UNIVERSIDADE ESTADUAL DO NORTE FLUMINENSE DARCY RIBEIRO
GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

MARCELO OLIVEIRA MELO

REGRESSÃO LINEAR MÚLTIPLA NA AVALIAÇÃO DE EXERCÍCIOS DE PROGRAMAÇÃO

CAMPOS DOS GOYTACAZES
2013

MARCELO OLIVEIRA MELO

Regressão linear múltipla na Avaliação de Exercícios de Programação

Monografia apresentada junto ao Curso de Ciência da Computação, da Universidade Estadual do Norte Fluminense Darcy Ribeiro – Campos / RJ, como requisito para obtenção do título de Bacharel em Ciência da Computação.
Orientadora: Profª. Dra. Annabell Del Real Tamariz

CAMPOS DOS GOYTACAZES
2013

MARCELO OLIVEIRA MELO

Regressão linear múltipla na Avaliação de Exercícios de Programação

Monografia apresentada junto ao Curso de Ciência da Computação, da Universidade Estadual do Norte Fluminense Darcy Ribeiro – Campos / RJ, como requisito para obtenção do título de Bacharel em Ciência da Computação.
Orientadora: Profª Dra. Annabell Del Real Tamariz

Aprovado em 16/08/2013

COMISSÃO EXAMINADORA

_____

Profª Dra. Annabell Del Real Tamariz
Orientadora – Universidade Estadual do Norte Fluminense UENF

_____

Prof. Angel Coca Balta
Universidade Estadual do Norte Fluminense UENF/LCMAT

_____

Prof. Fermin A. Tang Montané
Universidade Estadual do Norte Fluminense UENF/LCMAT

.

DEDICATÓRIA

Dedico este trabalho a Deus e a Oshieoyá-Samá por terem me apoiado nos momentos fáceis e difíceis. E aos meus pais por terem me ajudado de diversas formas a chegar até aqui.

AGRADECIMENTOS

Agradeço a Deus e a Oshieoyá-Samá por terem me ajudado tanto nos momentos fáceis quanto nos difíceis.

Agradeço aos meus pais Adjanes e Neida, que são a minha base. Muito obrigado por terem me apoiado de diversas formas e me permitido fazer faculdade.

Á minha orientadora Annabell por ter tido paciência comigo durante a orientação e ter me confiado esse projeto. Muito obrigado orientadora.

Ao meu professor Rodrigo por ter me ajudado em boa parte técnica desta monografia.

A mulher inesquecível na minha vida Ana Paula por ter revisado todo este texto e também por ter me auxiliado de várias formas durante todo o curso.

E à todos os amigos e familiares que de alguma forma contribuíram durante o desenrolar do curso com apoio moral ou nos trabalhos.

“Errei mais de 9 mil cestas e perdi quase 300 jogos.
Em 26 diferentes finais de partidas
fui encarregado de jogar a
bola que venceria o jogo e falhei.
Eu tenho uma história repleta de falhas e
fracassos em minha vida.
E é exatamente por isso que sou um sucesso."
Michael Jordan

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE TERMOS

Caso de uso - Documento de teste onde são detalhados como um determinado requisito deve comportar-se.

Resumo

Este trabalho visa apresentar um novo modelo de avaliação automática de exercícios de programação usando regressão linear múltipla e métricas de engenharia de software, definidos, propostos e implementados nesta monografia. O modelo de avaliação foi implementado em um sistema computacional onde, através de testes de unidade automatizados e algumas métricas de engenharia de software, consegue realizar uma avaliação automática no código do aluno, obtendo no final a nota que o mesmo conseguiu no desenvolvimento de exercícios de programação.

PALAVRAS-CHAVE: Avaliação de Exercícios, Teste de Unidade, Regressão Linear Múltipla, interface gráfica.

Abstract

This work show a new model of automated assessment of programming exercises with multiple linear regression and some software engineering metrics, defined, proposed and programmed in this work. The assessment´s model was programmed in a system, where is applied applies automatic assessment in student´s code through unit tests and some software engineering metrics, obtaining in the end the final mark and your situation of mastery in programming.

KEYWORDS: Automated assessment, unit test, multiple linear regression.

1 Introdução

A programação constitui a base para muitos campos em que a informática se aplica, assim como capacita o indivíduo a utilizar a lógica de programação na resolução de problemas, fator muito importante em disciplinas avançadas. Segundo [MOREIRA,2009b] programação deve ser aprendida na prática, então uma possibilidade de avaliar se um determinado aluno sabe programação ou não é fazer testes pedindo para ele escrever código. Neste sentido pensou-se, neste trabalho, fazer avaliações com o aluno pedindo para ele resolver os problemas escrevendo código computacional. Em seguida, foi adotada uma metodologia que combina método para revisar o código digitado, através de um compilador, um processo para comparar o código escrito pelo estudante com uma especificação funcional através dos testes de unidade e um modelo que consiga avaliar características específicas do código através das métricas e da regressão linear múltipla. Desta forma foi possível determinar se o aluno sabe ou não programação. Para o desenvolvimento desta pesquisa foi aplicada a técnica de teste conhecida como teste de unidade, para a validação do código e algumas métricas adotadas por [MOREIRA,2009a] em conjunto com um modelo de regressão linear múltipla para a análise quantitativa do código.

1.1. Objetivo Geral

Este trabalho tem como objetivo desenvolver um modelo de avaliação automática de exercícios de programação, de forma que professores e alunos possam ter um *feedback* rápido e preciso das atividades que estão realizando, melhorando desta forma a aprendizagem.

1.2. Objetivos Específicos

Para o alcance do objetivo geral, este trabalho também tem a finalidade de:

- Definir e implementar os testes de unidade, uma das principais formas de avaliação de código existentes na atualidade.
- Definir e aplicar as principais métricas de engenharia de software e o método da regressão linear múltipla, para que auxiliassem na avaliação dos exercícios.
- Apresentar o sistema proposto.
- Apresentar a forma em que são guardados os dados dos alunos que usam o sistema para futuras análises.

1.3. Justificativas

Vive-se em uma época em que o conhecimento se faz necessário em todos os aspectos do cotidiano, porém o problema é como obter esse conhecimento visto que existe uma gama de fatores que exercem influência retardando o processo. O desafio no aprendizado de programação pode ser consequência de vários fatores, tais como: base matemática fraca, má compreensão do problema e falta de entendimento do assunto. Dificuldades no aprendizado de programação refletem em altos índices de reprovação e consequentemente em mau desempenho do aluno em outras matérias que têm programação como base [MOREIRA,2009b].

Em [MOREIRA,2009a] apresenta-se uma dificuldade relevante encontrada pelos educadores no processo de ensino-aprendizagem de programação usando ambientes virtuais:

Como avaliar manualmente todos os exercícios dos alunos, para turmas grandes?

Considerando que na programação é comum achar várias soluções algorítmicas para um mesmo problema este trabalho apresenta uma metodologia adequada ao problema com o intuito de responder a dificuldade apresentada em [MOREIRA,2009a]. Pois é preciso adotar um modelo que se adapte, adequadamente, à situação em questão.

Após pesquisar na literatura pôde-se perceber que apesar de haver trabalhos como os de Karavirta [KARAVIRTA,2011] e Shamsi [SHAMSI,2012] que realizam avaliação automática através de testes de unidade e análise estática, não foi encontrado para a linguagem C , um avaliador que realize avaliação dinâmica através de teste de unidade, sendo este tipo de sistema importante para esta linguagem, devido a linguagem C ser muito utilizada para o ensino de Introdução à Programação [JANICIC,2012] [GUO,2006]. Assim para contribuir para a área de pesquisa este trabalho visa o desenvolvimento e implementação de técnicas de avaliação métrica que considerem os conhecimentos do aluno, no conteúdo específico de programação para a linguagem C utilizando testes de unidade para a análise qualitativa do código e o modelo de Regressão Linear Múltipla em conjunto com as métricas de engenharia de software para a análise quantitativa do mesmo.

