritmos e Programação do semestre inicial de Ciência da Computação e áreas afins, identificando o foco da disciplina, além da busca por questões e questionários comumente utilizados para avaliar os alunos do primeiro ano do curso de Ciência da Computação.

Fez-se uma seleção de questões que foram utilizadas como benchmark do corretor automático, questões estas que foram aplicadas a um grupo de controle e avaliadas por um grupo de professores. A correção dessas questões foi posteriormente comparada com a correção realizada pelo software de correção automática.

Para o teste piloto, criou-se um gabarito e utilizou-se de uma prova realizada pela turma de Algoritmos e Programação I ministrada pelo professor André Luís Alice Raabe. Após foi solicitado a um grupo de avaliadores a correção destas provas utilizando um gabarito.

Ao final documentou-se a situação atual do desenvolvimento do PortugolStudio, e apresentou-se o a implementação do corretor, e compararam-se os resultados entregues pelo corretor com os obtidos nas correções dos professores. Podendo assim documentar melhorias possíveis na implementação do corretor automático.

1.4 Estrutura do trabalho

Este documento está estruturado em quatro capítulos. O Capítulo 1, Introdução, apresentou uma visão geral do trabalho. No Capítulo 2, Fundamentação Teórica, é apresentada uma revisão bibliográfica sobre: o ensino introdutório de programação, mecanismos de correção automática, assim como uma análise a respeito de trabalhos similares. O Capítulo 3 apresenta o estudo piloto onde é definido o instrumento de teste e avaliação e apresenta os resultados obtidos no teste piloto de correções. O capítulo 4 apresenta as modificações realizadas no PortugolStudio e Portugol-Núcleo além de descrever como foi implementado o sistema de correção de algoritmos, concluindo, o Capítulo 4 está a comparação entre o desempenho do corretor com a nota entregue pelos professores e as melhorias que podem ser realizadas no corretor para que este melhore seu desempenho. No final deste texto é apresentada a conclusão, contendo o resgate dos objetos e trabalhos futuros. O texto ainda inclui um apêndice que complementa as informações apresentadas no trabalho.

2. FUNDAMENTAÇÃO TEÓRICA

Neste capitulo são discutidos os principais temas de interesse deste Trabalho de Conclusão de Curso, sendo eles: o cenário atual da informática na educação dentro da UNIVALI, a dificuldade de aprendizado e ensino de Algoritmos e Programação, Técnicas de correção automática de algoritmos, e ao final são apresentados trabalhos similares.

2.1 Informática na Educação

2.1.1 Ensino de algoritmos e programação

Algoritmos eficientes e estruturas de dados são fundamentais para qualquer aplicativo computacional. Por essa razão esses tópicos estão presentes no currículo dos cursos de Ciência da Computação (SCHEFFLER, 2008).

A comunidade de informática na educação tem investido uma quantidade considerável de energia estudando o desempenho dos programadores iniciantes, pois a disciplina de Algoritmos e Programação I sofre um desgaste muito grande com alunos que optam por não concluir a disciplina (SCHULTE e BENNEDSEN, 2006 apud PETERSEN et. al, 2011).

Para Lahtinen et. al (2005) programar não é fácil de aprender, pois requer o entendimento correto de conceitos abstratos onde muitos alunos tem dificuldade devido à natureza do assunto. Além disso, muitas vezes, não há recursos suficientes e os alunos sofrem com a falta de instrução individual. Há também grupos de estudantes grandes e heterogêneos e, portanto, o aumento na dificuldade em conceber o ensino de modo que seja benéfico para todos os alunos. Isso muitas vezes leva a altas taxas de abandono nos cursos de programação.

Kinnunen e Malmi (2006) relatam que independente da conhecida importância em aprender programação introdutória os resultados são decepcionantes, muitas instituição de ensino relatam deficiências no aprendizado de programação por parte dos estudantes. Este problema origina-se em equívocos nos estudos iniciais. O mau entendimento de conceitos básicos dificulta o avanço nos estudos. Uma consequência do mau aprendizado é o relato por parte das instituições de ensino de uma evasão de 20 a 40 por cento, ou as vezes mais, de estudantes nas matérias de programação introdutória.

Muitos dos alunos já utilizam computadores antes de ingressar na computação, mas a maioria deles ingressão sem saber nada de programação (DALY e WALDRON, 2004). Saikkonen et al, (2005) cita que durante as aulas de programação os alunos realizam vários pequenos exercícios, para ensiná-los a escrever seus primeiros programas.

O modelo de aprendizado mais citado de nas publicações de correção automática é a taxonomia de Bloom. A taxinomia lista: conhecimento, compreensão, aplicação, análise, síntese e avaliação, como aspectos cognitivos de aprendizado (TSINTSIFAS, 2002).

Exercícios de resposta livre são adequados para avaliar os mais altos níveis da taxonomia de Bloom, especificamente: a aplicação de conhecimento, análise, síntese e avaliação. Geralmente os exercícios de programação são quase que exclusivamente de resposta livre (TSINTSIFAS, 2002).

2.2 Correção automática de algoritmos

Um sistema de correção automática de algoritmos é uma aplicação auxiliada por computador para verificar, avaliar ou mesmo para dar nota a um exercício de programação. A principal razão para o desenvolvimento de todos os sistemas de correção automática é facilitar ao professor de programação verificar e avaliar o desempenho dos estudantes da turma de Algoritmos (RAHMAN *et al*, 2009).

Uma das vantagens mais significativas da correção automática é a diminuição da carga de trabalho dos professores de Algoritmos. A correção manual de algoritmos é um trabalho tedioso e repetitivo que consome tempo. Professores precisam conduzir as aulas, preparar materiais, provas e exercícios, acompanhar o aprendizado dos alunos, participar de pesquisas em seu laboratório, escrever artigos científicos, apresentar resultados de pesquisa, participar de seminários e conferências, entre outros. Essas atividades restringem os professores de passar por cada exercício entregue pelos seus alunos com a devida atenção (RAHMAN e NORDIN, 2007).

As três principais funcionalidades de um corretor automático são: submissão, correção, e feedback. Alguns corretores possuem mais funcionalidades como detecção de plágio, gerenciamento de especificação e entrega de exercícios, administração de notas da disciplina, entre outras (NAUDÉ, 2007).

Segundo Ala-Mutka (2005) há duas abordagens principais utilizadas em correção automática de algoritmos. A primeira se faz através da abordagem dinâmica, necessitando da

execução do algoritmo para determinar sua exatidão. A segunda abordagem é a estática onde o algoritmo é corrigido sem ser executado.

A análise dinâmica só pode ser realizada compilando e executando o código fonte, o que implica em somente ser possível analisar algoritmos sem erros de sintaxe. Contudo a análise estática corrige o programa mesmo que possua erros de sintaxe que impossibilitem a execução pelo compilador (RAHMAN e NORDIN, 2007).

Segundo Naudé (2007) há uma variedade de propriedades de avaliação utilizadas nos corretores automáticos, essas propriedades descrevem o que pode ser obtido através da análise estática ou dinâmica, e serem aplicadas em ambas as granularidades: todo o programa ou uma parte dele. A Tabela 1 cruza as principais categorias de correção com as propriedades que são avaliadas.

Tabela 1 - Propriedades das abordagens de correção, adaptado de Naudé (2007).

	Estática	**Dinâmica**
Eficiência		X
Funcionamento	X	X
Cobertura de teste	X	X
Características estruturais	X	
Estilo de programação	X	
Complexidade	X	
Métricas de software	X	

2.2.1 Análise dinâmica

A análise de algoritmos utilizando as técnicas da abordagem dinâmica envolve a execução do código fonte. A execução de código gera saída de dados que podem ser comparados com saídas de controle providos pelo professor. A fim de determinar a exatidão e eficiência do código, a execução é feita utilizando um conjunto de dados de teste (RAHMAN & NORDIN, 2007).

Para Rahman e Nordin, (2007), a necessidade de executar o algoritmo, implica em expor o software de correção automática a falhas ocasionadas pelo código fornecido pelo aluno, como laços

infinitos ou algum erro crítico em tempo de execução, que por sua vez ocasionam o travamento do sistema.

Os principais aspectos do programa que são valiosos para avaliar na abordagem dinâmica são eficiência de execução e funcionamento válido. A cobertura por testes pode ser realizada utilizando ambas as abordagens dinâmica e estática, porém ela é mais comumente feita através da abordagem dinâmica (NAUDÉ, 2007).

2.2.1.1 Eficiência

Segundo Naudé (2007) a medida da eficiência do algoritmo do aluno é utilizada por um dos primeiros corretores automáticos, o ASSYST (FORSYTHE; WIRTH, 1965 apud NAUDÉ, 2007). Porem eles relataram ineficácia em medir a qualidade do programa do aluno através da medida da eficiência.

Um dos problemas encontrados na implementação dessa medida no ASSYST foi que a maioria dos programas simples são executados muito rapidamente e a diferença de tempo é demasiadamente pequena para ser considerada significativa (NAUDÉ, 2007).

Naudé (2007) destaca que o valor pedagógico em medir a eficiência do tempo de execução é incerto. Na maioria das correções é improvável que isso seja relevante, particularmente no ensino introdutório de programação. Os programas podem apenas precisar de uma medida de tempo limite de execução com o intuito de identificar se não entraram em um loop infinito.

2.2.1.2 Funcionamento

Para Naudé (2007) o funcionamento de um programa computacional é provavelmente o mais visível critério sobre o qual ele pode ser julgado. Tanto que é um dos meios mais populares para avaliar o programa dos alunos automaticamente.

Verificar o funcionamento correte de um programa depende criticamente de duas coisas: prover um conjunto de dados de teste ao rodar o programa do aluno e capturar as saídas geradas e validar a acurácia destas saídas. Enquanto a maioria dos corretores automáticos utiliza entrada e saída para se comunicar com o programa, os corretores mais recentes utilizam técnicas como Java reflection para invocar métodos e acessar os valores brutos diretamente nas variáveis (NAUDÉ, 2007).

Naudé (2007) destaca que acessar valores brutos como valores booleanos, números reais, ou estruturas de dados complexas, é mais simples que acessar textos no fluxo de saída como realizado por muitos corretores automáticos. Textos de saída geralmente introduzem variações desnecessárias na formatação dos dados. Para conter o problema de formatação, soluções têm sido propostas como filtrar os espaços em branco e utilizar expressões regulares ou uma gramática formal para descrever a saída do programa.

2.2.1.3 Cobertura por teste

Segundo Naudé (2007) a cobertura por teste não só requer que o aluno envie o programa que soluciona o exercício como também exige um conjunto formalizado de casos de teste. Para Naudé isso tem claras vantagens pedagógicas, pois prove uma oportunidade de introduzir as práticas atuais da indústria em aplicar testes unitários. E também exige que os alunos considerem o comportamento de seus softwares sob uma variedade de dados de entrada.

Algumas problemáticas podem acontecer ao avaliar a cobertura por teste. Uma delas é que o estudante consegue um escore 100% de cobertura de testes mesmo que os testes sejam realizados em um programa que não resolve o exercício por completo. Outro problema é que, mesmo que o aluno faça um conjunto considerável de testes e eles passem, o aluno pode receber menos de 100% na cobertura de teste, isso acontece porque pode haver algum código inacessível aos testes (NAUDÉ, 2007).

Para Naudé (2007) um problema geral ao exigir dos alunos um conjunto de testes completo é que não dão valor ao importante que os testes unitários têm no desenvolvimento de software confiável. Uma abordagem para melhorar o interesse dos alunos nos testes unitários é dar nota se os casos de teste expuserem falhas nos programas dos colegas de classe (GOLDWASSER, 2002 apud NAUDÉ, 2007).

2.2.2 Análise estática

Segundo Rahman, et. al (2007) há várias técnicas categorizadas como pertencentes a abordagem de análise estática de correção automática de algoritmos. Destacam-se os seguintes métodos de avaliação: avaliação do estilo de codificação, detector de erros de programação, medição de métricas de software, avaliação de projeto, detector de palavra chave, detector de plágio, análise de similaridade estrutural e análise de similaridade não estrutural.

Truong et al (2004) destaca que os métodos de análise estática de algoritmos variam de comparação de *Strings* de código fonte, como maneira mais simples, a comparação do grafo que representa o programa em memória, como maneira mais complexa de análise.

Segundo Rahman & Nordin (2007 apud Jesus 2010) em alguns casos se faz necessário, devido à complexidade da solução do exercício, a definição de vários esquemas de gabarito, de maneira que as possíveis variações de solução feita pelo aluno sejam contempladas pelo mecanismo de correção.

2.2.2.1 Análise estrutural

Naudé (2007) define duas análises feitas sobre a estrutura do código fonte sendo similaridade estrutural e descoberta de características. Na similaridade estrutural é definido se a estrutura de um programa está de acordo com o que é esperado em um programa que soluciona o exercício. A descoberta de características trata-se da busca de um padrão de característica estrutural dentro de um subconjunto da estrutura de um programa. A análise estrutural depende de uma representação não linear do programa, como parse tree, abstract syntax trees (AST), ou estruturas de dados similares.

Chanon (1966, apud Naudé, 2007) apresentou um dos primeiros corretores automáticos com comparação de estrutura de programa, sua intenção era encontrar plágio no código fonte. Seus estudos mostraram que há um esforço computacional significativo ao realizar a comparação de estruturas de programas

Embora alguns corretores automáticos utilizem comparação de estrutura, esta técnica é tipicamente utilizada para feedback em vez de ser utilizada para pontuar o exercício. De forma similar a técnica de descoberta de características, comumente procura certas estruturas dentro do código, mas sem se preocupar com a pontuação que o aluno vai receber (Naudé, 2007).

2.2.2.2 Estilo e métricas de software

Segundo Naudé (2007) estilo e métricas de software são características sobre o código fonte do programa que podem ser facilmente mensuradas. Rees (1982 apud Naudé, 2007) identificou dez métricas simples para julgar a qualidade de um programa, cinco dessas métricas se preocupam com a formatação enquanto que as outras cinco se preocupam com identificadores e com a estrutura do programa.

As dez métricas são: tamanho da linha, densidade de comentários, identação, espaços em branco, espaço embutido, decomposição do código, palavras reservadas, tamanho do nome das variáveis, quantidade de variáveis distintas, e quantidade de rótulos e *goto*.

Para Naudé (2007) métricas de software podem ser capazes de corrigir algoritmos, mas poucos corretores automáticos tem tirado proveito. Métricas de software parecem relevantes no cenário educacional, no entanto elas por si só são apenas valores numéricos, a dificuldade está em interpretar esses valores no contexto do ensino.

2.3 Trabalhos similares

Douce et. al (2005) revisou os corretores automáticos encontrados na literatura e os classificou em três gerações de corretores automáticos de acordo com sua época de desenvolvimento e suas características. A primeira geração é caracterizada primeiros esforços publicados em correção automática e operação de baixo nível, enquanto que a segundo geração é são tipicamente orientada a linha de comando e mais geral. Douce descreve a terceira geração como corretores baseados na plataforma web, a segunda geração continua em desenvolvimento paralelamente.

A Tabela 2 demonstra alguns dos corretores automáticos classificados por Douce et. al (2005) e a granularidade da correção realizada pelos corretores automáticos, é possível observar que a terceira geração de corretores passou a diminuir a granularidade da correção.

Tabela 2 - Geração do corretor x granularidade da correção, adaptado de Naudé (2007).

	Hollingsworth	ASSYST	BOSS	Kassandra	TRY	**PortugolStudio**	CourseMaster	ELP	HoGG	PASS	Scheme-robo
Geração	I	II	II	II	II	II	III	III	III	III	III
Todo o programa	X	X	X	X	X	X	X		X		
Função									X	X	X
Declaração / Expressão							X	X			X

2.3.1 Primeira geração

Segundo Douce et. al (2005) um dos mais antigos exemplos de corretor automático de algoritmo pode ser encontrado em Hollingsworth (1960). Os estudantes submetiam programas escritos em linguagem *assemby* em cartões perfurados, o corretor rodava o programa do aluno e devolvia duas respostas diferentes: “resposta errada” ou “programa completo”.

Outro exemplo de corretor de primeira geração destacado por Douce et. al (2005) é um corretor que examina programas escritos em Algol desenvolvido por Forsythe e Wirth (1965), este possuía três funcionalidades que eram suprir dados de testes para o exercício, acompanhar o tempo de execução do programa e um controle das notas.

Para o desenvolvimento dos corretores automáticos de primeira geração era necessário modificações no compilador e no sistema operacional. Hext e Winnings (1969 apud Douce et. al, 2005) programaram um corretor que comparava dados de teste armazenados com os dados obtidos através da execução dos programas dos alunos. Após a execução da correção um relatório era gerado, incluindo resultados detalhados dos testes.

2.3.2 Segunda geração

Segundo Douce et. al (2005) a segunda geração de corretores automáticos são caracterizados pela separação do corretor em diferentes ferramentas de linha de comando. Um bom exemplo de uma de um antigo corretor de segunda geração é o TRY (REEK, 1989). Este consiste em um conjunto de scripts Shell que se encarregam em das tarefas de interagir com o compilador, construir o programa do aluno e testa-lo. Uma das principais diferenças com a primeira geração, é que o corretor era utilizado pelos alunos e não pelo professor, provendo assim um feedback imediato.