1.4. Estrutura do Trabalho

O presente capítulo expõe uma breve introdução do tema com o intuito de familiarizar os leitores com o problema abordado, a justificativa para o desenvolvimento desta pesquisa e seus objetivos. O capítulo 2 exibe diversos trabalhos relacionados com este estudo, principalmente os que mais contribuíram para o embasamento teórico desta pesquisa. O capítulo 3 explana sobre a avaliação automática de exercícios, suas origens e suas classificações segundo a literatura. Também são abordados no capítulo 3 os Testes de Unidade e as Métricas de Engenharia de Software, técnicas que são utilizadas na aplicação deste trabalho. O capítulo 4 aborda sobre a regressão linear múltipla,

bem como quais são os dados que são utilizados nesta técnica e como ela é utilizada neste trabalho como método que beneficia a avaliação. O capítulo 5 apresenta o modelo proposto, bem como sua utilização para realizar a avaliação automática de exercícios de programação de um aluno para determinar se ele sabe programação ou não. Também é apresentada a implementação do sistema e como ele deve ser implantado, de forma que o aluno possa usufruir dos seus recursos. Já no capítulo 6 é trabalhado um estudo de caso, onde são expostas as possibilidades do sistema. Já nos capítulos 7 e 8 são apresentadas as conclusões e os trabalhos futuros que podem ser acrescentados no sistema, respectivamente.

2 Trabalhos relacionados

Existem alguns trabalhos relacionados com esta área de pesquisa e nesta seção foram citados os que mais contribuíram para a formação desta pesquisa.

O trabalho desenvolvido por [RAHMAN,2011] avalia os programas dos alunos em linguagem C, realizando análise estática através de uma técnica de conversão do código em pseudocódigo. A avaliação é feita em três fases: Análise léxica, geração do pseudocódigo e comparação do pseudocódigo.

A análise léxica consiste em transformar o código-fonte em tokens e analisá-lo, igual é feito nos compiladores. No trabalho de Rahman contém todos os símbolos que são aceitos na análise léxica. A lista de tokens é usada como entrada para a geração do pseudocódigo, que costuma simplificar a comparação dos mesmos. O padrão de pseudocódigo adotado no trabalho de Rahman foi um proposto por [ROBERTSON,2004], já que não existe um padrão consensual do mesmo. A conversão é feita utilizando uma tabela de conversão, que especifica a relação entre as sentenças do código-fonte e os comandos em pseudocódigo.

Após a conversão para pseudocódigo, é realizada a comparação com os pseudocódigos do professor. A avaliação é feita de duas maneiras diferentes: Uma através dos tipos e valores das variáveis e a outra através da similaridade entre os dois pseudocódigos. O professor tem que inserir diversas soluções-modelo para que o sistema possa corrigir as várias soluções dos alunos. O programa do aluno é comparado com todas as soluções-modelo e a solução que for mais compatível com o programa do aluno será a usada para aplicar a avaliação deste.

Uma extensão do sistema Moodle foi desenvolvida por [MOREIRA,2009a] para a avaliação de programas Java, utilizando métricas de engenharia de software e regressão linear múltipla. Este sistema utiliza as seguintes métricas de engenharia de software: A quantidade de uso de funções pré-definidas da linguagem, o número de palavras reservadas, o número de declarações, o número de linhas do programa, a complexidade ciclomática de McCabe e o volume de Halstead. E para retornar a nota final do aluno, esta extensão do sistema Moodle faz o uso da Regressão Linear Múltipla. Observando que o sistema desenvolvido por [MOREIRA,2009a] não avalia os resultados dos programas dos alunos, o trabalho desta monografia buscou resolver este impasse através da adoção dos Testes de Unidade.

Uma aplicação WEB (AWTM) para realizar Teste de Mesa em algoritmos dos alunos foi desenvolvida por [MEDEIROS,2002]. O Teste de Mesa é um teste onde é feita uma simulação do algoritmo, em que uma tabela com os valores das variáveis e constantes vai sendo montada e inspecionada e no final da inspeção uma das três conclusões é inferida: o algoritmo obteve o resultado esperado, o algoritmo contém erros nos comandos ou o algoritmo contém erros no fluxo de execução (erro nas estruturas de controle).

Em sua tese [SYMEONIDIS,2006] criou um sistema que aplica avaliações automáticas em programas Java, chamado CourseMarker e neste sistema as avaliações dos programas dos alunos são realizadas dinamicamente através de expressões regulares que delimitam o que deve ter em uma solução do aluno ou não. Os programas dos alunos são rodados com um conjunto de dados de teste pré-definidos, que servem como dados de entrada de forma que o sistema possa controlar o fluxo de execução do programa. Ao executar o sistema, o programa do aluno produz uma saída, que é comparada com um conjunto de padrões de expressões regulares, tendo no final um feedback com o resultado de cada teste.

Para fundamentar a avaliação automática de exercícios foi pesquisado um sistema em JavaScript proposto por [KARAVIRTA,2011], que avalia aspectos dinâmicos e estáticos.

A **Avaliação Dinâmica** é feita em 4 categorias:

- *Funcionalidade*: Testa se a solução do aluno atende aos requisitos do exercício. Normalmente é feita inserindo dados de entrada e checando os dados de saída do programa. A ferramenta de teste de unidade JSUnit[1] e o framework de teste Selenium[2] foram utilizados para esta tarefa.
- *Eficiência*: Avalia o uso do tempo, da memória, da rede e outros. A ferramenta hrtimer[3] e o sistema Dromaeo[4] foram utilizados para esta tarefa.
- *Habilidades para testar*: avalia a qualidade dos testes que os estudantes estão fazendo para seus programas. O sistema incentiva também que os alunos façam testes para seus próprios programas. A avaliação é feita através de uma ferramenta chamada JSCoverage[5], que analisa o quanto os testes do aluno cobre o programa do mesmo.

1 http://www.jsunit.net/

2 http://seleniumhq.org/

3 http://hrtimer.mozdev.org/

4 https://wiki.mozilla.org/Dromaeo

5 http://siliconforks.com/jscoverage/

- *Características especiais*: Algumas vezes pode ser interessante para o aluno aprender características específicas de determinado navegador. Características especiais ou específicas se encarregam de avaliar e prover feedback da solução do aluno em diferentes navegadores. Esta tarefa é possível através das ferramentas TestSwarm[6] e JsTestDriver[7], que aplicam os testes nos programas dos estudantes em diferentes navegadores e coletam os resultados.

A **Análise Estática** é feita em 5 categorias:

- *Estilo*: Não há um padrão consensual. Algumas convenções foram adotadas a respeito deste, como identação, espaço entre as linhas e outras. A ferramenta JSLint[8] é a responsável por esta tarefa.
- *Erros de programação*: Esta categoria é responsável por alguns erros como variáveis não inicializadas e falta de ponto e vírgula no final do comando. As ferramentas utilizadas são a JSLint e a JavaScriptLint[9].
- *Métricas de engenharia de software*: Lidam com métricas como complexidade ciclomática e quantidade de linhas de código. JSMeter[10] é a ferramenta responsável por esta categoria.
- *Design*: É responsável por detectar padrões de projeto e implementação de algoritmos de ordenação nos programas. Não foi encontrada nenhuma ferramenta para automatizar essa categoria.
- *Características especiais ou específicas*: Na análise estática significa a restrição de algumas funções específicas da linguagem. Não foi encontrada nenhuma ferramenta para esta categoria, porém uma das possibilidades é o uso de expressões regulares.

Para fundamentar a avaliação automática dos exercícios também foi estudado o sistema eGrader proposto por [SHAMSI,2012], que aplica avaliações em programas Java dos alunos. A avaliação dos exercícios é feita tanto dinâmica quanto estaticamente. A análise dinâmica neste sistema é realizada através da ferramenta de teste de unidade *JUnit*, que provou-se ser bastante abrangente, completa e efetiva [MASSOL,2003]. A análise estática é feita em duas partes: A similaridade-estrutural, que é realizada através da representação de gráficos e a qualidade, que é

6 http://www.testswarm.com

7 http://code.google.com/p/js-test-driver/

8 http://www.jslint.com

9 http://www.javascriptlint.com/

10 http://code.google.com/p/jsmeter/

avaliada através das seguintes métricas de engenharia de software: Número de Variáveis, Número de Classes e Número de Chamada de Métodos.

3 Avaliação Automática de Exercícios de Programação

Segundo Rahman [RAHMAN,2007a] avaliação automática de exercícios é uma forma de avaliar exercícios de programação dos estudantes, sem a necessidade de fazê-las manualmente. Um dos primeiros trabalhos nesta área de pesquisa foi o desenvolvido por Hollingsworth [HOLLINGSWORTH,1960], que desenvolveu um avaliador automático para programas assembly. A partir deste trabalho surgiram pesquisas para diversas outras linguagens como C,C++, Java e outras [DOUCE,2005].

De acordo com Rahman [RAHMAN,2007a] independente da linguagem geralmente são utilizadas duas abordagens para a avaliação de programas. Uma é a análise estática e outra é a análise dinâmica. A avaliação dinâmica fica por conta dos testes de unidade [KARAVITRTA,2011] e a avaliação estática fica por conta das métricas de engenharia de software [MOREIRA,2009a]. Assim, o trabalho proposto por esta pesquisa realiza avaliação automática tanto dinâmica quanto estática.

3.1 Análise Dinâmica

As análises que envolvem execução do código são denominadas de análise dinâmica [RAHMAN,2007b] e possibilitam um feedback rápido, permitindo que os alunos aprimorem sua programação através de um processo de "tentativa e erro", que reforça o aprendizado [NGHI,2007].

GRADER [FORSYTHE,1965] foi o primeiro sistema a realizar análise dinâmica nos programas dos alunos. Depois dele surgiram diversos outros como TRY [REEK,1989] , BOSS [LUCK,1998] e In-Step [ODEKIRK-HASH,2001].

Esta análise pode ser feita de duas formas: através do teste de caixa preta e teste de caixa branca.

a) Teste de caixa-preta - Também conhecido como teste de entrada/saída, esta é uma estratégia de teste de software que consiste em verificar os momentos em que o programa não coincide com a sua especificação, sem ter conhecimento da estrutura interna do mesmo [MYERS,2004]. Para utilizar esta abordagem para encontrar todos os erros possíveis em um programa, deve-se adotar o critério de entrada exaustiva no programa, fazendo uso de todas as possíveis condições de entrada. Esta é a forma de análise implementada neste trabalho, pois além de atender a todos os requisitos, é uma abordagem mais simples de implementar e testar.

b) Teste de caixa-branca - Também conhecido como teste dirigido pela lógica, esta estratégia de teste de software leva em consideração a estrutura interna do programa. Esta estratégia de teste adota o critério de teste exaustivo dos caminhos, onde todos os possíveis caminhos que um programa possa percorrer são testados. Esta forma não foi implementada neste projeto por duas razões: Testar todos os caminhos possíveis pode tornar o campo de teste muito alto e esta forma de teste, na maioria das vezes, é bem mais lenta que o teste de caixa-preta, não sendo muito utilizada em trabalhos que envolvam teste.

A análise dinâmica gera uma saída, que é comparada com uma estabelecida pelo tutor. Como esta análise requer execução, prevenções precisam ser tomadas para impedir que o sistema avaliador saia do ar caso o programa que está sendo executado contenha um erro crítico de execução ou alguma falha de memória. Estas prevenções são tomadas pela ferramenta de Teste de Software CUnit [BRANDÃO,2005], uma ferramenta que aplica teste de unidade nos programas, um meio de realizar a avaliação dinâmica.

3.1.1 Teste de Unidade

Teste de Unidade é um conceito conhecido na área de desenvolvimento de software há algum tempo. Na década de 70, Kent Beck desenvolveu o conceito para a linguagem SmallTalk e desde então ele tem portado este conceito para diversas outras linguagens de programação [OSHEROVE,2009].

Segundo [HUTCHESON,2003] e [MYERS,2004] teste de unidade é um método para testar pequenos módulos de código como sub-rotinas ou procedimentos mais empregado no meio empresarial e profissional porém, neste trabalho, esta técnica está sendo utilizada para validar a resposta escrita pelos alunos, cuja resolução consiste na codificação de um procedimento contendo a lógica do problema. O método consiste em comparar o procedimento codificado com assertivas que servem como uma especificação funcional que define o procedimento.

Duas causas podem gerar incompatibilidade entre o resultado do procedimento programado e o resultado da assertiva: a) um erro no módulo (variável não declarada ou estouro de memória) ou b) o resultado retornado está incorreto (devido à uma falha na programação do procedimento ou a assertiva foi especificada incorretamente) [MYERS,2004].

Estes testes são aplicados de forma automática através da ferramenta de teste de software para a linguagem de programação C chamada CUnit, que segue o padrão xUnit de desenvolvimento de ferramenta de teste de unidade [BRANDÃO,2005]. O padrão xUnit foi desenvolvido por Kent Beck [BRANDÃO,2005] que vem com um conjunto de modelos de teste para os programadores criarem os seus próprios testes.

Kent Beck quando implementou o padrão xUnit tinha como objetivo apresentar uma alternativa às fracas ferramentas de teste de software da época e que muitas vezes tinham que ser

refeitas e reestruturadas devido a uma modificação no programa-base, o que consumia muito tempo dos programadores.

Em [MESZAROS,2007] é apresentada a estrutura da ferramenta de teste que implementa o padrão xUnit. Neste trabalho apresenta-se uma versão em português, vide Figura 3.1, para melhor compreensão do conceito, porém respeitando a versão original.

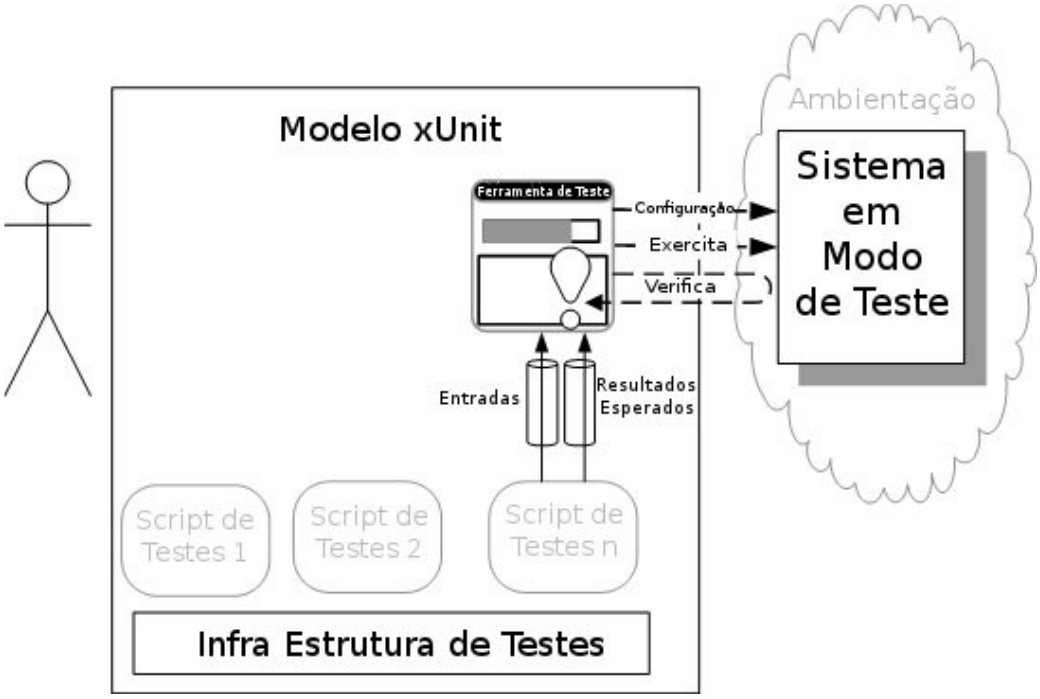

Figura 3.1 : Modelo do padrão xUnit para desenvolvimento de ferramentas de teste

No modelo xUnit da Figura 3.1 os Scripts de teste são escritos pelos programadores, que contêm os dados de entrada e saída esperados pelo programa que será testado. Os testes são rodados pela ferramenta de teste, que é a ferramenta de teste, no SMT (Sistema em Modo de Teste). O ciclo de testes é composto por três tarefas: Configuração, Exercita e Verifica.