Jackson e Usher (1997 apud NAUDÉ, 2007) descrevem o ASSYST, um dos primeiros a possuírem interface gráfica, possuía a submissão e a correção em tarefas separadas. ASSYST adicionou um valor pedagógico ao requerer dos alunos a submissão de seus próprios valores de teste. Isso introduz ao aluno o conceito de cobertura de código, exigindo que eles pensem cuidadosamente sobre seu código e sua vulnerabilidade em casos específicos.

Outros corretores similares ao ASSYST podem ser considerados como de segunda geração como o Ceilidh, BOSS, TRAKLA, e Robo-prof, eles podem ser classificados como de segunda

geração por não serem totalmente web, por esse motivo, atualmente ambas as gerações, segunda e terceira, estão em desenvolvimento (NAUDÉ, 2007).

2.3.3 Terceira geração

Douce et. al (2005) classificou os corretores de terceira geração aqueles que são baseados na plataforma web e que possuem de forma geral as técnicas mais recentes de correção automática de algoritmos.

Segundo Douce et. al (2005) um exemplo da terceira geração de corretores é o CourseMaster que analisa os programas dos alunos através de diversos critérios com ênfase em explorar o projeto do código. Este também se destaca pelo rico feedback provido ao aluno, permitindo aos alunos identificar quais partes do exercício deixou de fazer corretamente.

CourseMaster é um sistema cliente servidor para auxilio nas matérias de programação. Ele prove funcionalidades de correção automática para os estudantes em Java e C++ e administra as notas resultantes das correções, soluções e materiais da disciplina. O estudante consegue submeter um algoritmo para avaliação e recebe um feedback imediato. (TRUONG et al, 2004).

Naudé classifica o Scheme-Robo como um corretor automático de terceira geração e destaca que este possui ambas as técnicas estática e dinâmica e também por este conseguir testar partes individuais do programa do aluno.

Saikkonen et. al (2001) cita um sistema que realiza correções com técnicas dinâmicas juntamente com a detecção de plágio, denominado Scheme-Robo. Na análise da estrutura do algoritmo entregue pelo aluno, ele utiliza uma arvore sintática abstrata.

2.3.4 Outros corretores

Encontraram-se outros corretores na literatura e preferiu-se não classifica-los em nenhuma das gerações destacadas por Douce et. al (2005), nesta sessão são encontrados dois corretores automáticos AutoLEP e EscrachoBot.

Wang *et al* (2010) demonstram um sistema construído utilizando a combinação das técnicas estática e dinâmica denominado AutoLEP. Para Wang *et al*. (2010) a combinação dessas técnicas possibilita a correção de programas com erros sintáticos e lógicos, além de reconhecer se o algoritmo do aluno cumpre com as especificações do exercício.

O funcionamento do AutoLEP é realizado em três passos. Primeiro o algoritmo do aluno e os gabaritos do exercício são transformados em um grafo de dependências. Segundo, são realizadas transformações nos grafos de dependências dos modelos e resposta do aluno preservando seu significado semântico. Finalmente uma pontuação é recuperada através do cálculo da normalização do grafo de dependência do algoritmo do aluno com os grafos das soluções modelo (Wang et al. 2010).

Segundo Wang et al. (2010) o sistema AutoLEP foi bastante aceito pelos alunos, monitores e professores, ajudou os monitores e professores a aumentar a qualidade do ensino de programação e reduziu a carga de trabalho. Ainda segundo eles o AutoLEP promoveu o interesse em programar, aumentou as habilidades de programação e a capacidade de autoaprendizagem dos alunos.

Jesus (2010) utiliza técnicas de correção automática dentro de um jogo de soluções de problemas algorítmico denominado EscrachoBot para identificar se o aluno passou ou não pelo exercício. O corretor utiliza de análise dinâmica e estática. A análise estrutural do algoritmo é realizada através da serialização da AST do programa e concentra-se na descoberta de características, comparando o algoritmo do aluno com a solução ideal.

2.3.5 Trabalhos realizados no Grupo de Informática na Educação da UNIVALI

Na UNIVALI existe um grupo de informática na educação que atua desde 2000 promovendo pesquisa focada na utilização e desenvolvimento de tecnologias educacionais, este foi formalizado como grupo no diretório do CNPq em 2002 e possuí pesquisadores do Mestrado em Computação Aplicada, Mestrado Acadêmico em Educação, Bacharelado em Ciência da Computação e Tecnólogo em Sistemas para Internet.

Um dos esforços do grupo de informática na educação é o ensino de algoritmos e programação, neste sentido já publicou e desenvolveu diversos artigos e softwares para auxiliar a disciplina de Algoritmos e Programação I. Dentre esses softwares destacam-se os seguintes: 1) ALICE (SILVA; RAABE, 2005), (RAABE; GIRAFFA, 2006) e (VAHLDICK; RAABE, 2008), 2) Bipide (VIEIRA; ZEFERINO; RAABE, 2009) e (MORANDI; et. al, 2006), 3) Escracho (JESUS, 2010), 4) EscrachoBot (JESUS, 2010), e 5) WebPortugol (HOSTINS; RAABE, 2007) e (RAABE; DAZZI; SANTIAGO, 2007).

O ambiente ALICE é um software web desenvolvido por Silva (2005), para colaborar com a participação extraclasse dos alunos através da disponibilização de materiais didáticos e ele também

é um sistema tutor inteligente que disponibiliza exercícios aos alunos conforme seu desempenho na disciplina e indica ao aluno quais conceitos ele deve estudar mais, auxiliando assim o professor na tarefa de acompanhar o aluno. Contudo no ALICE a correção dos algoritmos entregues nos exercícios é feita manualmente para então o ALICE entregar um novo exercício ao aluno.

Existe um contínuo interesse no desenvolvimento de pesquisas relacionadas ao ambiente ALICE, neste sentido destaca-se a adição ao suporte à importação e execução de objetos de aprendizagem no padrão SCORM, através do componente CELINE desenvolvido por Vahldick (2008), tornando o ALICE um repositório de objetos de aprendizagem, possibilitando assim a migração dos exercícios de texto simples para objetos SCORM, o que facilita a reutilização dos exercícios por outros repositórios SCORM.

O Bipide é um ambiente de desenvolvimento integrado de caráter educacional, desenvolvido por Vieira et. al (2009) para ser utilizado nas disciplinas de Algoritmos e Programação I e demais disciplinas de introdução à Ciência da Computação onde visa à redução das abstrações dos conceitos iniciais do ensino de programação e a assimilação de conceitos de Arquitetura e Organização de Computadores envolvidos na programação, simulando algoritmos escritos em Portugol sobre um processador educacional denominado BIP (MORANDI; et al, 2006).

As principais funcionalidades do Bipide são: um editor de código fonte Portugol, identificação de palavras reservadas e símbolos da linguagem, identificação de erros de compilação, execução passo a passo, simulação gráfica das aplicações sobre a arquitetura dos processadores BIP I e BIP II, simulador de instruções dos processadores BIP, geração de códigos assembly, binário e hexadecimal, e geração de códigos MIF (ROM e RAM) e VHDL para os processadores BIP e para o microcontrolador µBIP.

Escracho é um ambiente de programação que possibilita a construção de pequenos algoritmos arrastando e encaixando itens que representam comandos de programação a fim de diminuir a barreira exposta pela sintaxe de programação, este foi desenvolvido por Jesus (2010) e inspirado pelo Scratch desenvolvido pelo MIT. O ambiente Escracho possui um mecanismo de correção automática de algoritmos realizando a avaliação da estrutura do programa, das entradas e saídas esperadas e a presença de instruções chave para conclusão do exercício entregue pelo aluno.

Desenvolvido por Jesus (2010) o EscrachoBot é um jogo educacional focado na resolução de problemas algorítmicos com preocupação em aumentar a contextualização destes problemas na expectativa de aumentar o interesse dos alunos em resolvê-los, a programação dentro do EscrachoBot é realizada utilizando o Escracho e ambos fazem parte de uma pesquisa para avaliar a utilização de jogos no ensino de programação.

WebPortugol é um interpretador da linguagem de programação Portugol executável em web browser utilizando Java Applet no qual auxilia o desenvolvimento de algoritmos, este foi desenvolvido por Hostins (2007) e atualmente está integrado ao sistema ALICE para os alunos resolverem os exercícios, conta com um espaço para enunciados, um console para entrada e saída, um componente para exibição de erros de compilação e a possibilidade de executar o algoritmo do aluno no modo passo a passo, para ajudar o aluno a encontrar possíveis erros no código.

No atual cenário do grupo de informática na educação o presente trabalho se alinha no interesse em aumentar o feedback e o suporte extraclasse ao aluno da disciplina de Algoritmos e Programação I através da possibilidade de futuramente o corretor automático corrigir os exercícios do ambiente ALICE, e também colabora na evolução da sintaxe do Portugol para assemelha-lo as demais linguagens utilizadas nos cursos de Ciência da Computação e Sistemas para Internet como C, Java e PHP, ao finalizar o desenvolvimento do PortugolStudio e do interpretador do Portugol 2.

2.3.5.1 Portugol Núcleo e PortugolStudio

O PortugolStudio é uma IDE - *Integrated Development Environment* (Ambiente de desenvolvimento integrado) que está em fase final de desenvolvimento no Laboratório de Soluções em Software e visa dar suporte a nova sintaxe do Portugol. Desenvolvido na plataforma Java, contém um interpretador Portugol e a interface gráfica em módulos separados, como demonstra a Figura 2. O interpretador possui uma arvore sintática abstrata (AST - Abstract Syntax Tree), um analisador sintático e um semântico. O PortugolStudio possui um editor, um console de execução e um componente que exibe a lista de erros devolvidos pelo interpretador.

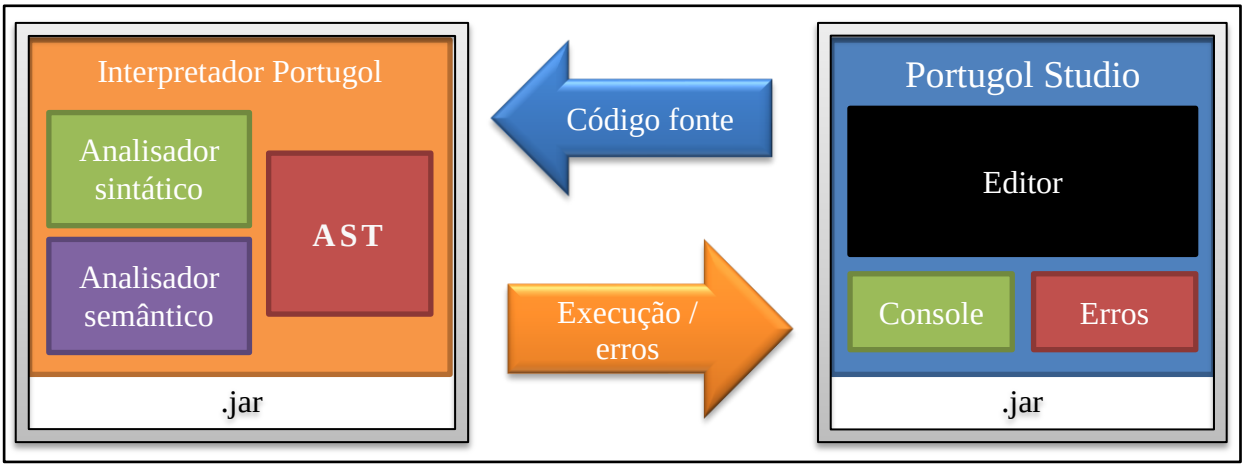

Figura 2 - Módulos do Portugol Studio

O analisador sintático é responsável por verificar se uma cadeia de símbolos pode ser gerada por uma gramática específica, normalmente transformando um texto em uma estrutura de dados denominada arvore sintática abstrata. A análise sintática também se preocupa em verificar se a ordem da construção do código fonte está correta. Por exemplo, o comando Enquanto deve sempre vir acompanhado de uma expressão condicional, o mesmo acontece com os comandos Se e Senão. Se nenhum erro é encontrado, o analisador sintático consegue construir a AST com sucesso.

Uma arvore sintática abstrata (AST) é uma representação em forma de árvore da estrutura sintática do código fonte escrito em uma linguagem de programação. Cada nó da arvore representa uma construção que ocorre no código fonte (PARR, 2009).

A análise semântica é responsável por verificar se a estrutura construída pelo analisador sintático pode ser executada. Durante a análise semântica são verificadas regras sensíveis ao contexto, que não são possíveis de serem verificadas durante a análise sintática, como por exemplo, regras de escopo, regras de visibilidade e consistência de tipos.

A AST do Portugol é orientada a objeto e a principal hierarquia está demonstrada na Figura 3. Cada linha do código fonte é transformada em um objeto que representa sintaticamente parte ou a linha como um todo, e os relacionamentos entre os objetos que compõem a AST são primordiais para colaborar com a abstração da sintaxe.

Toda a linha de código fonte no Portugol é uma generalização de um NoBloco, isso faz com que em certos momentos qualquer tipo de construção da linguagem seja aceita em uma linha de código, com isso no Portugol não é possível prever qual será a linha declarada após uma declaração de função por exemplo, o mesmo acontece em linguagens como o C, o C++, o Java, o PHP e similares. Esse tipo de construção é possível porque na AST em certos momentos o nó esperado é do tipo NoBloco, o topo da hierarquia de objetos que representam nós da árvore sintática abstrata. Os objetos que herdam diretamente do NoBloco são NoDeclaração, NoExpressao, NoPara, NoSe, NoEscolha, NoFacaEnquanto, NoEquanto, NoCaso, NoRetorne, NoPercorra e NoPare, esses objetos conseguem representar e especificar a maior parte das construções aceitas pelo Portugol, restando apenas os objetos que herdam de NoDeclaracao e NoExpressao, pois ambas as classes são abstratas.

Qualquer tipo de expressão que venha a acontecer no Portugol tem como topo da hierarquia a classe NoExpressao, isso permite vários tipos de construções em uma expressão, como uma

operação matemática, uma referência a uma variável na tabela de símbolos, ou um valor constante. Fazem parte da hierarquia de NoExpressao as classes NoOperacao, NoReferencia, NoIncremento, NoDecremento, NoMenosUnario, NoNao, NoLogico, NoInteiro, NoReal, NoCaractere, NoCadeia, NoVetor e NoMatriz, nessa hierarquia as expressões matemáticas são representados por objetos do tipo NoOperacao, NoIncremento e NoDecremento, os valores que podem aparecer em uma expressão são representados por objetos NoReferencia, quando dentro da expressão acontece o acesso a valores de uma variável, ou um objeto equivalente ao tipo de dado do valor constante passado à expressão. O NoChamadaFuncao estende NoReferencia por ser um tipo de referência.

Toda as declarações que ocorrem na linguagem Portugol são representadas por um objeto da hierarquia de NoDeclaracao na AST, existem quatro tipos de declarações possíveis sendo elas a declaração de variável, função, vetor e matriz, com isso existe uma classe para cada tipo, sendo elas NoDeclaracaoVariavel, NoDeclaracaoFuncao, NoDeclaracaoVetor e NoDeclaracaoMatriz, essa distinção de objetos dentro da AST é necessária porque a forma dessas declarações são distintas entre si na linguagem Portugol.

A raiz da AST é um objeto do tipo ArvereSintaticaAbstrata que agrega objetos NoDeclaracao, essas declarações pertencem ao escopo global do programa, e os objetos que abrem um novo escopo NoDeclaracaoFuncao e NoPara, por exemplo, agregam objetos do tipo NoBloco. ArvoreSintaticaAbstrataBiblioteca é uma AST que no Portugol representa declarações fora do programa, similar ao “.h” do C, enquanto que a classe ArvoreSintaticaAbstrataPrograma representa o programa principal e só é possível uma instância dela em uma AST no Portugol.