A tarefa de Configuração é responsável por carregar todos os objetos e variáveis que são necessários para realizar o teste.

A tarefa de Exercita é responsável por rodar o programa com os dados de entrada determinado pelo script de teste e capturar as saídas do programa testado.

A tarefa de Verifica checa se as saídas do programa testado coincidem com as saídas esperadas, exibindo um relatório dos testes que foram aplicados, quais foram sucedidos, quais falharam, em que linha e o porquê.

3.2 Análise Estática

A análise estática é a abordagem em que a análise do código é feita sem a necessidade de rodá-lo [VILLELA,2008]. Analisadores estáticos realizam a varredura diretamente no código-fonte. Este estudo utiliza o processo de compilação e as métricas de engenharia de software que são exemplos de análise estática do código.

a) Compilação - Segundo [AHO,1995] compilação é um processo que lê um programa escrito em uma linguagem – a *linguagem fonte* – e o traduz para um programa escrito em outra linguagem – a *linguagem destino*. Como importante parte deste processo, o compilador relata ao usuário a presença de erros no programa-fonte. No processo de avaliação desse trabalho utiliza-se o compilador GCC configurado para não exibir os erros para o aluno, pois se o objetivo é descobrir se o aluno sabe programação ou não, exibir esses erros poderia facilitar a resolução do exercício.

GCC, ou GNU CompilerColletion, é uma coleção de compiladores produzida pela Free Software Foundation (FSF) no projeto GNU [STALLMAN,2001]. Em seu projeto inicial suportava apenas a linguagem C, e era denominado de GNU C Compiler. Atualmente suporta diversas outras linguagens como Objective-C,C++ e Fortran. É o compilador padrão para os sistemas operacionais linux e unix.

Além da compilação, as métricas de engenharia de software são utilizadas para realizar análise estática nos programas dos alunos.

3.2.1 Métricas de Engenharia de Software

A comunidade de engenharia de software define o termo métrica de diversas formas, dentre elas: medição de um sistema de software, processo de medição ou medição dos objetos [ABRAN,2010]. Métricas são boas para resumir determinados aspectos do programa e a escolha delas deve ser muito bem planejada, de forma que traga resultados mais precisos e exatos, sendo eles bons ou ruins [LANZA,2006].

As medidas (métricas) de software podem ser divididas, segundo [PRESSMAN,1995] em algumas categorias conforme explanado a seguir.

a) Quanto à Forma - Essa divisão leva em consideração a forma como são medidas as métricas. *As Métricas Diretas* são as que podem ser obtidas sem a dependência de nenhum outro atributo como as métricas de quantidade de linhas de código, custo, esforço e quantidade de defeitos apresentados. E a*s Métricas Indiretas* são as obtidas com base em outras métricas como as de eficiência, qualidade e funcionalidade [COSTA,2011].

b) Quanto à Produtividade - Focam na quantidade do software resolvido, por exemplo, casos de uso resolvidos e quantidade de iterações que foram feitas.

c) Quanto à Qualidade - Referem-se a quanto o software adequa-se às exigências implícitas e explícitas do cliente. Alguns fatores podem ser considerados para analisar a qualidade do software, entre eles:

- *Corretitude*: Mede o quanto o software executa as funções que lhe foram definidas. Uma das medidas mais comuns de corretitude é a quantidade de defeitos por mil linhas de código, onde um defeito é definido como uma falha em relação aos requisitos exigidos.
- *Manutenibilidade*: Mede a facilidade com que um programa pode ser reparado se um erro for encontrado, adaptado se o ambiente modificar-se ou estendido se o cliente desejar incluir requisitos funcionais nele.
- *Integridade*: Mede a capacidade que um sistema tem de suportar invasões (tanto acidentais quanto intencionais) de hackers e/ou vírus.
- *Usabilidade*: Mede o grau de facilidade de uso do sistema pelo usuário, sendo basicamente medida através de quatro características: a aptidão física e/ou mental necessária para utilizar o sistema; o tempo necessário para o indivíduo tornar-se hábil no sistema; o aumento líquido de produtividade (em relação à abordagem anterior) que é medido quando o sistema é usado por alguém considerado hábil e uma avaliação subjetiva (muitas vezes obtida através de entrevista) das atitudes dos usuários em relação ao sistema.

d) Quanto à Técnica - Concentram-se nas características do software, por exemplo, se este segue alguma norma técnica ou o grau de modularidade.

e) Quanto ao Tamanho - São medidas derivadas de atributos de tamanho do software como linhas de código, custo, esforço e quantidade de documentação. Exemplos: *Produtividade:* mil linhas de código-fonte/pessoa-mês. *Qualidade*: defeitos/ mil linhas de código-fonte. *Custo*: dinheiro /linha de código-fonte. *Documentação*: páginas de documentação/mil linhas de código-fonte.

f) Quanto à Função - Métricas orientadas à função são baseadas em medidas indiretas do software, como qualidade e funcionalidade. A técnica de análise de Pontos por Função é um exemplo destas métricas.

g) Quanto às Pessoas – São métricas que processam informações sobre a forma como as pessoas desenvolvem software e sobre a efetividade de ferramentas e métodos segundo os desenvolvedores.

3.2.1.1 Número de Linhas do Programa

Número de linhas do programa é a métrica mais popular e uma das mais antigas que existem [ZUSE,1996]. Ela parte do princípio que quanto mais linhas de código um programa possui, mais complexo ele é. Apesar de existirem controvérsias a respeito do uso desta métrica por não haver uma relação direta entre linhas de código e funcionalidade do programa e ela não levar em consideração a diferença entre as diversas linguagens, esta métrica possui as vantagens de ser fácil de automatizar e de ser uma métrica intuitiva [PRESSMAN,1995].

3.2.1.2 Número de Palavras Reservadas

Segundo [AHO,1995] palavras reservadas são palavras que não podem ser utilizadas como identificadores, como por exemplo, nome de variáveis e classes por terem uso reservado para a gramática da linguagem. Esta métrica tem como objetivo contabilizar a quantidade de palavras reservadas da linguagem usadas no programa, sendo mais um indicador para a avaliação do programa do aluno.

3.2.1.3 Complexidade Ciclomática de McCabe

Complexidade ciclomática tem por objetivo avaliar a manutenibilidade do software [SYMEONIDIS,2006] normalmente utilizando a teoria dos grafos para representar o fluxo de controle do programa.

O fluxo de um programa pode ser representado por um grafo direcionado, que é um grafo onde o fluxo entre os nós é direcionado, como o apresentado na Figura 3.2. Os nós, representados por círculos nomeados com letras de A até G, são blocos sequenciais de código e os arcos representam o fluxo entre estes blocos de código.

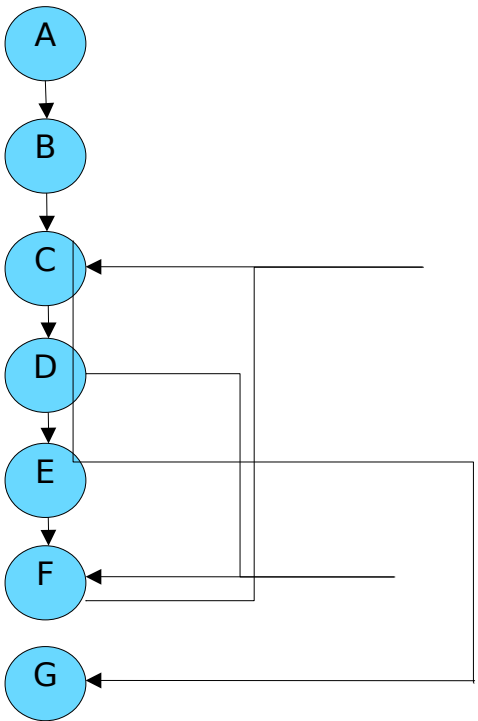

Figura 3.2: Grafo direcionado

O grafo direcionado apresentado na Figura 3.2 corresponde ao código computacional apresentado na Figura 3.3.

```
public static boolean ePrimo(int numero) {
A: boolean primo = true;
B: int contador = 2;
C: while (contador < numero) {
D:    if (numero % contador == 0) {
E:       primo = false;
      }
F:    contador++;
   }
G: return primo;
}
```

Figura 3.3: Código Computacional

A fórmula geral para calcular a complexidade ciclomática é a seguinte:

$$v(G) = e - n + 2 * p \tag{3.1}$$

onde:

v(G): A complexidade ciclomática

e: A quantidade de arcos

n: A quantidade de nós

p: O número de módulos

Existe um teorema descrito em [McCabe,1976] que diz que para programas de única entrada e única saída, é possível montar um grafo fortemente conectado, como o apresentado na Figura 3.4 e simplificar a fórmula geral da complexidade ciclomática para a expressão (3.2) ou (3.3).

$$v(G) = e - n + 2 \quad (3.2)$$

$$v(G) = D + 1 \quad (3.3)$$

onde:

D - Quantidade de estruturas de decisão no programa.

Figura 3.4: Grafo fortemente conectado

A expressão representada em (3.3) é amplamente usada na implementação da métrica correspondente a complexidade ciclomática de McCabe e é a utilizada pela ferramenta RSM, pois admite que um grafo não seja montado para o cálculo da métrica, bastando uma rápida análise do código.

Segundo [McCabe,1976] um limite máximo razoável para a complexidade de um programa é 10, sendo na maior parte dos casos em que programas atingiram complexidade maior que esta precisaram de reformulações e ajustes.

4 Regressão Linear Múltipla

A regressão linear múltipla é um recurso da estatística que tem como objetivo predizer dados

[BARROS,2008] e no trabalho desta monografia será usado como método que beneficia na avaliação dos alunos. É bastante utilizado na área de Mineração de Dados, quando se deseja prever um conjunto de dados futuros, a partir de um montante de dados passados. A previsão é feita calculando a variável dependente na qual se deseja fazer a predição (Y) a partir de um conjunto de variáveis independentes (X_1 , X_2 , X_3 ,..), que são os indicadores que influenciam na variável dependente. A fórmula geral do cálculo da Regressão Linear Múltipla é a seguinte:

$$Y = b_0 + b_1 * X_1 + b_2 * X_2 + \ldots + b_p * X_p \qquad (4.1)$$

onde:

Y: Variável dependente

X_1 , X_2 , ... , X_p *:* Variáveis independentes

$b_0, b_1 b_2$ *,.. ,bp:* Coeficientes técnicos que melhor preveem um dado.

Matricialmente a fórmula (4.1) pode ser escrita como a fórmula (4.2):

$$Y = Xb \qquad (4.2)$$

Sendo:

$$Y = \begin{pmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_n \end{pmatrix}, \qquad X = \begin{pmatrix} 1 & X_{11} & X_{12} & \cdots & X_{1p} \\ 1 & X_{21} & X_{22} & \ldots & X_{2p} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & X_{n1} & X_{n2} & \cdots & X_{np} \end{pmatrix},$$

A matriz de coeficientes $b = \begin{pmatrix} b_0 \\ b_1 \\ \vdots \\ b_p \end{pmatrix}$

Para obter uma boa predição da variável dependente, é imprescindível que os coeficientes técnicos estejam de acordo com a variável dependente. Os coeficientes técnicos são os parâmetros que melhor preveem a nota. O cálculo dos coeficientes técnicos e da constante da regressão linear múltipla é realizado através da metodologia dos mínimos quadrados, utilizando uma tabela dos dados históricos dos exercícios.

4.1 Método dos Mínimos Quadrados

O método dos mínimos quadrados tem como objetivo tratar dados experimentais sujeitos a erros estatísticos [HELENE,2006]. Assim, para o cálculo dos coeficientes técnicos, necessários na

formulação da regressão linear, com o intuito de minimizar os erros de previsão, foi adotada a notação dos mínimos quadrados conforme demonstrada abaixo.

$$S(b) = \sum_{i=1}^{n} (Y_i - b_0 - (b_1 * X_{i1}) - (b_2 * X_{i2}) - \ldots - (b_p * X_{ip}))^2 \quad (4.3)$$

$$Em\ notação\ matricial: (Y - Xb)^2 = (Y - Xb)^T (Y - Xb) \quad (4.4)$$

O valor de b que minimiza o erro de previsão é chamado de coeficiente técnico de b. Segundo [HELENE,2006] existe um único valor dado por $\frac{dS}{db}$ em que $\hat{b}$ = b

$$0 = \frac{dS}{d b_i} = \frac{d}{d b_i} (y^T y - b^T X^T y - y^T Xb + b^T X^T Xb) \vee \square_{b=\hat{b}} = -2 X^T y + 2 X^T X \hat{b} \quad (4.5)$$

Assumindo que $X^T X$ é inversível, o coeficiente técnico $\hat{b}$ é dado por (4.6), segundo [HELENE,2006] .

$$\hat{b} = (X^T X)^{-1} X^T y \quad (4.6)$$

Esta fórmula é utilizada através da ferramenta Xuru [XURU,2008], disponibilizada na internet que realiza cálculo dos mínimos quadrados e da regressão linear múltipla além de apresentar os erros de previsão dos métodos.

5 *Modelo e Implementação*

Neste capítulo é apresentado o modelo proposto, bem como sua implementação para funções ao invés de programas completos, para realizar a avaliação automática dos exercícios. Para realizar a avaliação automática dos exercícios, este trabalho se fundamenta nas técnicas de teste de unidade e da Regressão Linear Múltipla que foram escolhidas de forma que a avaliação feita pelo software avaliador seja a mais próxima possível de uma realizada por um professor humano.

Atualmente os cursos de graduação nas áreas de ciência da computação e engenharias incluem disciplinas de programação básica, sendo o maior desafio do professor desenvolver no aluno a capacidade de raciocinar na mesma sequência do computador. Diante disso, é possível perceber a dificuldade encontrada pelos graduandos e o aparente desânimo dos mesmos logo no início da graduação, já que o tipo de raciocínio exigido não é muito presente em seu cotidiano.

Sendo este um dos motivos de grande evasão logo nos primeiros períodos da graduação o desenvolvimento de aplicações voltadas para o propósito de facilitar a tarefa do professor no ensino de programação básica começou a surgir e neste capítulo será exposto toda a linha de desenvolvimento do sistema, desde a arquitetura do sistema até os aspectos do desenvolvimento do ambiente.

5.1 Interface do sistema

Atualmente temos uma primeira implementação em linguagem Java do Modelo de Avaliação da Resposta do Aluno. Criou-se uma interface, vide figura 5.1.

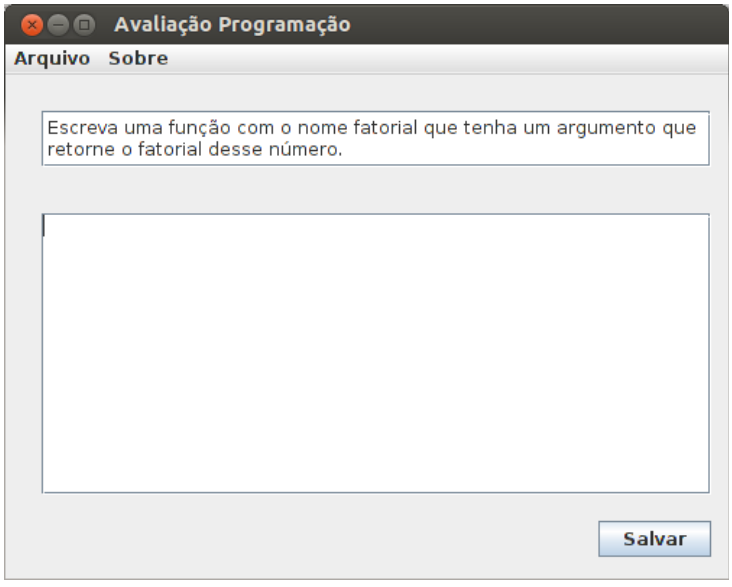

Figura 5.1: Tela principal do exercício

O sistema é definido para trabalhar em dois processos. O primeiro processo consiste em uma fase de treinamento, onde insere-se exercícios dentro do banco de exercícios, escrevendo o código dos testes de unidade, a resposta-modelo e o enunciado do problema. O segundo processo consiste em corrigir o exercício do aluno, conforme modelo descrito na Figura 5.2 e determinar a situação do aluno com relação ao seu aprendizado de programação.