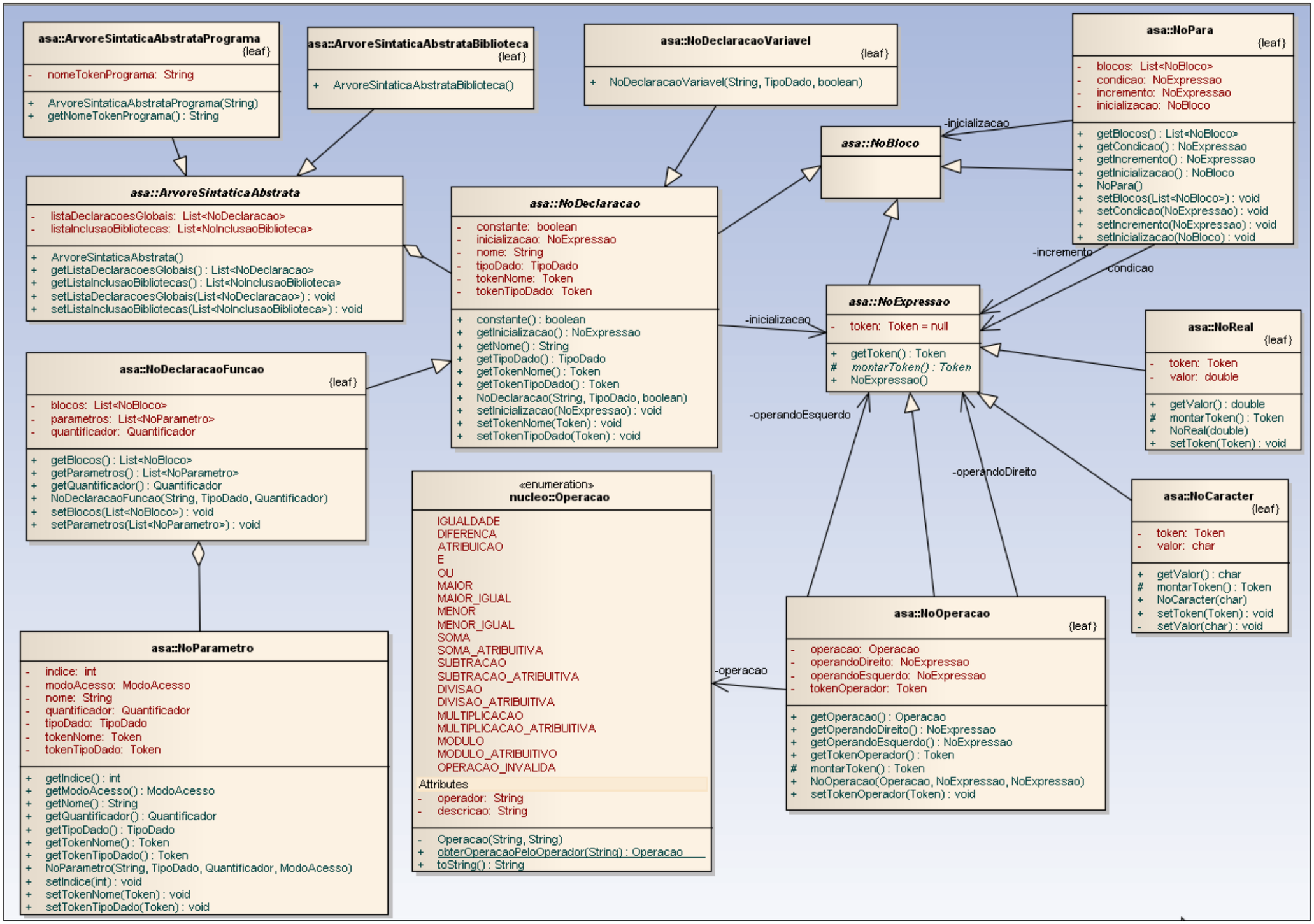

Figura 3- Diagrama de classes da AST do Portugol.

Ao receber o código fonte que deverá ser interpretado como no exemplo da Tabela 3, o interpretador então realiza uma análise sintática deste código fonte e em seguida constrói uma AST que representa o código fonte, instanciando objetos relacionados a cada tipo de construção do código fonte como demonstrado pela Figura 4. Após a construção da AST, ele então realiza a análise semântica navegando na estrutura representada pela AST, e ao final o interpretador Portugol percorre a AST e executa o programa. O Interpretador Portugol devolve ao Portugol Studio o resultado do programa ou a lista de erros encontrados durante as análises sintática e semântica.

Tabela 3 - Exemplo de um programa escrito em Portugol.

```
programa
{
	funcao inicio()
	{
		inteiro soma = 0
		inteiro n = 4

		para ( inteiro k = 0; k <= n; k++ )
		{
			soma = soma + k
		}
		escreva( "A soma vale: ", soma )
	}
}
```

A representação do código fonte demonstrado na Tabela 3 em objetos que compõem a AST no interpretador Portugol pode ser observado na Figura 4 onde neste caso, há uma instancia do objeto ArvoreSintaticaAbstrataPrograma, este objeto então agrega uma declaração de função NoDeclaracaoFuncao que possui uma lista de objetos do tipo NoBloco, o primeiro item da lista é uma declaração de variável representado por uma instância de NoDeclaracaoVariavel, essa instancia possui o estado soma no atributo nome e possui como inicialização um objeto do tipo NoInteiro, o mesmo acontece na segunda linha de código, mudando apenas os estados do objeto. O terceiro item do bloco é um objeto NoPara uma referência a nós da árvore que compõem seus estados inicialização, condição e incremento além de agregar NoBlocos pertencentes a seu escopo, por fim, o ultimo objeto na lista de NoBloco da função inicio é um objeto NoChamadaFuncao.

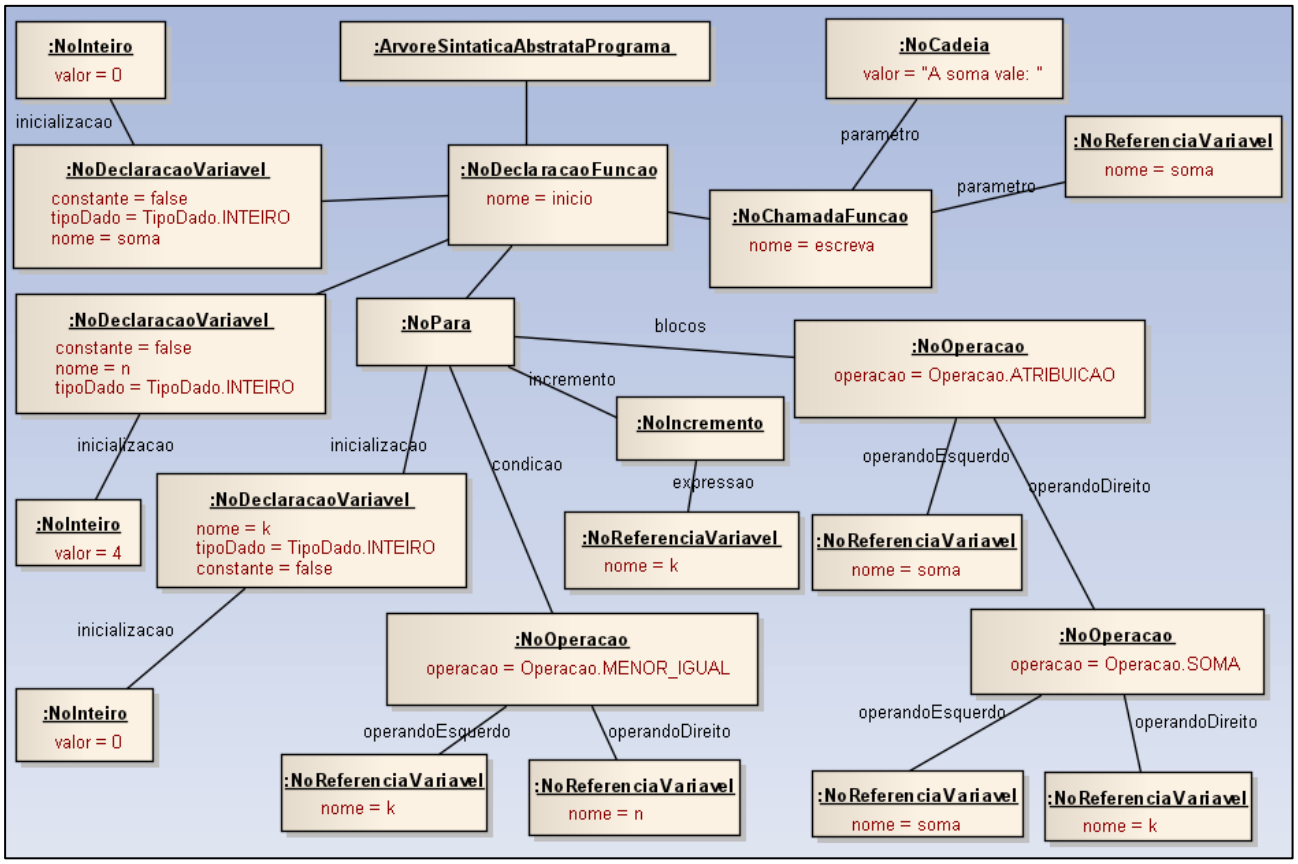

Figura 4 - Exemplo de uma instância de AST que representa um programa escrito em Portugol.

Realizaram-se testes na última versão e verificou-se que o interpretador Portugol apresenta alguns problemas relacionados à manipulação de matrizes e vetores, porém a maioria das estruturas que ele propõe-se a suportar está funcionando corretamente. Observou-se também que o Portugol Studio está sem destaque de sintaxe no editor de código fonte, além de não possuir um sistema de ajuda.

Na Figura 5 é demonstrada a atual interface gráfica do Portugol Studio. Na porção maior da tela (Centro) está localizado o editor de código fonte, que possui navegação em abas entre os arquivos abertos, na parte superior, localizam-se os controles de menu e botões, e na parte inferior separado por aba estão o console de execução e o componente que lista os erros devolvidos pelo interpretador.

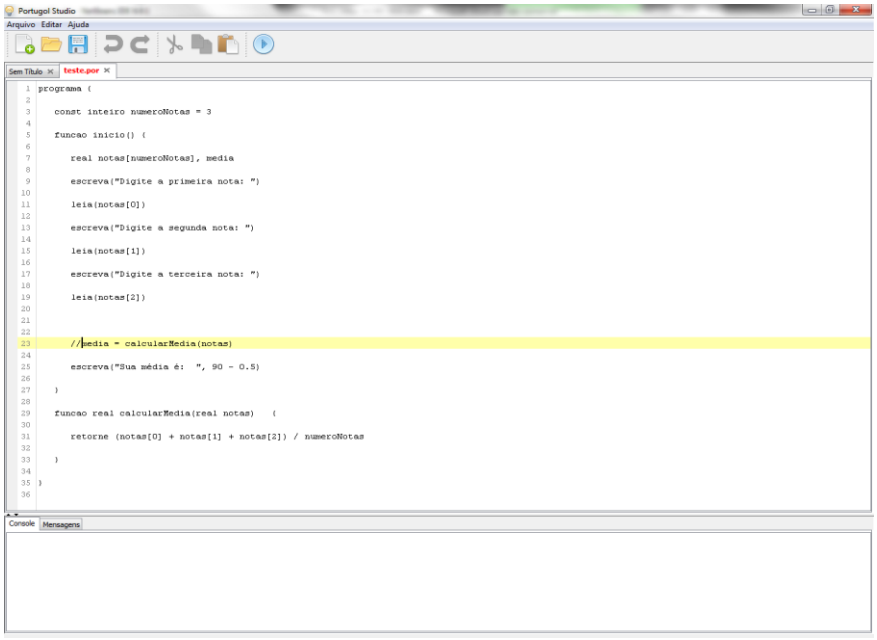

Figura 5 – Visão geral do PortugolStudio versão 0.1.

2.4 Considerações

Há diversos esforços em construir e evoluir os corretores automáticos de algoritmos, cada qual focado em uma determinada linguagem de programação e focado em seu ambiente de ensino, como demonstrado por Douce et. al (2005). Na Tabela 4 é possível observar alguns corretores e as principais características de correção exercidas por eles.

Tabela 4 - Tipos de correção utilizados pelos corretores, adaptado de Naudé (2007).

	Hollingsworth	ASSYST	BOSS	Kassandra	TRY	**PortugolStudio**	CourseMaster	ELP	HoGG	PASS	Scheme-robo
Eficiência		X									X
Funcionamento	X	X	X	X	X	X	X		X	X	X
Cobertura por teste		X									
Características estruturais						X	X	X			X
Estilo de programação		X									
Complexidade		X						X			
Métricas de software								X			

Visto o cenário atual dos corretores automáticos, o presente trabalho utilizará da correção do funcionamento e características estruturais através das abordagens estática e dinâmica, para pontuar e submeter um feedback nos exercícios dos alunos.

Este trabalho se alinha com os esforços do grupo de pesquisa de informática na educação da UNIVALI ao auxiliar alunos nos exercícios extraclasse podendo futuramente ser integrado ao ambiente ALICE para melhorar o tempo de entrega de novos exercícios do tutor inteligente.

Os aspectos que devem ser implementados no Portugol são: 1) Finalizar suporte a vetores e matrizes no interpretador, 2) Incluir um sistema de Ajuda, 3) Adicionar no editor um sistema de destaque de sintaxe, e 4) Acrescentar um espaço para leitura de exercícios e um botão para submissão da solução para o corretor.

Foi necessário criar um espaço no PortugolStudio para a exibição do exercício que o aluno deve realizar, esse exercício será lido de um arquivo XML (*Extensible Markup Language*), contendo o enunciado, e este enunciado será exibido ao aluno para que ele consiga soluciona-lo.

Com a finalização do PortugolStudio, os alunos terão acesso a uma linguagem mais similar as utilizadas no decorrer do curso, linguagens essas derivadas do C, como a linguagem Java. Aumentando assim a gama de ferramentas para estudos de introdução a programação que atualmente conta com o Escracho, Bipide e Webportugol.

3. ESTUDO PILOTO

Nesta sessão são apresentados os detalhes do modelo proposto para correção, o resultado do teste piloto realizado com os avaliadores para validar a proposta de avaliação, e o detalhamento do Portugol Studio. Por fim é apresentada a proposta de implementação do corretor de algoritmos.

3.1 Catalogação de um conjunto de questões

Um conjunto de exercícios para resolução de algoritmos focados em programação introdutória se faz necessário para criação de um benchmark onde um grupo de alunos resolve os exercícios e um grupo de professores corrige esse conjunto de questões resolvidas. Essa correção e as respostas dos alunos servirão para auxiliar a escolha das técnicas de correção automática e também servirão para comparar o resultado devolvido pelo corretor automático e a correção feita pelos professores.

Para criar esse benchmark as questões devem abordar a ementa da disciplina de Algoritmos I, por isso procurou-se por exercícios que já tivessem publicações como sendo recomendados ou frequentemente utilizados para ensino de programação introdutória.

Essa procura de um conjunto de exercícios na literatura foi realizada utilizando um protocolo de busca e a pesquisa foi realizada nos repositórios da ACM Digital Library e IEEExPlorer, além do buscador de artigos e publicações Google Scholar.

Utilizou-se das seguintes palavras chave:

- Algorithm exercises CS1;
- Algorithm exercises introductory programming;
- Exercises algorithms;
- Exam student introductory programming; e
- Test student introductory programming.

Não foram encontrados conjuntos de questões ou provas de algoritmos classificados e catalogados como essenciais nas disciplinas de Algoritmos. Encontraram-se provas e materiais

feitos por professores que ministram esse tipo de disciplina, mas nenhum teste demonstrando que aquelas questões supriam por completo o conteúdo de programação introdutória e Algoritmos I. Por esse motivo, tomou-se a liberdade de elaborar em conjunto com o Prof. André Raabe, uma lista de questões abrangendo os principais conceitos destacados por Tew e Guzdial (2010) como fundamentais para disciplina de programação introdutória. Os conceitos são os que seguem.

- Fundamentação (Variáveis, atribuição, etc.);
- Operadores lógicos;
- Desvios condicionais (if / else);
- Laços iterativos (for);
- Laços condicionais (while);
- Vetores;
- Funções (parâmetros de métodos);
- Funções (valores de retorno);
- Recursão; e
- Conceitos básicos de orientação a objeto (Definição de classes, chamada de métodos).

Tew e Guzdial (2010) não publicaram a lista de exercícios que eles elaboraram em seu estudo de validar um teste que examina por completo estudantes da disciplina de Algoritmos e Programação. A lista de questões encontra-se no Apêndice A e englobam os seis primeiros itens da lista de conceitos evidenciados por Tew e Guzdial (2010) e consiste em 60 exercícios para escrita de código fonte que foram separados em quatro categorias com base na ementa atual da disciplina de algoritmos, sendo elas:

1. Algoritmos Sequenciais;
2. Desvios Condicionais;
3. Laços de Repetição; e

4. Vetores e Matrizes.

3.2 Definição da métrica

Segundo Ala-Mutka e Järvinen (2004) não há critérios comuns de medição para tarefas básicas de programação na literatura. Normalmente, o funcionamento correto, um bom projeto, um bom estilo de programação são exigidos. No entanto as definições e os pesos relativos dessas características na avaliação variam muito. Diferentes professores enfatizam diferentes características, baseando os seus critérios, sobre sua experiência pessoal. Essa diferença de avaliação entre professores sugere que os critérios de avaliação devem ser claramente especificados.

Portanto para assemelhar as correções entre avaliadores sobre a correção piloto, foram adaptados os critérios utilizados pelo SistLog – Sistema de Auxílio à Avaliação de Algoritmos (MIRANDA, 2000). Os critérios são os que se seguem:

- Identificação correta da sequência das instruções;
- Uso correto das estruturas de controle;
- Uso correto das condições nos desvios e laços;
- Geração correta das saídas de dados;
- Identificação correta das entradas de dados;
- Uso correto da sintaxe;
- Variáveis foram inicializadas quando necessário; e
- Variáveis foram declaradas e possuem tipo correto.

Pediu-se aos avaliadores para informarem o grau de importância que cada critério possui, de acordo com sua percepção, na correção de um exercício de algoritmos. Os pesos dos critérios definidos pelos avaliadores e a média estão demostrados na Figura 6. Observou-se o desvio padrão dos valores informados pelos avaliadores como demonstrado pela

Tabela 5.