5.2 Fase de Treinamento

Antes de começar a corrigir as soluções dos alunos, exercícios são inseridos, conforme fluxograma apresentado na Figura 5.2.

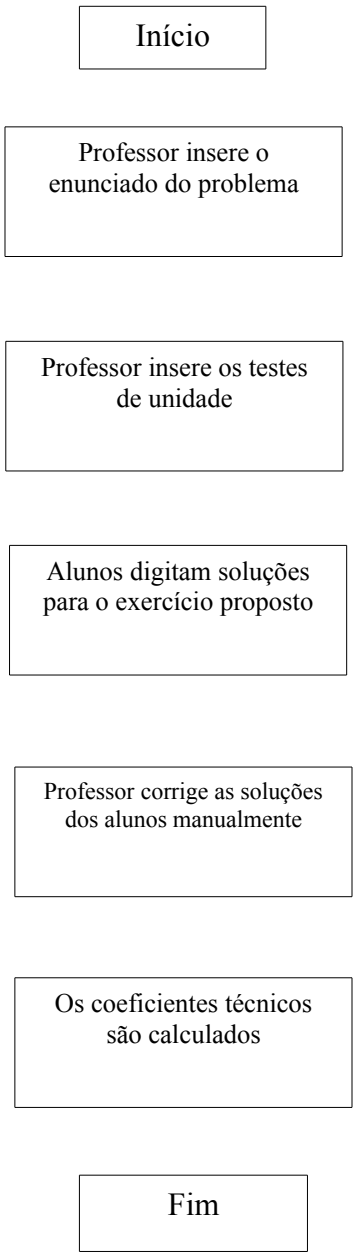

Figura 5.2 : Fluxograma da fase de treinamento

Para a inserção dos enunciados dos problemas e dos testes de unidade, foi necessária a modificação do próprio código-fonte do sistema.

Para a definição dos indicadores, foi realizada a implementação de uma base de dados em MySQL Server, onde as respostas dos alunos fossem armazenadas para que os coeficientes técnicos pudessem ser calculados.

A base de dados implementada foi a definida na Figura 5.3

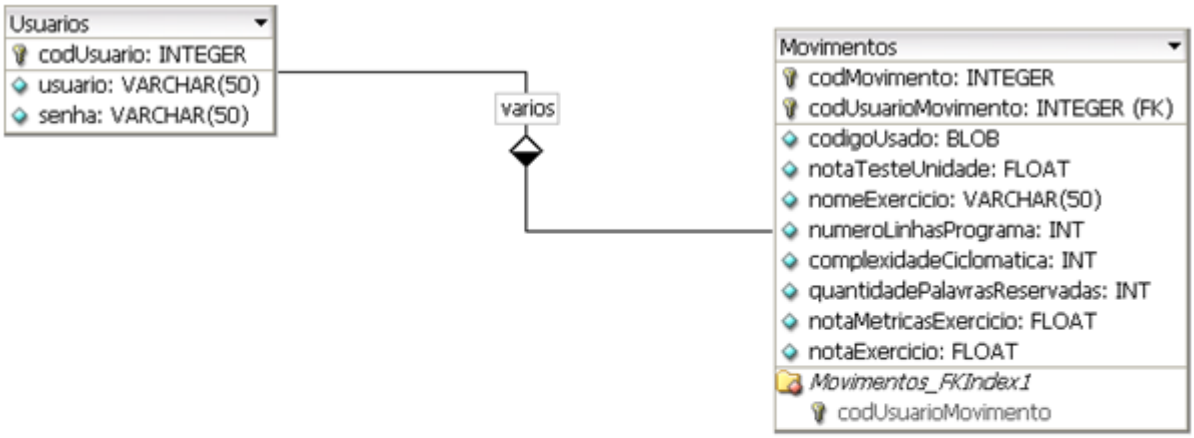

Figura 5.3 : Base de dados HALYEN

Os indicadores foram calculados com base nas respostas dos entrevistados em relação aos seguintes exercícios:

1. Cálculo do fatorial
2. Cálculo da série de Fibonnaci
3. Valor dos sanduíches de uma lanchonete
4. Avaliação do múltiplo de um número
5. Cálculo do produtório
6. A soma de dois números
7. A soma dos múltiplos de uma faixa de números
8. Somatório de uma faixa de números.

Foram selecionados 5 alunos do 9º período do curso de Ciência da Computação da UENF, propondo a eles a resolução de alguns exercícios escolhidos aleatoriamente pelo sistema, cujas

soluções foram corrigidas manualmente, a princípio, para que pudesse montar uma base de treinamento para o sistema, resultando nos dados contidos na Tabela 5.1.

N. L. P.	***C.C.***	***Q. P. R.***	***Nota manual***
9	*2*	*2*	*10*
3	*1*	*1*	*10*
7	*2*	*2*	*10*
5	2	4	10
4	1	1	10
4	1	1	10
6	2	4	5
8	2	2	8
4	1	1	10
6	2	4	10
9	3	3	10
7	2	4	10
3	1	1	10
9	2	4	9,5
7	2	4	10
3	2	1	10
7	2	2	10
3	2	1	10
3	1	1	10
3	2	1	10
9	3	3	10
7	2	2	10
5	2	3	8

Tabela 5.1 : Resultados do primeiro experimento

Onde:

N. L. P.: Número de Linhas do Programa

C. C.: Complexidade ciclomática

Q. P. R.: Quantidade de Palavras Reservadas.

Com os dados apresentados na Tabela 5.1, fazendo uso da metodologia dos mínimos quadrados, através da equação 4.2, sendo N.L.P. X_1 , C.C X_2 e Q.P.R X_3 e as notas manuais Y, foi possível calcular os coeficientes técnicos.

b_0 = b_1 = b_2 = b_3 =

-9.948739638 $6.834714938 \cdot 10^{-2}$ $9.249703942 \cdot 10^{-2}$ $4.069108442 \cdot 10^{-1}$

Resultando na fórmula apresentada em (5.1).

$$y = 6.834714938 \cdot 10^{-2}\ x1 + 9.249703942 \cdot 10^{-2}\ x2 - 4.069108442 \cdot 10^{-1}\ x3 + 9.948739638 \quad (5.1)$$

Posteriormente, para verificar a consistência dos dados utilizados, foram calculados os erros de previsão, que consiste na distância entre a nota manual e a nota dada pela regressão múltipla, apresentados na Tabela 5.2.

Nota manual	***Nota da Regressão***	***Erro***
10	9.935036373	$6.496362714 \cdot 10^{-2}$
10	9.839367281	$1.606327187 \cdot 10^{-1}$
10	9.798342074	$2.016579259 \cdot 10^{-1}$
10	8.847826087	1.152173913
10	9.907714431	$9.228556928 \cdot 10^{-2}$
10	9.907714431	$9.228556928 \cdot 10^{-2}$
5	8.916173236	3.916173236
8	9.866689223	1.866689223
10	9.907714431	$9.228556928 \cdot 10^{-2}$
10	8.916173236	1.083826764
10	9.620622568	$3.793774319 \cdot 10^{-1}$
10	8.984520386	1.015479614
10	9.839367281	$1.606327187 \cdot 10^{-1}$
9,5	9.121214684	$3.787853155 \cdot 10^{-1}$
10	8.984520386	1.015479614
10	9.931864321	$6.813567924 \cdot 10^{-2}$
10	9.798342074	$2.016579259 \cdot 10^{-1}$
10	9.931864321	$6.813567924 \cdot 10^{-2}$
10	9.839367281	$1.606327187 \cdot 10^{-1}$
10	9.931864321	$6.813567924 \cdot$

		10^{-2}
10	9.620622568	$3.793774319 \cdot 10^{-1}$
10	9.798342074	$2.016579259 \cdot 10^{-1}$
8	9.254736931	1.254736931

Tabela 5.2 : Erros de regressão do primeiro experimento

Analisando os resultados do experimento, verificou-se que em algumas respostas o erro de previsão atingiu aproximadamente os valores entre 2 e 4 pontos quando o esperado é um número menor que 1, acredita-se que este problema tenha ocorrido devido aos seguintes fatos: a utilização dos mesmos coeficientes técnicos para a avaliação de todas as questões e os entrevistados não terem respondido a todos os exercícios.