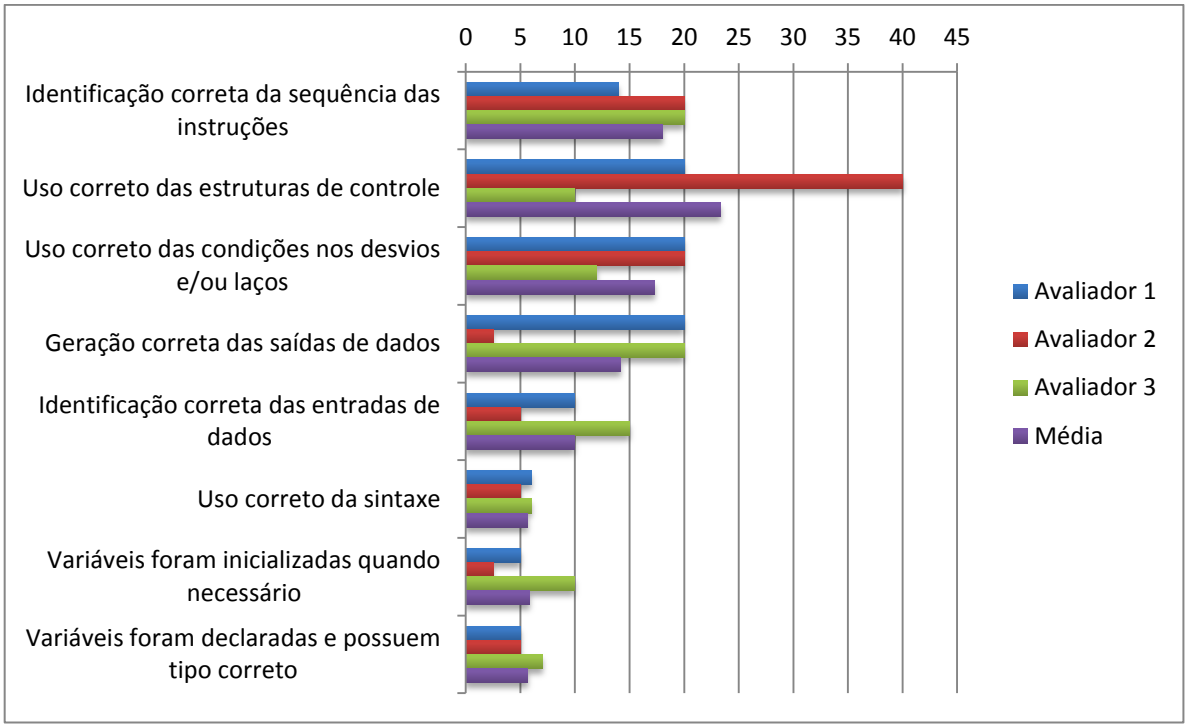

Figura 6 - Gráfico de importância dos critérios definido pelos professores avaliadores.

Tabela 5 - Desvio padrão dos pesos dos critérios definidos pelos avaliadores.

Pesos por Avaliador			Média	Desvio padrão dos pesos
Avaliador 1	**Avaliador 2**	**Avaliador 3**		
14	20	20	18,00	**3,46%**
20	40	10	23,33	**15,28%**
20	20	12	17,33	**4,62%**
20	2,5	20	14,17	**10,10%**
10	5	15	10,00	**5,00%**
6	5	6	5,67	**0,58%**
5	2,5	10	5,83	**3,82%**
5	5	7	5,67	**1,15%**

No gráfico da Figura 6 é possível observar que os valores dos critérios entregues pelos avaliadores são similares, e que a maior parte da relevância ficou nos três primeiros critérios, além de o Avaliador 2 focar sua pontuação no critério “Uso correto das estruturas de controle” .

Realizou-se um teste piloto utilizando-se, para cada avaliador, um mesmo conjunto de provas, provas estas que foram realizadas pela turma de Algoritmos I (Um) ministrado pelo prof. André Raabe no primeiro semestre de 2011. Para a correção utilizou-se um gabarito, seguindo os critérios definidos anteriormente nesta sessão, como o demonstrado na Tabela 6, onde pediu-se para o avaliador informar a porcentagem que o aluno contemplou em cada critério de cada questão.

Tabela 6 - Gabarito de correção.

	Notas					
Critérios	**Q1**	**Q2**	**Q3**	**Q4**	**Q5**	**Q6**
Identificação correta da sequência das instruções						
Uso correto das estruturas de controle						
Uso correto das condições nos desvios e laços						
Geração correta das saídas de dados						
Identificação correta das entradas de dados						
Uso correto da sintaxe						
Variáveis foram inicializadas quando necessário						
Variáveis foram declaradas e possuem tipo correto						

Obteve-se a nota de cada questão através da somatória dos produtos do peso do critério com a porcentagem de acerto do critério, obtendo-se a porcentagem de acerto da questão como um todo. Por fim multiplicou-se a porcentagem total de acerto com o peso da questão na prova. Recuperou-se a nota final do aluno através da somatória das notas individuais das questões. Essa operação está representada pela Equação 1 onde NP é nota da prova, C é critério e Q é questão.

$$NP = \sum_{j=1}^{m} \left(\frac{\sum_{i=1}^{n} Ci * pesoCi}{100} \right) * pesoQj \qquad \text{Equação 1}$$

Removeu-se a questão quatro (Q4) da amostra de notas, pois esta questão é de múltipla escolha e foge do escopo do corretor automático. Distribuiu-se então o peso da questão quatro proporcionalmente entre as demais questões para a prova continuar valendo 10. A comparação entre todas as notas obtidas pelo sistema de avaliação proposto pode ser observada na Figura 7.

Observou-se que, apesar da definição dos critérios, houve diferença de interpretação no critério “Geração correta das saídas de dados”, percebeu-se que os avaliadores Avaliador 1 e Avaliador 3, descontaram pontos, quando o algoritmo não imprimia corretamente, portanto quando era identificado algum erro de estrutura que ocasionava a impressão incorreta dos dados, esse critério também era descontado, enquanto que o Avaliador 2 verificou apenas se o aluno tomou o cuidado de exibir algo relevante em relação ao solicitado pela questão da prova.

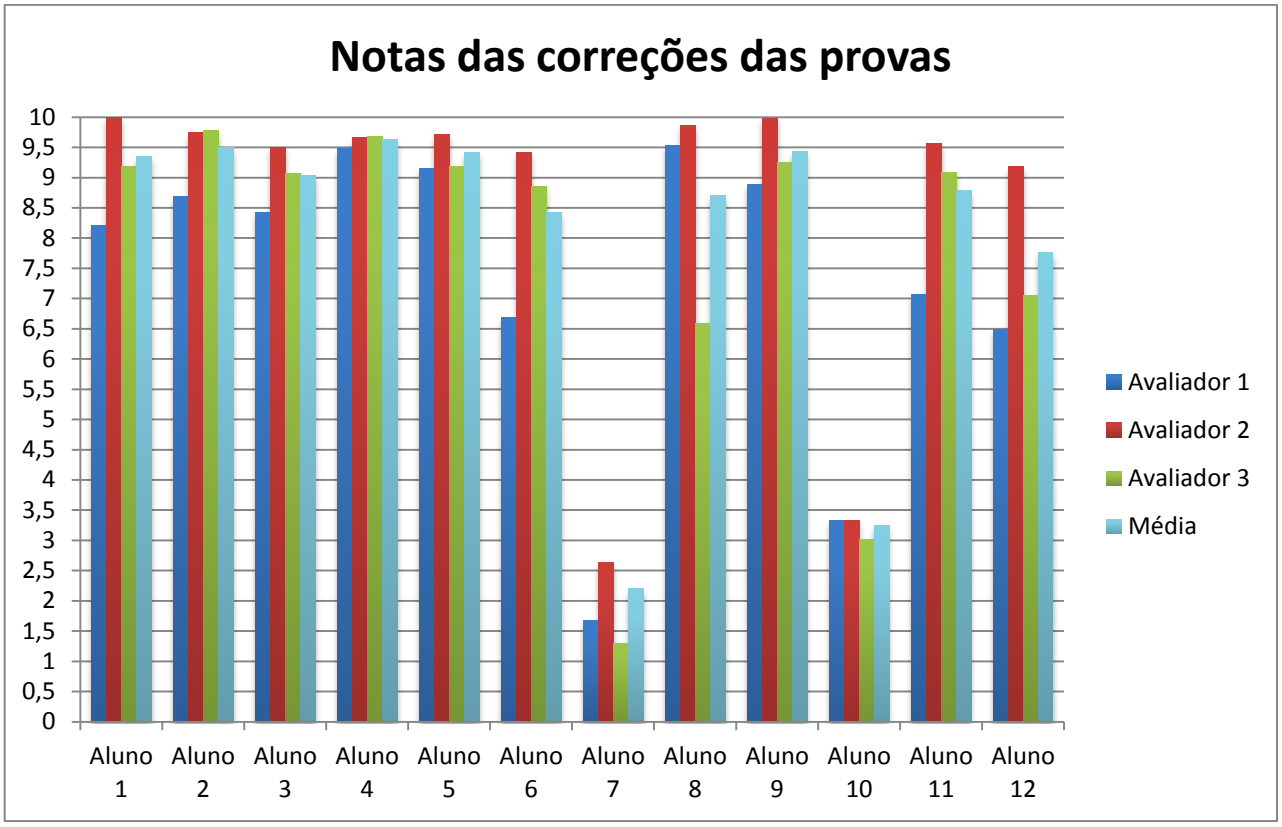

Figura 7 - Gráfico de comparação das notas entregues pelos avaliadores e correção fictícia.

Constatou-se também que todas as avaliações realizadas pelo Avaliador 3, continham o critério "Variáveis foram inicializadas quando necessário" em branco, com a anotação de que este critério não se aplicava na correção da prova.

Contudo o teste piloto serviu para avaliar se a utilização de critérios e pesos para correção de algoritmos é valida, e também auxiliou a identificar problemáticas nas definições dos critérios. Para resolução dessa problemática na interpretação dos critérios durante a execução do teste final o formulário recebeu diretivas para auxiliar a interpretação dos critérios.

4. DESENVOLVIMENTO

Nesta sessão são apresentadas as ferramentas utilizadas, as modificações realizadas no Portugol, os detalhes do desenvolvimento do corretor automático, e por fim uma síntese do teste final e comparação entre o corretor automático e a avaliação feita por professores.

4.1 Ferramentas utilizadas

Durante o desenvolvimento dos softwares envolvidos nessa pesquisa utilizaram-se algumas ferramentas, dentre essas, destaca-se o sistema de controle de código fonte (*source code control system*). Segundo HUNT e THOMAS (1999) um SCCS permite navegar entre todas as alterações feitas no código fonte, saber quando foram realizadas, por quem foram realizadas, este também permite identificar releases do software e facilita a edição concorrente de um mesmo arquivo.

O sistema de controle de código fonte utilizado nesse TCC foi o git[2] que dentre as vantagens listam-se: Desenvolvimento Distribuído, Desenvolvimento não linear e utilização por projetos de grande porte. Este foi utilizado em conjunto do github[3]. O github trata-se de uma ferramenta online de colaboração que hospeda repositórios git e permite gerenciar *issues* e *releases* e facilita o trabalho e organização de equipes distribuídas. Com a publicação dos softwares presentes nesse TCC no github estes passaram a ser de código aberto, e estão disponíveis nos seguintes endereços: http://github.com/fpelz/PortugolCorretor, http://github.com/fpelz/PortugolStudio e http://github.com/fpelz/Portugol-Nucleo.

4.2 Alterações do Portugol

Durante o desenvolvimento do TCC necessitou-se realizar alterações no PortugolStudio e Portugol-Núcleo visto que ambos não estavam maduros o suficiente para entrar em produção. O Portugol-Núcleo consumia a thread de EventDispatcher do Java Swing ocasionando travamentos na interface do PortugolStudio, a interface de entrada e saída foi modificada passando a dizer qual tipo de dado era a saída e qual o tipo de dado deveria ser a entrada. Foram identificados problemas

[2] Disponível em http://git-scm.com
[3] Disponível em http://github.com

críticos no analisador sintático fazendo com que este fosse reimplementado e o analisador semântico também sofreu modificações.

O PortugolStudio não estava integrado ao Portugol-Núcleo e o código base do PortugolStudio estava muito instável com problemas arquiteturais como: duplicação de código implícita, difícil extensibilidade devido à má definição de interfaces e acoplamentos e composições de objetos má definida. Tudo isso fez com que boa parte do código fosse reescrita enquanto a Turma de Algoritmos 2011/2 utilizava um software que quebrava constantemente. Os principais sintomas relatados pelos alunos foram: travamentos, botões que não mudavam de estado, usabilidade prejudicada pela forma como o PortugolStudio manipulava arquivos. Isso fez com que vários releases fossem lançados até os alunos receberem um software estável para utilização.

A modificação dos dois software foram feitas em paralelo, tanto o PortugolStudio quanto o Portugol-Núcleo, pois as varias mudanças na interface de comunicação do Portugol-Núcleo quebravam a integração entre os dois aplicativos.

As mudanças realizadas no Portugol-Núcleo são descritas por NOSCHANG (2011), destacam-se aqui aquelas que estão relacionadas com este TCC e que receberam influência na tomada de decisão durante o uso do PortugolStudio pelos alunos.

A primeira versão a entrar em produção do PortugolStudio a 0.1 a integração com o núcleo era feita com a instanciação dos analisadores sintático, semântico e do interpretador no método de actionPerformed() do botão play do PortugolStudio. Essa maneira de codificar facilitava o erro ao realizar chamadas incorretas ou fora de ordem ao Núcleo e com o interpretador executando dentro do actionPerformed ocasionavam travamentos no EventDispacther quando o algoritmo Portugol possuía um laço infinito por exemplo.

Para solucionar esse problema, a responsabilidade de analisar o código, transformá-lo em arvore sintática abstrata e em verificar se a arvore está apta a ir para o interpretador foi passada para o núcleo com a criação dos *Adapters* Portugol e Programa. A classe Portugol possuí o método compilar recebendo a String de código fonte e retorna uma instancia de Programa podendo gerar uma exceção de ErroDeCompilação caso ocorra algum problema nas análises sintática e semântica. Com uma instancia de Programa é possível recuperar a arvore sintática abstrata, executar o programa e interrompe-lo quando necessário, abstraindo assim as manipulações que eram realizadas no interpretador. A Figura 8 demonstra o diagrama de classes das interfaces expostas pelo núcleo

para realizar a integração. Na tabela 7 está demonstrado as chamadas ao núcleo para executar o código do aluno dentro do PortugolStudio.

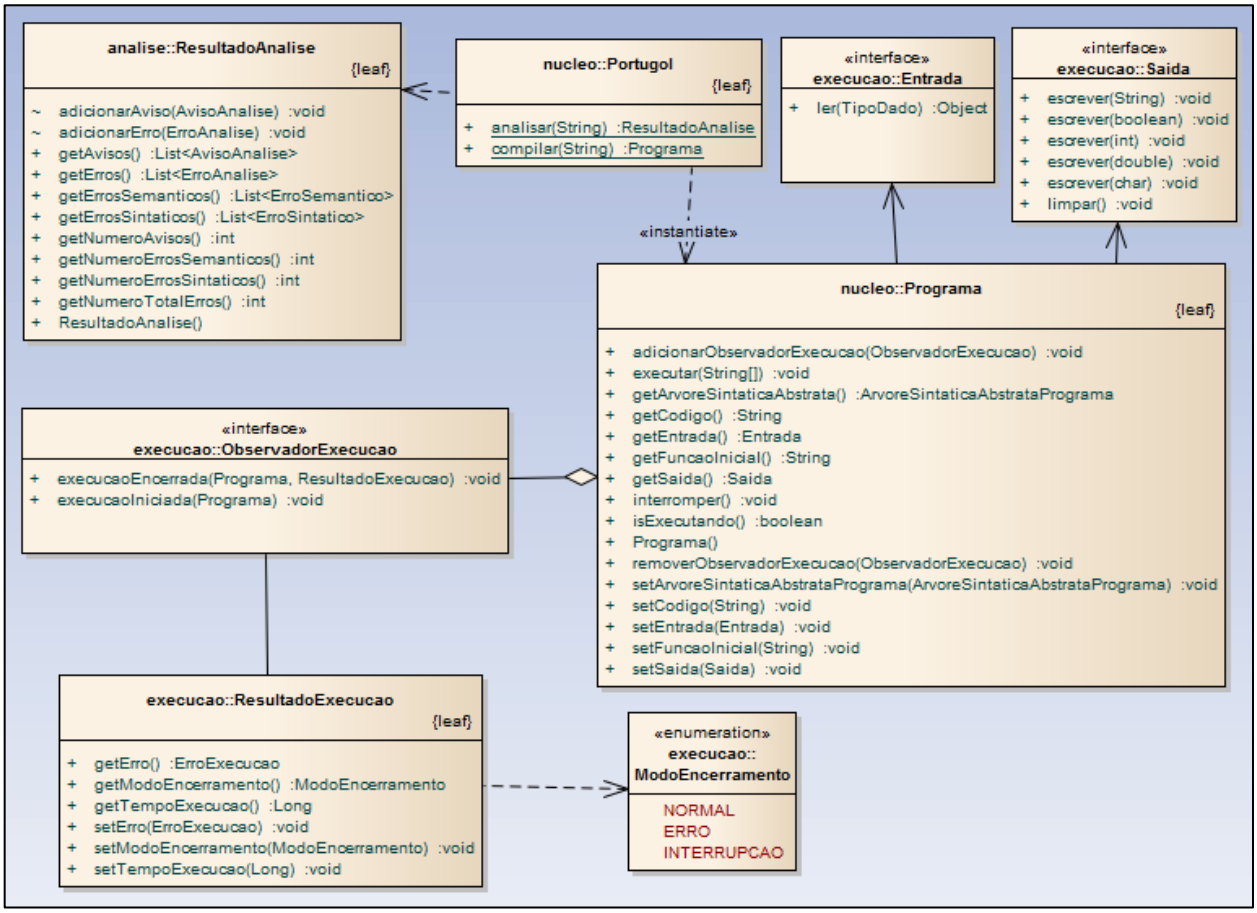

Figura 8- Classes para comunicação com o Portugol-Núcleo.