Para solucionar estas falhas, foi realizado o 2º experimento com 5 alunos iniciantes em programação, exigindo resposta em todas as questões, possibilitando a obtenção de coeficientes técnicos diferentes para cada exercício.

Os dados do 2º experimento estão expostos na tabela 5.3.

Nome Exercício	*N. L. P.*	*C.C.*	*Q. P. R.*	*Nota manual*
fatorial	6	3	4	10
fatorial	8	2	2	9
fatorial	6	2	4	10
fatorial	7	2	2	10
fatorial	17	3	5	10
fibonnaci	10	2	4	5
fibonnaci	5	4	4	10
fibonnaci	20	7	12	4
fibonnaci	18	4	5	10
fibonnaci	10	2	2	6
lanchonete	19	6	6	10
lanchonete	21	6	12	10
lanchonete	22	10	3	10
lanchonete	20	6	6	10
lanchonete	10	1	1	10
múltiplo	3	2	1	10
múltiplo	7	2	4	10
múltiplo	6	2	4	10
múltiplo	10	3	2	9
múltiplo	12	3	4	10
produtório	5	2	1	10
produtório	8	2	2	7

produtório	7	2	2	9
produtório	11	3	2	5
produtório	8	3	2	10
soma	3	1	0	10
soma	3	1	1	10
soma	5	1	1	10
soma	5	1	1	10
soma	5	1	1	10
somaMúltiplos	8	3	3	10
somaMúltiplos	8	2	3	9
somaMúltiplos	9	3	3	10
somaMúltiplos	9	3	3	10
somaMúltiplos	11	1	2	4
somatório	7	2	2	9
somatório	6	2	2	10
somatório	7	2	2	10
somatório	6	1	3	10
somatório	10	2	3	5

Tabela 5.3 : Resultados do segundo experimento prático com alunos

Através do cálculo dos coeficientes técnicos de cada questão, foram calculadas as fórmulas de regressão contidas na Tabela 5.4

Nome Exercício	Função de Regressão
Fatorial	$y = -2.962298025 \cdot 10^{-2}\ x1 + 2.064631957 \cdot 10^{-2}\ x2 + 2.432675045 \cdot 10^{-1}\ x3 + 9.184021544$
Fibonnaci	$y = 1.048746699 \cdot 10^{-1}\ x_1 + 3.024698039\ x_2 - 1.946729296\ x_3 + 4.697064808$
Lanchonete	$y = 7.105427358 \cdot 10^{-15}\ x1 - 7.105427358 \cdot 10^{-14}\ x2 - 1.421085472 \cdot 10^{-14}\ x3 + 10$
Múltiplo	$y = -2.307692308 \cdot 10^{-1}\ x1 + 8.076923077 \cdot 10^{-1}\ x2 + 3.846153846 \cdot 10^{-1}\ x3 + 8.461538462$
Produtório	$y = -1.7\ x1 + 2.9\ x2 + 2.25\ x3 + 10.45$
Soma	$y = 1.045950061 \cdot 10^{-14}\ x1 - 1.421085472 \cdot 10^{-14}\ x2 + 0x3 + 10$
SomaMúltiplos	$y = -4.547473509 \cdot 10^{-13}\ x1 + 1\ x2 + 4\ x3 - 5$
Somatório	$y = -0.5\ x1 - 3\ x2 - 3\ x3 + 25$

Tabela 5.4 : Fórmulas de regressão para o segundo experimento

Posteriormente foi calculado novamente os erros de previsão para checar se houve alguma melhora na abordagem, resultando na Tabela 5.5

Nota manual	*Nota da Regressão*	*Erro*
10	10.04129264	$4.129263914 \cdot 10^{-2}$
9	9.47486535	$4.748653501 \cdot 10^{-1}$
10	10.02064632	$2.064631957 \cdot 10^{-2}$
10	9.50448833	$4.955116697 \cdot 10^{-1}$
10	9.958707361	$4.129263914 \cdot 10^{-2}$
5	4.008290402	$9.917095978 \cdot 10^{-1}$
10	9.53331313	$4.666868696 \cdot 10^{-1}$
4	4.60669293	$6.066929304 \cdot 10^{-1}$
10	8.949954543	1.050045457

6	7.901748993	1.901748993
10	10	0
10	10	0
10	10	0
10	10	0
10	10	0
10	9.769230769	$2.307692308 \cdot 10^{-1}$
10	10	0
10	10.23076923	$2.307692308 \cdot 10^{-1}$
9	9.346153846	$3.461538462 \cdot 10^{-1}$
10	9.653846154	$3.461538461 \cdot 10^{-1}$
10	10	0
7	7.15	0.15
9	8.85	0.15
5	4.95	0.05
10	10.05	0.05
10	10	0
10	10	0
10	10	0
10	10	0
10	10	0
10	10	0
9	9	0
10	10	0
10	10	0
4	4	0
9	9.5	0.5
10	10	0
10	9.5	0.5
10	10	0

Tabela 5.5 : Erros de previsão para o segundo experimento

Conforme é possível notar, os erros de previsão reduziram tornando esta abordagem mais adequada para avaliação de exercícios.

5.3 Arquitetura do sistema

A arquitetura conceitual do sistema é apresentada na Figura 5.4. A proposta contém os quatro componentes principais: compilação do Programa, validação do programa, cálculo das métricas de avaliação e comparação com as da resposta modelo e nota do aluno.

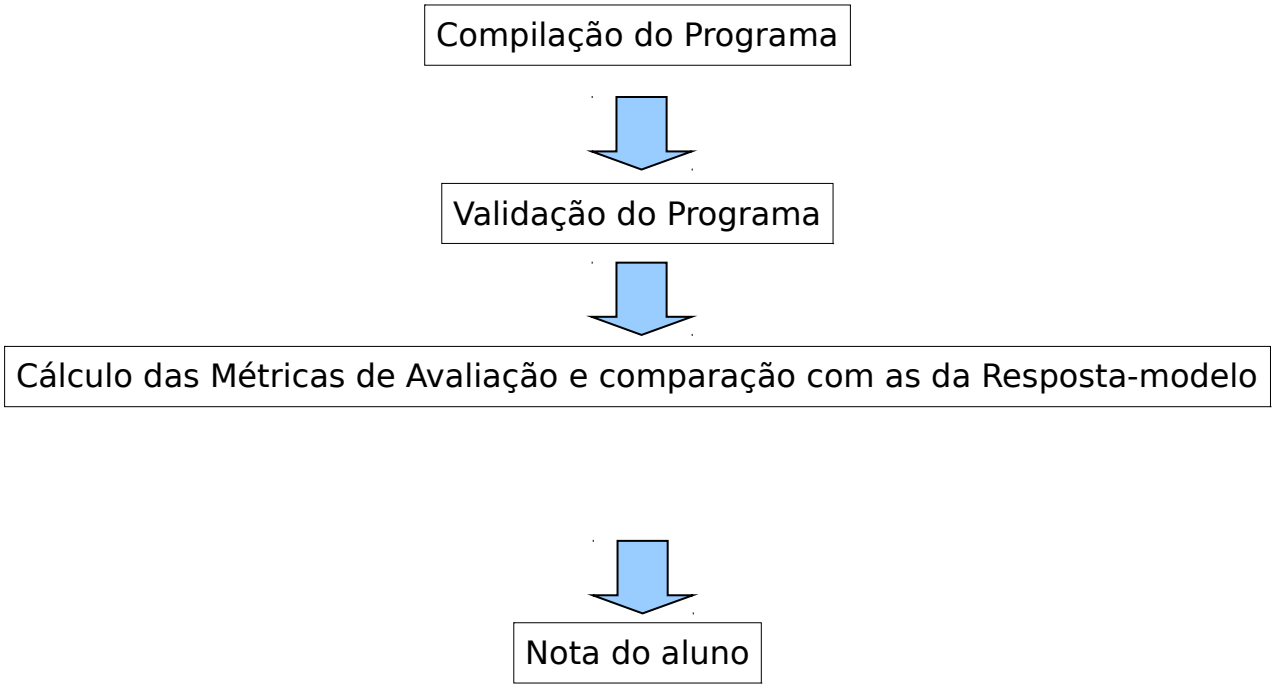

Figura 5.4: Modelo de Avaliação da Resposta do Aluno

5.4 Modelo do sistema

A partir da estrutura apresentada na Figura 5.4 nesta seção apresenta-se um detalhamento de cada um dos componentes; na Figura 5.5 apresenta-se o fluxograma do modelo de avaliação da resposta do aluno.