Tabela 7 - Código fonte de integração do núcleo com o PortugolStudio

```
private void executar() {
   AbaMensagemCompiladorabaMensage m = painelSaida.getMensagemCompilador();
   abaMensagem.limpar();
   try {
      String codigo = editor.getPortugolDocumento().getCodigoFonte();
      if (programa == null)
         this.programa = Portugol.compilar(codigo);
      programa.setEntrada(painelSaida.getConsole());
      programa.setSaida(painelSaida.getConsole());
      programa.adicionarObservadorExecucao(this);
      programa.executar(getParametros());
   } catch (ErroCompilacao erroCompilacao) {
      ResultadoAnalise resultadoAnalise = erroCompilacao.getResultadoAnalise();
      if (resultadoAnalise.getNumeroTotalErros() > 0) {
         abaMensagem.atualizar(resultadoAnalise);
         abaMensagem.selecionar();
```

```
        }
    }
}
```

Outra modificação realizada no Portugol-Núcleo foi a mudança na interface da AST para a aceitação de um *visitor* (GAMMA et al, 1995), o que permitiu posteriormente a implementação do Corretor, essa mudança acarretou na inclusão do método “aceitar” recebendo por parâmetro um objeto do tipo VisitanteASA em todos os objetos da hierarquia da AST. No corpo desse método é realizada a chamada ao método visitar do visitante passando como parâmetro sua própria referência como demonstrado na Tabela 8. A interface VisitanteASA possuí uma sobrecarga do método aceitar para cada objeto visitável contido na arvore sintática abstrata.

Tabela 8 - Código incluído para implementação do visitor.

```
@Override
public void aceitar (VisitanteASA visitante) throws Exception
{
          visitante.visitar(this);
}
```

O analisador sintático utilizava-se de um mecanismo do ANTLR onde este não parava a análise mesmo não conseguindo construir a AST corretamente, o que ocasionava em funcionamentos errôneos, quebras e travamentos do interpretador ao receber uma arvore incompleta, esse problema foi solucionado alterando a maneira como o parser trocava mensagens com o analisador sintático e com a adição de um listener de erros de parse.

A integração do Portugol-Núcleo com o PortugolStudio realizou-se através da implementação das interfaces de entrada e saída, estas interfaces foram vinculadas a AbaConsole e para garantir o correto funcionamento do Swing durante a chamada da interface entrada utilizou-se de um SwingWorker com o intuito de deixar a thread do núcleo esperando em background a entrada do usuário para não travar a GUI. O mesmo acontece com a interface saída, porem a utilização do SwingWorker neste caso foi para sincronizar a impressão dos valores no JTextArea.

O PortugolStudio foi todo dividido em abas através da herança da classe abstrata Aba, uma aba é um objeto que consegue adicionar-se em um JTabbedPane e possuí o controle sobre seu

cabeçalho, isso foi encapsulado para evitar código duplicado e todas as abas possuírem comportamentos similares.

Para agregar as abas foi criado o objeto PainelTabulado, sendo este um subtipo de JTabbedPane com uma interface que facilita a manipulação de objetos do tipo Aba que são agregados a ele. O PainelTabulado dispara um evento de aba selecionada passando por parâmetro a Aba que gerou o evento. O principal ouvinte do PainelTabulado é a tela principal, pois ela necessita saber qual aba foi selecionada para configurar, habilitar ou desabilitar as opções da barra de menus.

Importante destacar que a aba que possui as chamadas ao núcleo é o objeto AbaCodigoFonte esta se registra como ouvinte de execução, pois ela possui um instancia de Programa, ela possui uma referência ao PainelSaida, este que é composto por uma AbaConsole e uma AbaMensagemCompilador, além de possuir um Editor, no qual é o painel que fica o código digitado pelo aluno. Este objeto AbaCodigoFonte funciona como um wrapper entre o Portugol-Núcleo, o console, a tabela de mensagens e o editor, por esse motivo ele se registra como ouvinte e faz as requisições entre esses objetos, como por exemplo, quando uma AbaMensagemCompilador dispara um evento posicionarCursor(linha,coluna), a AbaCodigoFonte recebe e faz uma chamada ao editor. Na Figura 9 é possível verificar o diagrama de classes que demonstra as relações descritas anteriormente.

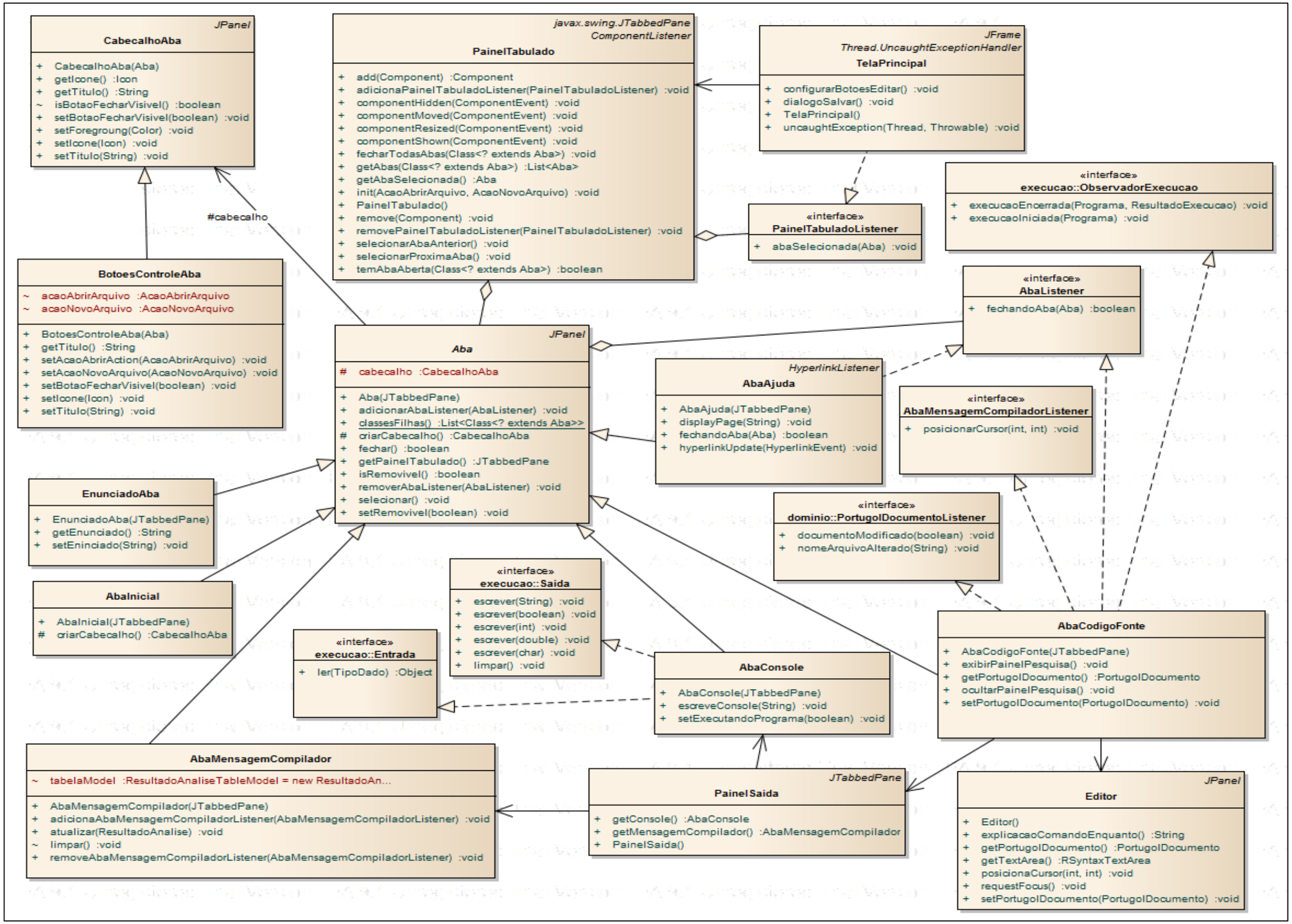

Figura 9 - Diagrama de classes do PortugolStudio

Outra modificação realizada no PortugolStudio foi na interface de usuário, com o intuito de melhor hierarquizar a distribuição dos componentes na tela, MATHIS (2011) demonstra que usuários estão habituados a entender hierarquias da esquerda para direita, do topo para baixo e de fora para dentro. Na nova interface os botões que influenciam mudanças na aba selecionada foram colocados dentro da aba em verde na Figura 10 e os botões que modificam a janela ficaram fora da aba em vermelho na Figura 10. Isso faz com que a distribuição dos botões fique mais próxima de onde a ação ocorrerá, na versão 0.1, por exemplo, ao clicar no botão salvar não da para identificar se tudo será salvo ou apenas a aba selecionada.

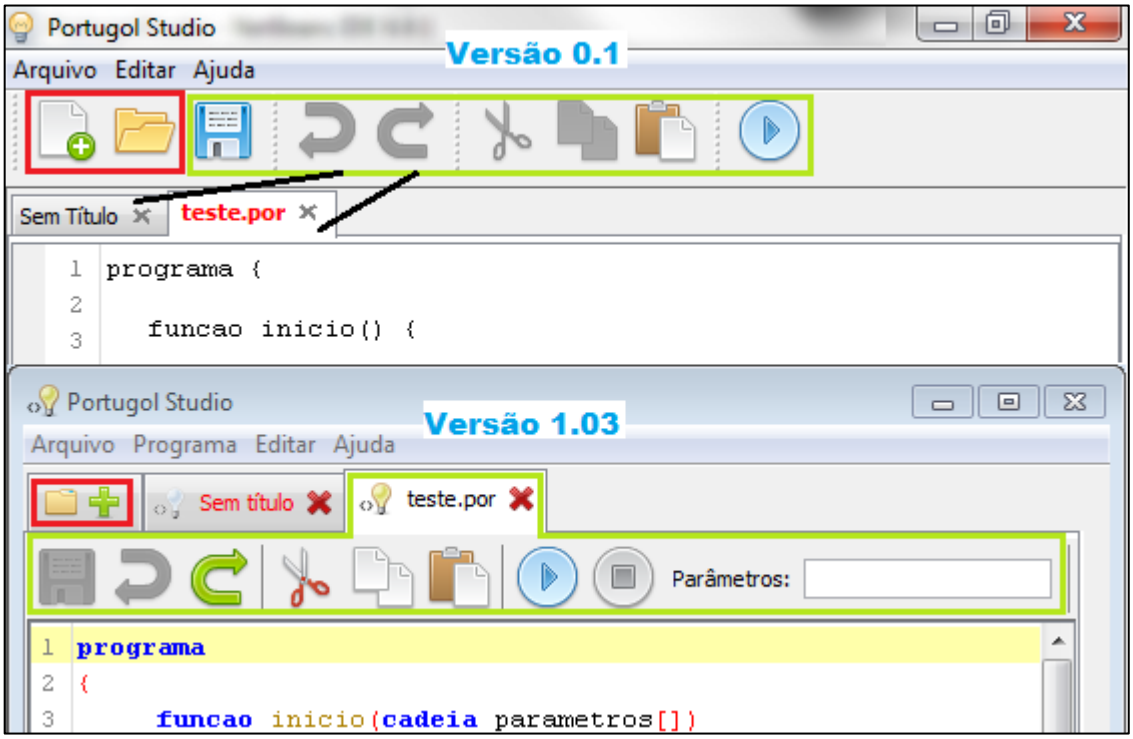

Figura 10 - Comparação das mudanças feitas na interface do usuário do Portugol Studio.

Para a integração do PortugolStudio com o corretor de algoritmos a interface de usuário foi alterado para mostrar ao usuário o resultado da correção e o enunciado, para isso foi adicionado à esquerda do editor um painel contendo a nota obtida depois dos testes, dicas verificadas na correção estrutural, além de demonstrar os casos que falharam listando as entradas e saídas esperadas e a saída recebida. Também foi necessário incluir uma nova aba que contem o enunciado da questão que será corrigida, estas modificações podem ser observadas na Figura 11.

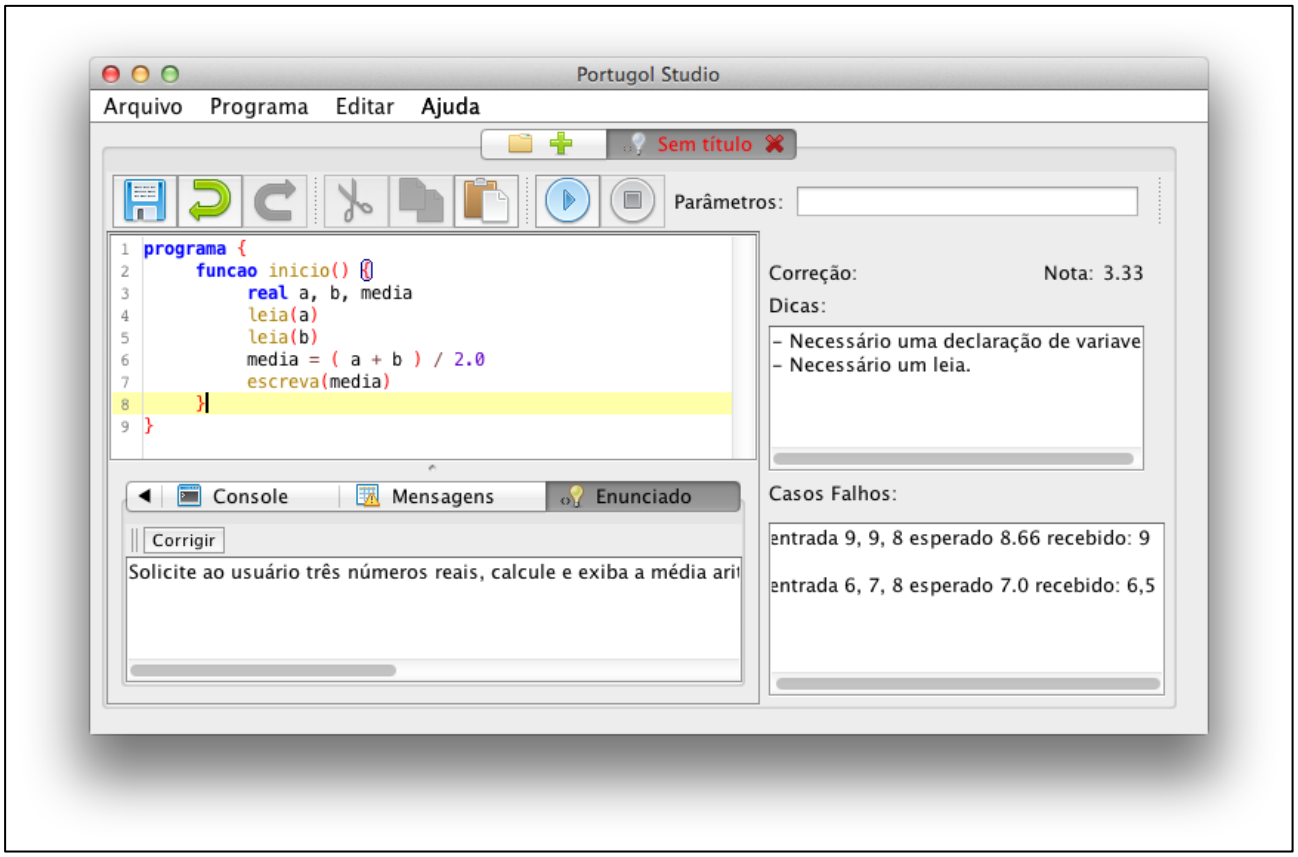

Figura 11 - PortugolStudio com corretor.

4.3 Corretor

A construção do corretor automático utilizou-se de um esquema similar ao proposto por Jesus (2010), onde se realiza comparação das estruturas e comparações das saídas do programa. Para isso foi implementado um visitante para a arvore sintática abstrata do Portugol-Núcleo que serializa os nós da AST em um XML que representa apenas a estrutura geral do algoritmo, outra codificação necessária foi a implementação das interfaces de entrada e saída do interpretador do Portugol-Núcleo para realizar os testes dinâmicos.

4.3.1 Corretor Estrutural

A etapa de correção das estruturas do algoritmo foi realizada comparando a serialização AST correspondente ao programa feito pelo aluno com uma serialização da AST que representa o gabarito do exercício em XML. Para a comparação foi utilizada uma biblioteca Java *open source*

denominada diffX[4], esta compara documentos XML através da análise da sequencia de eventos XML. A biblioteca disponibiliza um listener de eventos de comparação, sendo possível verificar no XML quais nós necessitam ser incluídos ou removidos para que os dois documentos XML fiquem iguais. O principal foco da análise estática está em demonstrar ao aluno qual parte de seu algoritmo deve ser reescrito para cumprir com o exercício.

Na Tabela 9 é possível verificar um exemplo de XML que resultou de uma serialização de AST utilizando o *visitor* implementado no corretor. O XML em questão representa a estrutura do algoritmo que se encontra na tag de solução do XML de questão da Tabela 10.

Tabela 9 - Estrutura serializada.

```
<programa>
  <funcao nome="inicio">
    <declaracao tipo="variavel"/>
    <declaracao tipo="variavel"/>
    <declaracao tipo="variavel"/>
    <declaracao tipo="variavel"/>
    <leia/>
    <leia/>
    <leia/>
    <escreva/>
  </funcao>
</programa>
```

A biblioteca disponibiliza um listener que dispara eventos ao realizar a comparação dos documentos XML e o corretor mapeia esses eventos de inserção ou remoção, verificando o pai da tag e então guarda esses eventos para posteriormente serem mostrados ao aluno como dicas para completar o exercício.

Como no XML de gabarito pode haver várias soluções exemplos, a que o corretor considera como apropriada para o aluno, é a que recebeu menos eventos de modificação em comparação com a serialização feita do algoritmo do aluno.

[4] http://www.topologi.com/diffx/

4.3.2 Corretor Dinâmico

O desenvolvimento do corretor dinâmico aconteceu através da implementação das interface de entrada e saída do Portugol-Núcleo, porém, ao contrário do PortugolStudio, no corretor as entradas de dados vinham através de uma XML e a saída entrega pelo núcleo é comparada com um valor correspondente ao que esta no XML de questão.

O XML que representa um exercício está demonstrado na Tabela 10 e este é transformado em objetos em memória utilizando a biblioteca Java open source XStream. Com isso as tags do XML são instanciadas no modelo de dados demonstrada na Figura 12, onde o objeto do tipo FabricaQuestao é responsável por realizar as chamadas à biblioteca XStream. Os dois objetos Converter são utilizados pela biblioteca para conseguir manipular corretamente a presença das tags entrada e saída no documento XML.

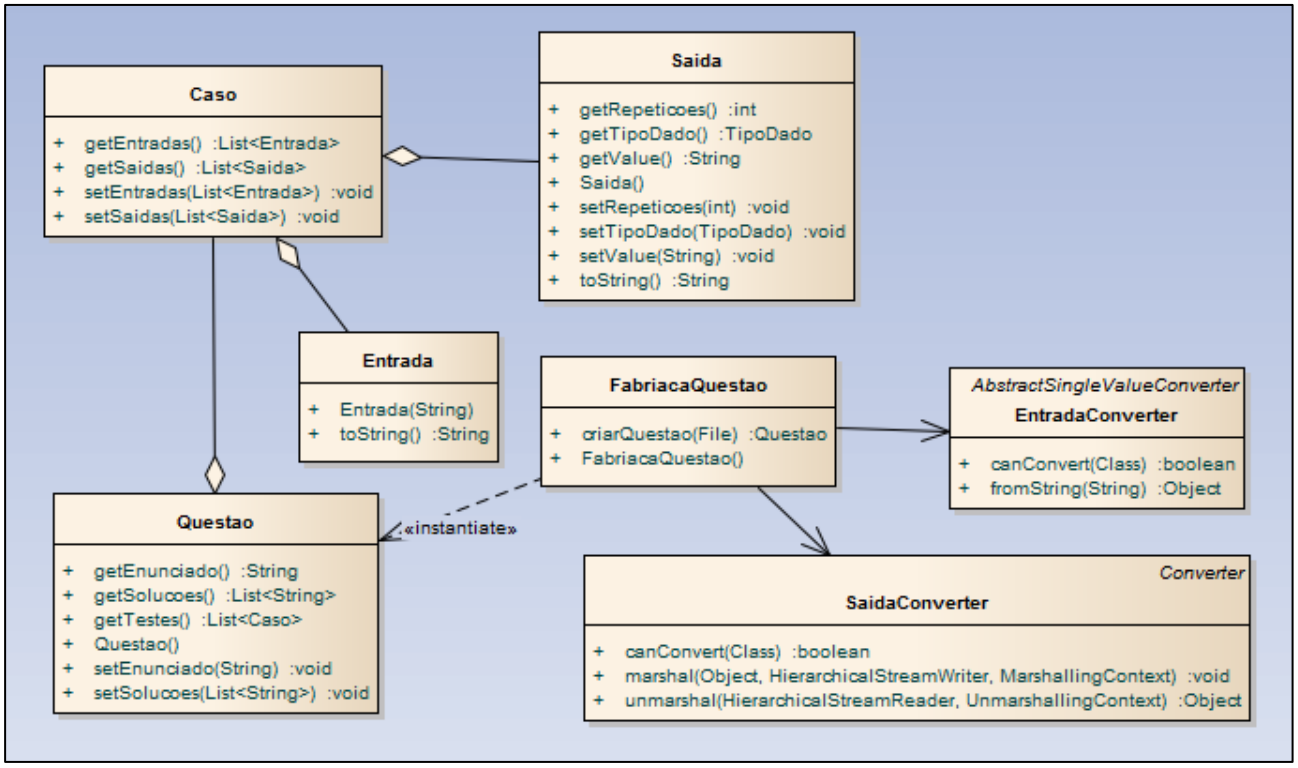

Figura 12 - Modelo de dados do corretor.

Com a implementação da interface Saída do núcleo, um método para cada tipo de dado foi codificado, isso facilitou a identificar se o dado de saída do programa é realmente o dado a ser comparado com o que está no gabarito XML, por exemplo, em um programa que espera como saída

um valor inteiro e o aluno em seu algoritmo utilizar chamadas ao escreva com textos informativos, estas saídas não serão comparadas, pois não são do tipo inteiro e sim do tipo cadeia.

Como o núcleo não expõe a tabela de símbolos do interpretador, a única maneira de intervir nos valores das variáveis e com a interface Entrada, isso faz com que o corretor se comporte como um usuário do sistema, sem poder manipular o funcionamento do interpretador enquanto este trabalha sobre o algoritmo do aluno.

Tabela 10 - Exemplo de XML de Questão.

```
<questao id="1">
    <enunciado>Solicite ao usuário três números reais, calcule e exiba a média
aritmética dos valores.</enunciado>
  <solucoes>
      <solucao>
        programa {
          funcao inicio() {
            real a, b, c, media
            leia(a)
            leia(b)
            leia(c)
            media = (a + b + c) / 3.0
            escreva(media)
          }
        }
      </solucao>
    </solucoes>
  <testes>
    <caso>
      <entradas>
        <entrada>6</entrada>
        <entrada>7</entrada>
        <entrada>8</entrada>
      </entradas>
      <saidas>
        <saida tipo-dado="real" repeticoes="1">7.0</saida>
      </saidas>
    </caso>
      <caso>
          <entradas>
              <entrada>9</entrada>
              <entrada>9</entrada>
              <entrada>8</entrada>
          </entradas>
          <saidas>
              <saída tipo-dado="real" repeticoes="1">8.6</saida>
          </saidas>
      </caso>
  </testes>
</questao>
```

A nota é calculada conforme a quantidade de casos que passam pelos testes, isso faz com que a nota esteja muito vinculada a escrita do XML de gabarito, caso este possua poucos casos de teste a nota é pouco fragmentada, além de possuir o risco de não cobrir todos os pontos onde o algoritmo do aluno pode falhar.

4.4 Resultados

Durante todo o semestre a turma de algoritmos e programação 2011/2 utilizou o PortugolStudio para praticar e realizar as atividades de aula, por conta de o software estar incompleto, várias modificações e versões foram lançadas para corrigir os problemas apontados pelos alunos, num total de 9 releases sendo elas as listadas a seguir: 0.1, 0.2, 0.3, 0.4, 1.0, 1.0.1, 1.01, 1.02, 1.03. Sendo que as versões com inicio 0 são as consideradas sem confiabilidade e com um projeto de software antigo, a estrutura apontada nas alterações demonstradas na sessão 4.2 só estão presentes na versão de inicio 1, o segundo numero indica que uma nova funcionalidade foi adicionada, como por exemplo, o localizar e substituir adicionada na versão 1.03, o terceiro numero como o que ocorreu na versão 1.0.1 indica que está verão possui um bugfix da versão 1.0, a qualidade do código contido no PortugolStudio quando a primeira release teve que ser lançada fez com que adotássemos a numeração 0, pois ele era de difícil extensão e não havia sido testado com rigor.

A constante reclamação dos alunos fez com que a evolução do PortugolStudio como IDE se tornasse prioritária, fazendo com que o cronograma sofresse modificações e parte do tempo alocado para o desenvolvimento do corretor, passou a ficar dedicado a reestruturar o PortugolStudio.

Com o uso do novo Portugol foi necessário realizar uma nova coleta de questões elaboradas pelos alunos, a coleta realizada no estudo piloto contém algoritmos com a sintaxe antiga, sintaxe essa utilizada pelo WebPortugol, a utilização de algoritmos com nova sintaxe se faz necessário pelo fato do corretor ser implementado utilizando o Portugol-Núcleo, isso implica em uma nova correção por parte dos professores para a comparação ser realizada.

A coleta de questões foi feita utilizando as questões de uma prova de algoritmos e de uma atividade curricular que continham problemas com laços, desvios e manipulação de vetores. Para coleta da atividade curricular foi utilizada o Formulário do Google com a intensão de os alunos já terem testado o algoritmo no PortugolStudio e com isso submeter apenas algoritmos que compilam e geram resultados, além de terem sido testados pelo aluno antes da submissão.

A prova de algoritmos foi dissertativa sem o auxilio do computador, o que implicou na digitalização das respostas dos alunos, nessa etapa foram selecionadas as respostas que continham algoritmos que compilavam ou que necessitavam apenas de ajustes simplórios que não implicassem na lógica de solução do aluno, como por exemplo, adicionar um abre chave.

Na coleta das avaliações dos professores foi requisitado novamente o grau de importância dos critérios, porém dessa vez o formulário que requisitava os critérios foi reformulado para dar um exemplo do que se pretende avaliar em cada critério, isso foi feito para auxiliar o professor a identificar o que o critério se refere. Esse formulário está disponível no Apêndice B.

Em seguida na Figura 13 é apresentada a média da importância de cada critério definido pelos quatro professores avaliadores que corrigiram as provas. É possível verificar que a distribuição dos pontos nos critério ficou similar ao ocorrido no estudo piloto, repetindo a ênfase nos três primeiros. Nota-se também que o Avaliador 1 concentrou a maior parte no segundo critério, ao ser questionado sobre o motivo de sua distribuição, este afirmou que ao perceber que o aluno fugiu da solução esperada ele costuma zerar a nota do aluno.

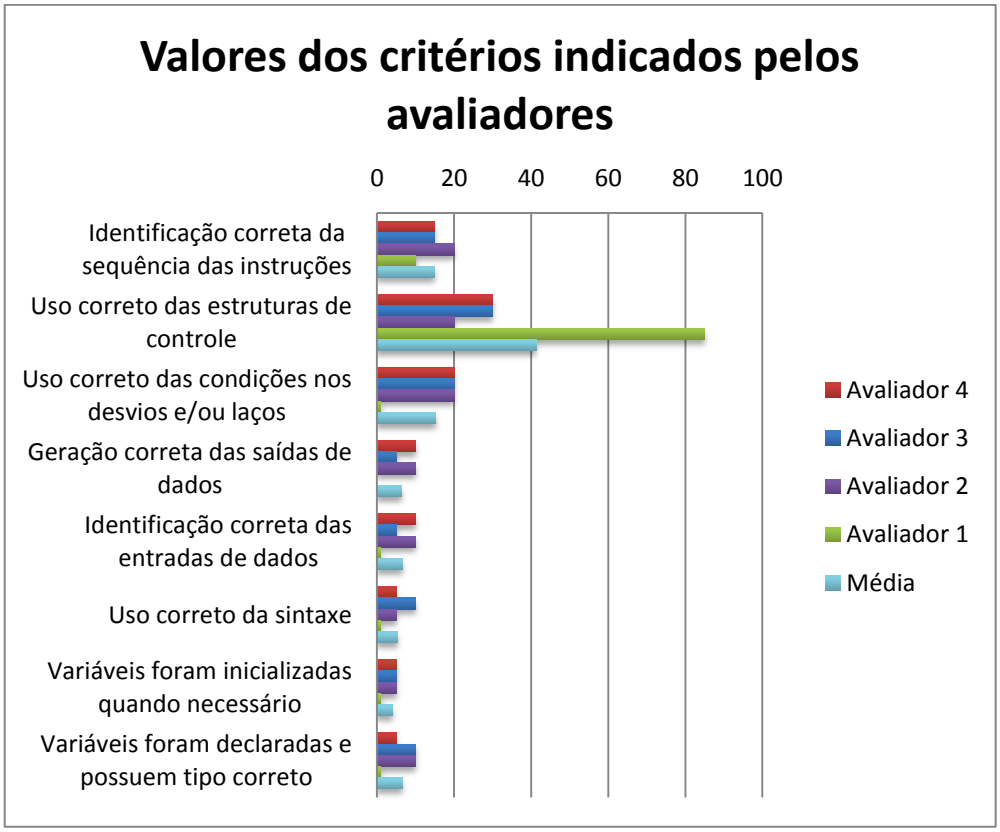

Figura 13 - Gráfico da média da importância dos critérios definido pelos dois avaliadores.

Para a comparação das correções foi coletada as respostas das questões dissertativas da prova realizada dia 28 de outubro de 2011 pelos alunos de Algoritmos e Programação 1 turma 2011/2. Uma atividade curricular também foi utilizada para coletar respostas dos alunos para comparação das correções.

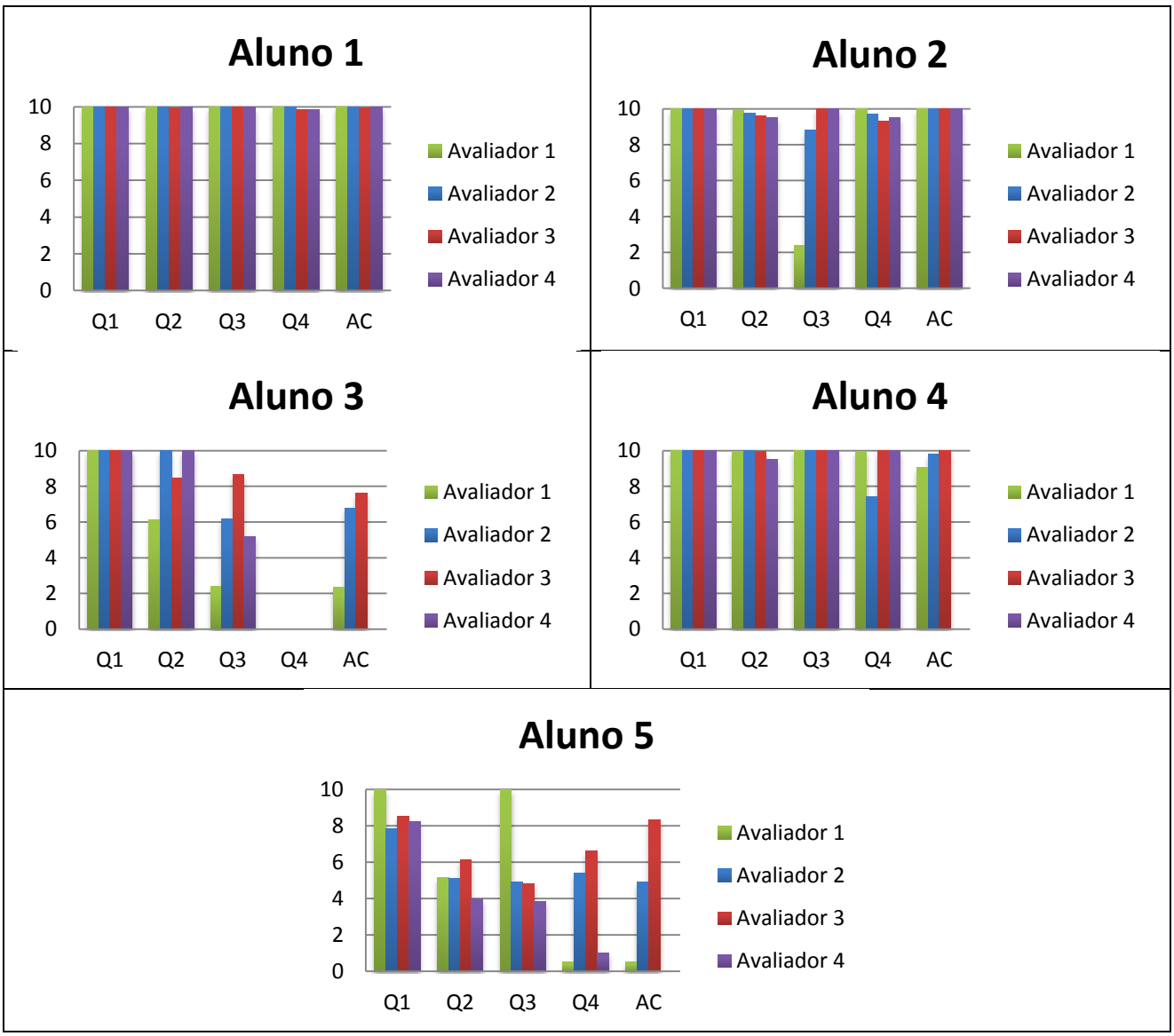

Figura 14 - Notas entregues pelos avaliadores.

É possível observar na Figura 14 as notas entregues pelos 4 avaliadores e verificar que na maior parte das questões os avaliadores deram notas similares, porem aconteceram algumas divergências, como é possível verificar na questão 3 do aluno 2 onde o avaliador 1 deu uma nota bastante baixa, foi por conta deste aluno ter usado uma solução não usual. Verifica-se que o avaliador 4 foi mais rigoroso na correção da atividade curricular, zerando alguns alunos que receberam nota dos demais avaliadores.

Na Figura 15 é possível observar os gráficos contendo a comparação da média da nota entregue pelos avaliadores e da nota entregue pelo corretor. No qual evidenciou que a implementação do corretor utilizando apenas os casos de teste prejudicou o seu desempenho e fez com que a nota recebida pelo aluno seja uma fração referente à quantidade de testes que foram bem sucedidos, como por exemplo, na questão 3 do Aluno 3, apenas um dos três casos de teste foi bem sucedida, fazendo com que neste caso ele receba um terço da nota.

Observou-se também que quando o algoritmo apresenta algum problema de execução, como laço infinito, divisão por zero, acessar índice inexistente no vetor, e etc., o corretor não consegue entregar uma nota ao aluno, deixando a questão com zero, Neste caso o corretor apenas consegue entregar ao aluno um relatório da avaliação estrutural.

Para melhorar a precisão da nota, uma estratégia que pode ser adotada é a utilização de alguma ponderação nas modificações estruturais, definindo a quantidade de nota descontada pela retirada ou inclusão de estruturas no algoritmo do aluno.

Outra possibilidade que não foi implementada é a utilização de alguns tipos de asserções falsas com o intuído de identificar algum erro que o aluno cometeu, como por exemplo, um algoritmo de ordenação de vetor, se o aluno errar a expressão e o vetor for ordenado ao contrário, uma asserção poderia verificar se o vetor está ao menos ordenando, já que os testes de ordenação crescente falhariam.

Uma problemática da atual implementação do corretor é a identificação das saídas de dados, como quando o aluno imprime outros valores de mesmo tipo de dado esperado pelo corretor, porem em ordem contrária, ou que não é a resposta efetiva.

Outro quesito que foi identificado apenas após a comparação com os professores é a verificação dinâmica de partes do código, como aconteceu com o Aluno 5 da Figura 15, a Q2 e Q3 receberam zero do corretor por não realizar a saída corretamente, mas o que fez ele receber uma pontuação intermediária dos professores foi a quase solução, não identificada pelo corretor.

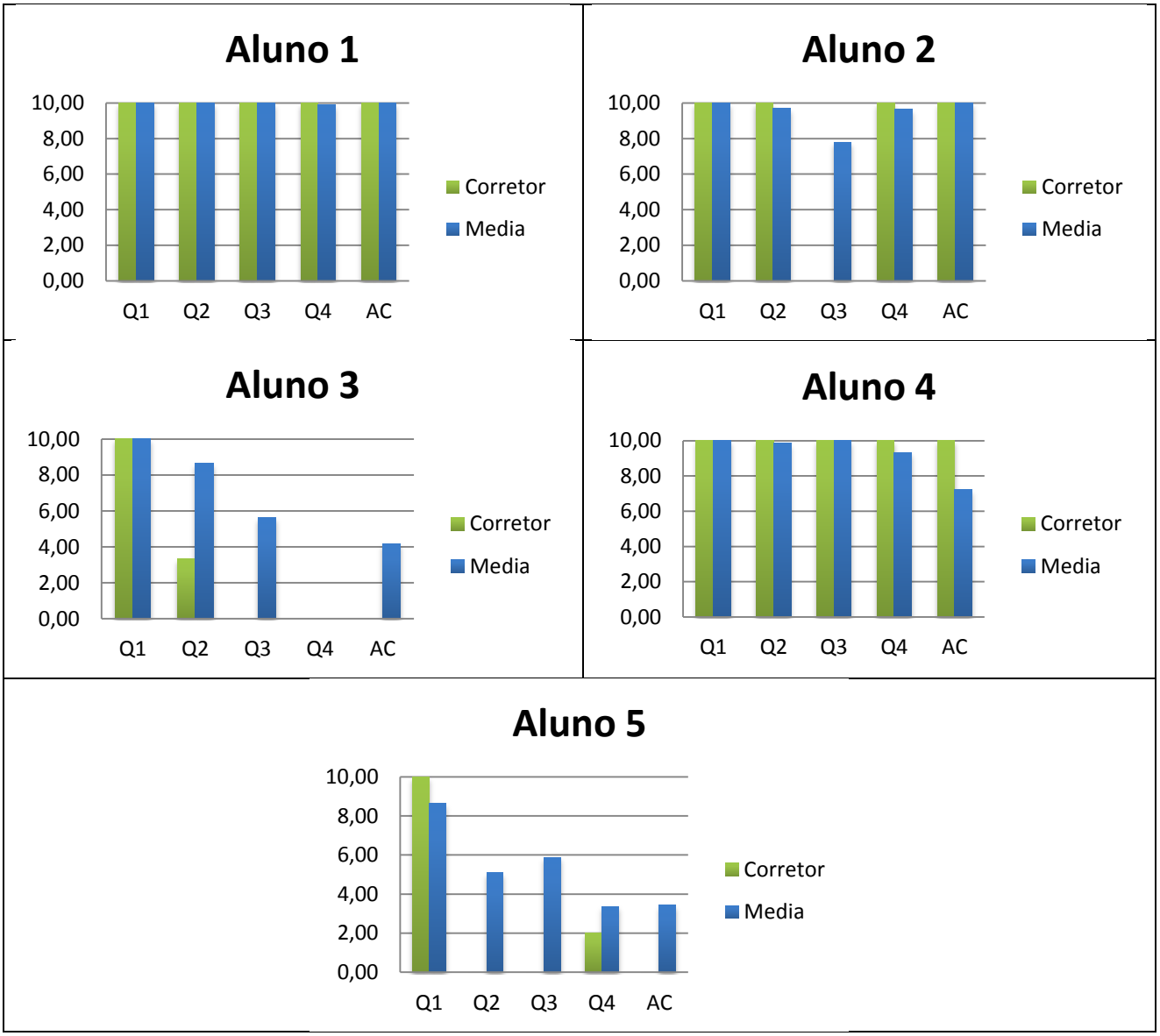

Figura 15 - Gráfico de comparação das notas entre avaliadores e corretor.

Um dos possíveis benefícios que não são apresentados nos dados coletados, pois foram feitos sem a utilização do PortugolStudio integrado ao corretor, é o potencial do corretor em entregar dicas de como alcançar a solução modelo e a demonstração dos casos falhos, podendo assim ajudar o aluno a não entregar um algoritmo incorreto.

O corretor apresenta um potencial para auxiliar os alunos, embora isso não tenha sido testado durante esse TCC, mas a expectativa é de que ele não sirva como um facilitador para o professor, poupando o trabalho de corrigir a prova, mas sim como uma ferramenta que auxilia o

aluno ao lado do depurador e compilador, a entender o que deve ser feito, e a fazer com que o aluno encontre qual parte está falha em seu algoritmo.

Muito trabalho tem de ser realizado no corretor para que este consiga um desempenho de correção similar ao professor, pois poucos ajustes na definição da nota entregue pelo corretor foram realizados durante este TCC, porem foi possível preparar um ambiente de benchmark para futuras melhorias no corretor.

5. CONCLUSÕES

Durante a elaboração deste Trabalho de Conclusão de Curso três softwares foram desenvolvidos para cumprir com o objetivo geral de “Desenvolver e avaliar a acurácia de um corretor automático de algoritmos comparando-o com correções feitas por professores.” Sendo eles o Portugol-Núcleo o PortugolStudio e o Corretor.

Para avaliar a acurácia do corretor foi realizado um estudo piloto com a intenção de preparar o benchmark de testes e definir as questões que seriam entregues aos alunos durante o semestre, além de assemelhar a correção entre diferentes professores com a utilização dos critérios definidos no estudo piloto.

Na decisão de como desenvolver o corretor automático foram feitas pesquisas na bibliografia para levantar os tipos de correção e como outros corretores funcionam, a fim de não gastar esforços criando uma solução já demonstrada como não usual por trabalhos similares.

Uma parte do esforço desse Trabalho de Conclusão de Curso foi em colocar em funcionamento o PortugolStudio e com isso fazer com que os alunos aprendam algoritmos com a nova sintaxe do Portugol durante as aulas de algoritmos 1.

No final deste trabalho foi possível comparar o corretor com a correção feita pelos professores e assim identificar os pontos críticos do corretor, porem esta etapa foi realizada tardiamente e prejudicou os ajustes na definição da nota que o corretor entrega.

Contudo o corretor criou a expectativa de se tornar uma ferramenta para auxiliar o aluno a ser autodidata, ao se integrar ao PortugolStudio e entregar ao aluno uma síntese do que pode ser melhorado em seu algoritmo, talvez colaborando para que este entenda o que deve ser resolvido no exercício.

Como trabalhos futuros é possível listar: a integração do corretor a um sistema tutor inteligente; a criação de um depurador no PortugolStudio; o empacotamento do PortugolStudio e Corretor em um objeto de aprendizado capaz de se corrigir; A comparação do desempenho de alunos que resolvem exercícios com feedback do corretor e outros sem; e a continuação do corretor para a utilização do mecanismo de comparação estrutural para colaborar na definição da nota.

REFERÊNCIAS BIBLIOGRÁFICAS

ALA-MUTKA, K. M., **A Survey of Automated Assessment Approaches for Programming Assisments.** Computer Science Education, p. 83-102, 2005.

ALA-MUTKA, K. M; JARVINEN, H.-M., **Assessment Process for Programming Assignments** Proceedings of the IEEE International Conference on Advanced Learning Technologies, IEEE Computer Society Washington, DC, USA, 2004.

CHANON, R. N. **Almost alike programs.** Proceedings of the 1966 21st national conference, p. 215–222. New York, NY, USA: ACM Press. 1966.

DALY, C.; WALDRON, J., **Assessing the assessment of programming ability** Proceedings of the 35th SIGCSE technical symposium on Computer science education, ACM, New York, NY, USA, p. 210 – 213, 2004.

DOUCE, C.; LIVINGSTONE, D.; ORWELL, J. **Automatic test-based assessment of programming: A review** Journal on Educational Resources in Computing, Volume 5 Issue 3, Article No. 4 ACM New York, NY, USA 2005.

GAMMA, E.; HELM, R.; JOHNSON, R.; VLISSIDES, J. **Design Patterns: Elements of Reusable Object-Oriented Software.** Addison-Wesley Professional; USA, 2 edition 1995.

GOLDWASSER, M. H., **A gimmick to integrate software testing throughout the curriculum.** Proceedings of the 33rd SIGCSE technical symposium on Computer Science education, p. 271–275, New York, NY, USA: ACM Press. 2002.

HEXT, J. B.; WININGS, J. W., **An automatic grading scheme for simple programming exercises**, Communications of the ACM, v.12 n.5, p.272-275, May 1969

HOSTINS, Higor ; RAABE, André Luís Alice . **Auxiliando a aprendizagem de algoritmos com a ferramenta Webportugol.** In: XIV Workshop de Educação em Computação - XXVII Congresso da SBC, 2007, Rio de Janeiro. Anais do XXVII Congresso da SBC, 2007. v. 1. p. 96-105.

HUNT, Andrew; THOMAS, Dave. **The Pragmatic Programmer: From Journeyman to Master.** Addison-Wesley 2000.

JESUS, E. A. D. **Avaliação Empírica da Utilização de um Jogo para Auxiliar a Aprendizagem de Programação**. Universidade do Vale do Itajaí. São josé. 2010.

KINNUNEN P.; MALMI, L. **Why students drop out CS1 course?** Proceedings of the second international workshop on Computing education research, New York, NY, USA p. 97 – 108, 2006.

LAHTINEN, E.; ALA-MUTKA K.; JÄRVINEN, H-M. **A study of the difficulties of novice programmers.** *In:* Annual Sigcse Conference On Innovation And Technology In Computer Science Education, 10., Caparica, Portugal. **Proceedings…** USA:ACM, 2005. p. 14-18.

MALMI, L.; KARAVIRTA, V.; KORHONEN, A.; NIKANDER, J., **Experiences on Automatically Assessed Algorithm Simulation Exercises with Different Resubmition Polices**. Educational Resources in Computing ACM, 2005.

MATHIS, Lukas. **Designed for Use: Create Usable Interfaces for Applications and the Web.** Pragmatic Bookshelf, 2011.

MIRANDA, E. M., **Protótipo de um sistema de auxílio à avaliação de algoritmos.** Universidade do vale do Itajaí. Itajaí, 2000.

MORANDI, Diana ; Pereira, Maicon ; RAABE, André Luís Alice ; ZEFERINO, Cesar Albenes . **Um Processador Básico para o Ensino de Conceitos de Arquitetura e Organização de Computadores.** Híften (Uruguaiana), v. 30, p. 73-80, 2006.

NAUDÉ, K. A. **Assessing Program Code through Static Structural Similarity** (Ms. Thesis) Faculty of Science at the Nelson Mandela Metropolitan University, 2007.

NOSCHANG, Luiz Fernando. **Adaptação Do Portugol Core Para Permitir A Integração Com Outras Ferramentas.** Universidade do Vale do Itajaí, 2012 (em fase de elaboração)

PARR, T., **Language Implementation Patterns: Create Your Own Domain-Specific and General Programming Languages.** The Pragmatic Bookshelf, 2009.

PETERSEN, A.; CRAIG, M.; ZINGARO, D., **Reviewing CS1 Exam Question Content.** Computers and Education, Dallas, Texas, USA. 2011.

RAABE, A. L. A.; DAZZI, R. L. S.; SANTIAGO, R. de. **Adquirindo experiência na construção de ferramentas de apoio a aprendizagem de algoritmos.** In: Workshop de Ambientes de Apoio a Aprendizagem de Algoritmos e Programação -, 2007, São Paulo. XVIII Simpósio Brasileiro de Informática na Educação, 2007

RAABE, André Luís Alice ; GIRAFFA, Lúcia Maria Martins . **Uma Arquitetura de Tutor para Promover Experiências de Aprendizagem Mediadas.** In: XVII SImpósio Brasileiro de Informática na Educação - SBIE2006, 2006, Brasília - DF. Anais do XVII SImpósio Brasileiro de Informática na Educação, 2006. v. 1. p. 589-598.

RAABE, André Luís Alice ; SILVA, Júlia Marques Carvalho da **. Um ambiente para atendimento as dificuldades de aprendizagem de algoritmos.** In: XIII Workshop de Educação em Computação - SBC2005, 2005, São Leopoldo. Anais do XXV Congresso da Sociedade Brasileira de Computação, 2005. p. 2326-2335.

RAABE, A. L. A. **Uma Proposta de Arquitetura de Sistema Tutor Inteligente Baseada na Teoria das Esperiências de Aprendizagem Mediadas**. Universidade Federal do Rio Grande do Sul. Porto Alegre. 2005.

RAABE, A. L. A.; DA SILVA, J. M. C. Um Ambiente para Atendimento as Dificuldades de. **XXV Congresso da Sociedade Brasileira de Computação**, São Leopoldo, 22 a 29 Julho 2005. 2326-2337.

RAHMAN, K. A.; AHMAD, S. ; NORDIN, M. J; MAKLUMAT , F. T. D. S., **The Design of an Automated C Programming Assessment Using Pseudo-code Comparison Technique.** IEEE Information Technology, 2009.

RAHMAN, K. A.; NORDIN, M. J**. A Review on the Static Analysis Approach in the Automated Programming Assessment Systems.** NATIONAL CONFERENCE ON PROGRAMMING 07 **Proceedings…**, Kuala Lumpur, Malaysia, 2007.

REEK, K. A. **The TRY system -or- how to avoid testing student programs**, ACM SIGCSE Bulletin, v.21 n.1, p.112-116, Feb. 1989

REES, M. J. **Automatic assessment aids for Pascal programs**. SIGPLAN Not., 17(10), p. 33-42, 1982.

ROHAIDA, R., **Penjanaan Data Ujian untuk Penaksiran Automatik Tugasan Aturcara C**, Universiti Kebangsaan Malaysia, 2003.

SAIKKONEN, R.; MALMI, L.; KORHONEN, A. **Fully automatic assessment of programming exercises**. ACM SIGCSE Bulletin, USA, v. 33, n. 3, p. 133-136, 2001.

SCHEFFLER, P., **Teaching Algorithmics – Theory and Practice.** In: Proc. 2nd Intern. Sc. Conf. "Informatics in the Scientific Knowledge." Varna 2008, pp. 259-269.

SCHULTE, C.; BENNEDSEN, J., **What do teachers teach in introductory programming?** Proceedings of the 2006 international workshop on Computing education research, Canterbury, United Kingdom, 2006.

TEW, A. E.; GUZDIAL, M., **Developing a validated assessment of fundamental CS1 concepts.** Proceedings of the 41st ACM technical symposium on Computer science education, New York, NY, USA, 2010

TSINTSIFAS, A. **A framework for the computer based assessment of diagram based coursework** (PhD thesis), University of Nottingham, School of Computer Science, 2002.

TRUONG, N.; ROE P.; BANCROFT, P. **Static analysis of students' Java programs.** *In:* Conference On Australasian Computing Education, 6., Dunedin, New Zealand. **Proceedings…** Darlinghurst:Australian Computer Society, jan. 2004. p. 317-325.

VAHLDICK, A. ; RAABE, André Luís Alice **. Adaptação de Conteúdo SCORM em Ambientes Inteligentes de Aprendizagem.** In: Simpósio Brasileiro de Informática na Educação, 2008, Fortaleza. Anais do XIX Simpósio Brasileiro de Informática na Educação, 2008. v. 1. p. 1-10.

VALLE, G. ; NOSCHANG, L. F. ; MIRANDA, E. M. de ; SILVA, J. M. C. da ; RAABE, A. L. A. . **Adaptação da Ferramenta Webportugol para o curso de Tecnologia em Sistemas para Internet.** In: XX Simpósio Brasileiro de Informática na Educação, 2009, Florianópolis. Anais do XX Simpósio Brasileiro de Informática na Educação, 2009. v. 1.

VIEIRA, Paulo; ZEFERINO, Cesar; RAABE, André. **Bipide: Ambiente de Desenvolvimento Integrado para Utilização dos Processadores BIP no Ensino de Programação**. XX Simpósio Brasileiro de Informática na Educação - SBIE2009, 2009, Florianópolis - SC. Anais do XX Simpósio Brasileiro de Informática na Educação, 2009. v. 1.

WANG, T. ; SU, X.; MA, P; WANG, Y; WANG, K; **Ability-training-oriented automated assessment in introductory programing course.** Computer & Education , China, 2010, 220-226

APÊNDICES

APÊNDICE A - LISTA DE EXERCÍCIOS

APRESENTAÇÃO

Esta lista de 60 problemas visa proporcionar oportunidades para os alunos consolidarem os conceitos fundamentais da lógica de programação. Acredita-se que aluno que conseguir resolver todos os problemas desta lista estará apto a resolver problemas mais complexos e a programar em qualquer linguagem de programação.

Os problemas listados estão classificados com complexidade crescente nas categorias apresentadas a seguir:

1. Algoritmos Sequenciais – 10 Problemas
2. Desvio Condicional – 10 Problemas
3. Laço de Repetição – 20 Problemas
4. Vetores e Matrizes – 20 Problemas

Os problemas são voltados para interface via console, ou seja, a entrada de dados se dará via teclado e a saída de dados na tela e em modo texto.
Bom proveito.

Algoritmos Sequenciais

1. Faça um programa que exiba na tela a mensagem "Olá Mundo !".
2. Faça um programa que solicita que o usuário digite o seu nome e exiba a mensagem "Olá" seguido do nome digitado pelo usuário.
3. Faça um programa que solicita ao usuário um número real e exibe na tela a metade do número digitado.
4. Faça um programa que calcula os gastos com combustível em uma viagem. O programa deve solicitar ao usuário a distância a ser percorrida em Km, o consumo do carro em Km/litro e o preço do litro do combustível. Como resposta o programa deverá informar qual o valor em R$ a ser gasto com combustível na viagem.
5. Faça um programa que solicita ao usuário dois números inteiros e armazena nas variáveis A e B. A seguir (utilizando apenas atribuições entre variáveis) troque os seus conteúdos fazendo com que o valor que está em A passe para B e vice-versa. Ao final exiba na tela os valores que ficaram armazenados nas variáveis.
6. O sistema de avaliação de determinada disciplina, é composto por três provas. A primeira prova tem peso 2, a segunda tem peso 4 e a terceira prova tem peso 6. Faça um programa que solicita as notas para o aluno, calcula e exibe a média final deste aluno.
7. Faça um programa para um terminal de autoatendimento bancário que realiza saques em dinheiro. O programa deve solicitar ao usuário qual o valor a ser retirado e deve exibir na tela qual a quantidade de cada cédula será entregue ao usuário. O programa sempre ceve tentar utilizar o menor número possível de cédulas. A máquina possui apenas cédulas de R$ 10, 5 e 1.
8. Faça um programa para calcular e exibir o digito verificador de uma conta bancária. O usuário deve digitar o número da conta que deve ser um número inteiro com 4 dígitos. O digito verificador será calculado como segue:

   a. Passo 1: Somar todos os quatro dígitos
   b. Passo 2: Multiplicar todos os quatro dígitos
   c. Passo 3: Subtrair o resultado da multiplicação (passo 2) pelo resultado da soma (passo 1)
   d. Passo 4: O digito verificador será o resto da divisão do resultado da subtração (passo 3) por 9.
9. Faça um programa que solicita ao usuário um número inteiro com três dígitos e exibe o número invertido (ex: usuário digitou 136, programa exibirá 631).
10. Faça um programa que converta um número decimal digitado pelo usuário em binário. O programa deve funcionar somente para números de 0 a 15.

Desvio Condicional

1. Faça um programa que solicita ao usuário um número inteiro e exibe este na tela. Se o número for negativo, antes de ser exibido, ele deve ser transformado no equivalente positivo.
2. Faça um programa que solicita ao usuário um valor inteiro e exibe uma mensagem informando se o número é par ou ímpar.
3. Faça um programa que solicita ao usuário uma letra e verifique se ela é uma vogal ou não exibindo uma mensagem correspondente.
4. Faça um programa que solicita a data de nascimento de uma pessoa e a data atual e exiba a idade desta pessoa em anos (A data deve ser armazenada em 3 variáveis inteiras para ano, mês e dia).
5. Faça um programa que solicita ao usuário dois números inteiros. O primeiro é o valor das horas e o segundo dos minutos. Verifique se a hora é válida e exiba uma mensagem correspondente (considere a hora 24:00 como inválida).
6. Faça um programa que exiba o maior dentre dois números reais digitados pelo usuário. Caso eles sejam iguais exiba uma mensagem correspondente.
7. Faça um programa que solicita um número inteiro e exibe uma mensagem indicando se ele é positivo, negativo ou zero.
8. Faça um programa que solicita ao usuário três números reais e exibe na tela apenas o menor deles.
9. Faça um programa que solicita ao usuário seu nome e as notas de três provas. Calcule a média aritmética e informe se o aluno foi Aprovado ou Reprovado (o aluno é considerado aprovado com a média igual ou superior a 6).
10. Faça um programa que solicita ao usuário três valores correspondentes aos lados de um triângulo. Informe se o triângulo é eqüilátero (possui 3 lados iguais), isósceles (possui dois lados iguais) ou escaleno (não possui lados iguais).
11. Faça um programa que solicita ao usuário para pensar um número de 1 até 8. A seguir o programa deve "adivinhar" o número que o usuário pensou, sendo que para isso poderá fazer no máximo três perguntas ao usuário cuja resposta deve ser sim ou não.

Laços de Repetição

1. Faça um programa que exiba 30 vezes na tela a mensagem "Não vou colar na prova".

2. Faça um programa que exiba na tela a tabuada do número 5 no seguinte formato: 5X1=5; 5X2=10; 5X3=15; ... ; 5X10=50.
3. Faça um programa que exiba na tela os números inteiros de 100 até 1.
4. Faça um programa que exiba na tela os números inteiros de 50 até 200.
5. Faça um programa que exiba na tela a soma dos números inteiros do intervalo [100, 200].
6. Faça um programa que solicita ao usuário dois valores inteiros e positivos que serão a base e o expoente. O programa deve calcular e escrever o resultado da base elevado à potência.
7. Faça um programa que solicita ao usuário uma quantidade indeterminada de números inteiros. O programa deve calcular e escrever a média aritmética apenas dos números pares. A entrada de dados deve ser encerrada quando o número ZERO for digitado.
8. Faça um programa que solicita ao usuário um número real positivo. Verifique se o número é realmente positivo, e em caso contrário solicite ao usuário digitar novamente (este processo pode se repetir inúmeras vezes e é chamado de consistência, pois garante que o número será válido após a entrada de dados).
9. Faça um programa para uma calculadora simples que solicita ao usuário dois operandos como entrada, seleciona uma das opções da lista (1- soma, 2- produto, 3- divisão, 4- potência) e exibe o resultado. O algoritmo deve executar repetidamente até que os dois operandos informados sejam iguais a zero.
10. Faça um programa que gere e exiba os 20 primeiros termos da série de Fibonacci (Os dois primeiros termos da série são 1 e 1, os termos subseqüentes são a soma dos dois últimos. Observe o exemplo: 1,1,2,3,5,8,13,21,..)
11. Faça um programa que solicita ao usuário o valor de N e calcule o valor de S na série S = 1/1 + 1/2 + 1/3 + ... + 1/N.
12. Faça um programa que solicita a idade de 10 pessoas e exiba a quantidade de pessoas que possui idade maior ou igual a 18 anos.
13. Faça um algoritmo que leia um conjunto de 20 números inteiros e indique, ao final, qual foi o menor valor digitado.
14. Faça um programa que solicita o peso de 25 pessoas e exibe qual o maior peso e qual o menor peso dentre os digitados.
15. Faça um programa que solicita 20 valores inteiros e exiba quantos são pares e quantos são ímpares.
16. Solicite ao usuário a digitação de um número inteiro, calcule e exiba o fatorial deste número
17. Faça um programa que verifica se um número digitado pelo usuário é um número primo, ou seja, um número que só é divisível por 1 e por ele mesmo.
18. Faça um programa que exiba na tela a tabuada de todos os números desde o 1 até o 10. Use o seguinte formato: 5X1=5; 5X2=10; 5X3=15; ... ; 5X10=50.
19. Faça um programa que exiba na tela os 20 primeiros números primos após o 100.
20. Faça um programa que solicita ao usuário o número de alunos de uma turma e o número de provas realizadas. A seguir o programa deve para cada aluno:
    a. Solicitar o nome do aluno
    b. Para cada prova realizada solicita a nota deste aluno
    c. Exibir o nome e a média aritmética das notas deste aluno

    Após o término da digitação o programa deverá exibir o nome do aluno com maior média.

Vetores

1. Faça um programa que solicita ao usuário 5 valores inteiros e armazene estes em um vetor.
2. Faça um programa que solicita ao usuário 50 valores reais e armazene em um vetor. Percorra este vetor e calcule e exiba a média aritmética dos valores.
3. Faça um programa que solicita ao usuário 30 valores e jogue os pares em um vetor e os ímpares em outro. Após a leitura calcule o somatório dos dois vetores e exiba o de maior valor.
4. Faça um programa em que o usuário preencha dois vetores de 10 posições. O programa deve fazer a multiplicação dos elementos de mesmo índice, colocando o resultado em um terceiro vetor. Exiba o vetor resultante.
5. Faça um programa que leia um vetor de 100 posições e mostre-o ordenado em ordem crescente.
6. Faça um programa que solicita ao usuário: código e quantidade vendida de 100 produtos de uma lanchonete. Armazene os códigos em um vetor e as quantidades em outro vetor. Após a entrada de dados, exiba um ranking dos produtos mais vendidos com seus respectivos códigos.
7. Faça um programa que solicita ao usuário 50 valores reais e armazene em um vetor. O programa deve verificar e exibir quantos elementos não repetidos (diferentes) existem neste vetor.
8. Faça um programa que solicita ao usuário 10 valores inteiros e armazene estes em um vetor. Após o programa deve exibir qual a posição (índice do vetor) do elemento de maior valor.
9. Faça um programa que solicita ao usuário 10 valores inteiros, armazene estes em um vetor. Após o programa deve verificar se o número 7 se encontra no vetor. Em caso positivo, exiba qual a posição em que ele foi encontrado. Se ele for encontrado mais de uma vez também quantas vezes ele foi encontrado.
10. Faça um programa que solicita ao usuário 8 valores inteiros, armazene estes em um vetor. Copie todo o conteúdo do vetor para um segundo vetor de forma invertida e exiba-o na tela.
11. Faça um programa que preenche dois vetores de 5 posições inteiras. A seguir crie um novo vetor contendo a união dos elementos dos dois vetores (A união deve excluir elementos repetidos). Exiba os vetores originais e o vetor união.
12. Faça um programa que preenche dois vetores de 5 posições inteiras. A seguir crie um novo vetor contendo a intersecção dos elementos dos dois vetores (O vetor intersecção deve possuir apenas os elementos que constam em ambos os vetores). Exiba os vetores originais e o vetor intersecção.

Matrizes

1. Faça um programa que preencha uma matriz de 5X5 com o elemento zero em todas as posições.
2. Faça um programa que preencha uma matriz de 5X5 com o elemento zero em todas as posições em que o índice de linha tem valor igual ao da coluna.
3. Faça um programa que solicita ao usuário 25 valores reais e armazene em uma matriz 5x5. A seguir troque todos os elementos da matriz que sejam maiores do que 100 pelo valor zero. Exiba a matriz original e alterada.

4. Faça um programa que preencha uma matriz de 5X5 com o elemento zero em todas as posições em que o índice de linha tem valor maior que o da coluna.
5. Faça um programa que manipula uma matriz quadrada de qualquer tamanho, somando os valores que não estão nem na diagonal principal nem na diagonal secundaria.
6. Faça um programa que solicita ao usuário 16 valores reais e armazene em uma matriz 4x4. O programa deve somar os elementos de cada uma das linhas armazenando o resultado da soma em um vetor. A seguir, deve multiplicar cada elemento da matriz pela soma da sua respectiva linha. Exiba na tela a matriz resultante.
7. Faça um programa que preenche duas matrizes, uma M (4X6) e outra N (6X4). A seguir o programa deverá criar uma nova matriz que seja o produto matricial de M por N.
8. Faça um programa que preenche uma matriz de 4X4 com números reais. O programa deve exibir qual a posição (linha e coluna) do elemento minimax, ou seja, o menor valor que esteja na linha em que se encontra o maior valor da matriz.

APÊNDICE B – FICHA DE CRIÉRIOS

Docente:_____

Por favor, informe na tabela abaixo conforme o grau de importância a ser atribuido a cada critério nas suas avaliações. A soma dos critérios deve resultar em 100 pontos.

Critério	**Importância**
A. Identificação correta da sequência das instruções	
B. Uso correto das estruturas de controle	
C. Uso correto das condições nos desvios e laços	
D. Geração correta das saídas de dados	
E. Identificação correta das entradas de dados	
F. Uso correto da sintaxe	
G. Variáveis foram inicializadas quando necessário	
H. Variáveis foram declaradas e possuem tipo correto	

A. identificação correta da sequência das instruções: busca avaliar se o aluno coloca as instruções na sequência esperada para um correto funcionamento do programa, como por exemplo: colocar o contador dentro do laço de repetição, colocar o escreva antes do leia.

B. Uso correto das estruturas de controle: busca avaliar se o aluno identifica e usa as estruturas (laços e desvios) necessários para resolver o problema, sem utilizar estruturas a mais ou a menos.

C. Uso correto das condições de desvios e laços: busca avaliar se as expressões dos desvios condicionais e laços de repetição foram definidas corretamente conforme o problema e a estrutura da solução.

D. Identificação correta das saídas de dados: busca avaliar se o aluno identificou corretamente quais são as informações que devem ser exibidas na tela. Ex: Se em um problema de média ele errar a fórmula, mas exibir o conteúdo da variável média ele identificou corretamente o que deveria ser exibido.

E. Identificação correta da entrada de dados: busca avaliar se o aluno identificou corretamente quais as informações devem ser solicitadas ao usuário e em que variáveis deve armazená-las, em outras palavras, quando o aluno utiliza o leia de forma correta e no momento adequado para solucionar o problema.

F. Uso correto da sintaxe: busca avaliar se o aluno escreveu um código compilável com a linguagem, sem erros sintáticos.

G. Variáveis foram inicializadas quando necessário: busca avaliar se todas as variáveis que tiveram seu valor recuperado da memória foram antes inicializadas, normalmente se aplica a contadores e somadores.

H. Variáveis foram declaradas e possuem tipo correto: busca avaliar se o aluno declarou e está utilizando variáveis com tipo de dados correto para a resolução do problema.