fluxograma da avaliação de UM exercício de programação

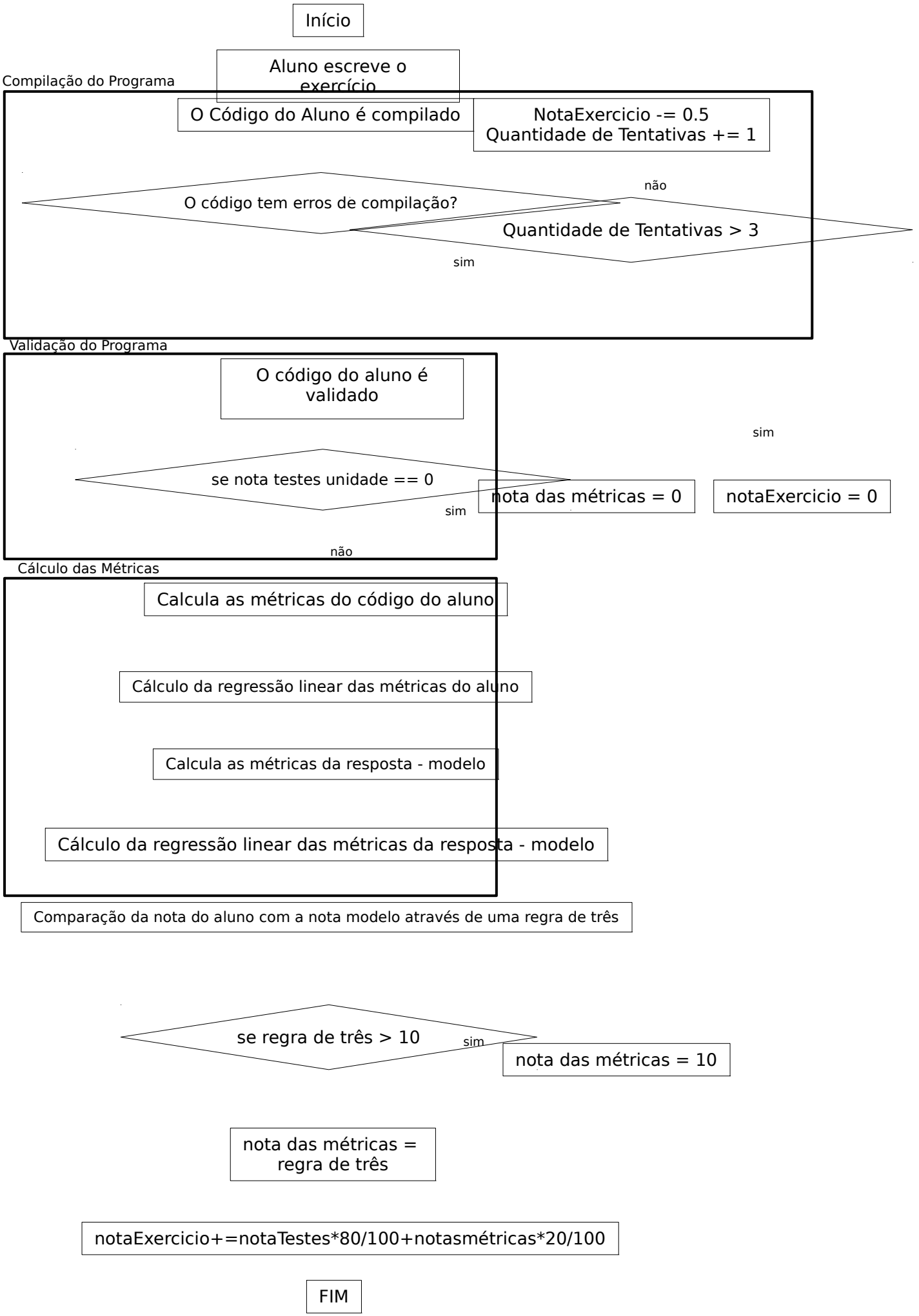

Figura 5.5 :

Fluxograma da avaliação de um exercício de programação

O modelo proposto tem como objetivo apresentar características adicionais, específicas da tecnologia dos testes de unidade e da regressão linear múltipla e descrever o fluxo da solução do aluno entre as fases do modelo de avaliação da Resposta do Aluno.

Passa-se um enunciado de um problema para o aluno, onde a resposta deve ser uma função, escrita em linguagem C, que deve implementar o que o enunciado exige. Essa função, quando submetida ao sistema será feita em várias fases.

A primeira fase é a compilação do programa, feita pelo compilador GCC, que verifica os erros léxicos e sintáticos que possam estar presentes na função. Esta primeira fase é muito importante para a avaliação da solução do aluno, pois a próxima fase da avaliação, Validação do programa, requer que a função seja executável, que não contenha erros léxico ou sintático [AHO,1995]. Pois uma solução que contenha algum erro desse tipo não poderá prosseguir para a próxima fase, retornando essa solução a fase inicial da avaliação, onde deve ser corrigida pelo aluno para que seja submetida novamente ao sistema.

Caso a função do aluno passe na compilação do programa, a avaliação prossegue para a segunda fase: Validação do Programa. Constituída pelo modelo da solução através dos testes de unidade, esta etapa possui informações sobre os requisitos que uma solução de um aluno deve atender, em uma estrutura similar a um caso de uso, porém escrita em código computacional, denominada de caso de teste, que permite uma aplicação por parte do sistema.

O framework de testes CUnit faz a validação junto a solução do aluno, devolvendo ao sistema os requisitos em que ele passou e em quais ele não passou. Os casos de teste, baseados em assertivas, formam uma perspectiva em relação ao problema. Tal visão é comparada com a solução do aluno, resultando então nas falhas e acertos dele em relação aquele problema, resultando na nota para a validação.

Após a Validação do Programa, a avaliação prossegue para a terceira etapa: Cálculo das métricas de avaliação e comparação com as da resposta-modelo.

Segundo [LINO,2007] o cálculo das métricas é uma abordagem adequada para avaliar, caracterizar e analisar um programa fonte. As métricas representam uma abordagem quantitativa para medir a qualidade do software, possibilitando que o aluno passe a ser avaliado em termos de complexidade, onde é medida a distância da qualidade da solução do aluno em relação a solução do professor. O presente estudo utiliza as seguintes métricas: o número de linhas do programa, o número de palavras reservadas e a complexidade ciclomática de McCabe que são calculadas através da ferramenta desenvolvida pela M Squared Technologies chamada Resource Standard Metrics (RSM) [TECHNOLOGIES, 2009], que calcula métricas de engenharia de software para muitas linguagens.

O cálculo das métricas é complementado com a utilização da regressão linear múltipla, que vem sendo utilizada em pesquisas para avaliar automaticamente ensaios em linguagem natural [HEARST,2000] e programas de computador [MOREIRA,2009a]. Porém, vale enfatizar que o sucesso da regressão linear múltipla depende de uma definição criteriosa dos indicadores calculados.

Através do cálculo das métricas e da regressão linear múltipla são previstas as notas da complexidade da solução do aluno e do professor (resposta-modelo), que serão confrontadas. Este estudo adotou o método descrito por [MOREIRA,2009a] para realizar a comparação com a resposta-modelo. Este método consiste em uma regra de três que nivela as duas notas, resultando em uma nota das métricas final.

A avaliação para o exercício finaliza quando é estabelecida uma relação entre a nota da validação e a nota das métricas, sendo que a validação representa 80% e as métricas 20% da nota do exercício. A partir desta nota é possível determinar a situação de aprovação do aluno, a partir do fluxograma descrito na Figura 5.6

FLUXOGRAMA DA SITUAÇÃO DO ALUNO

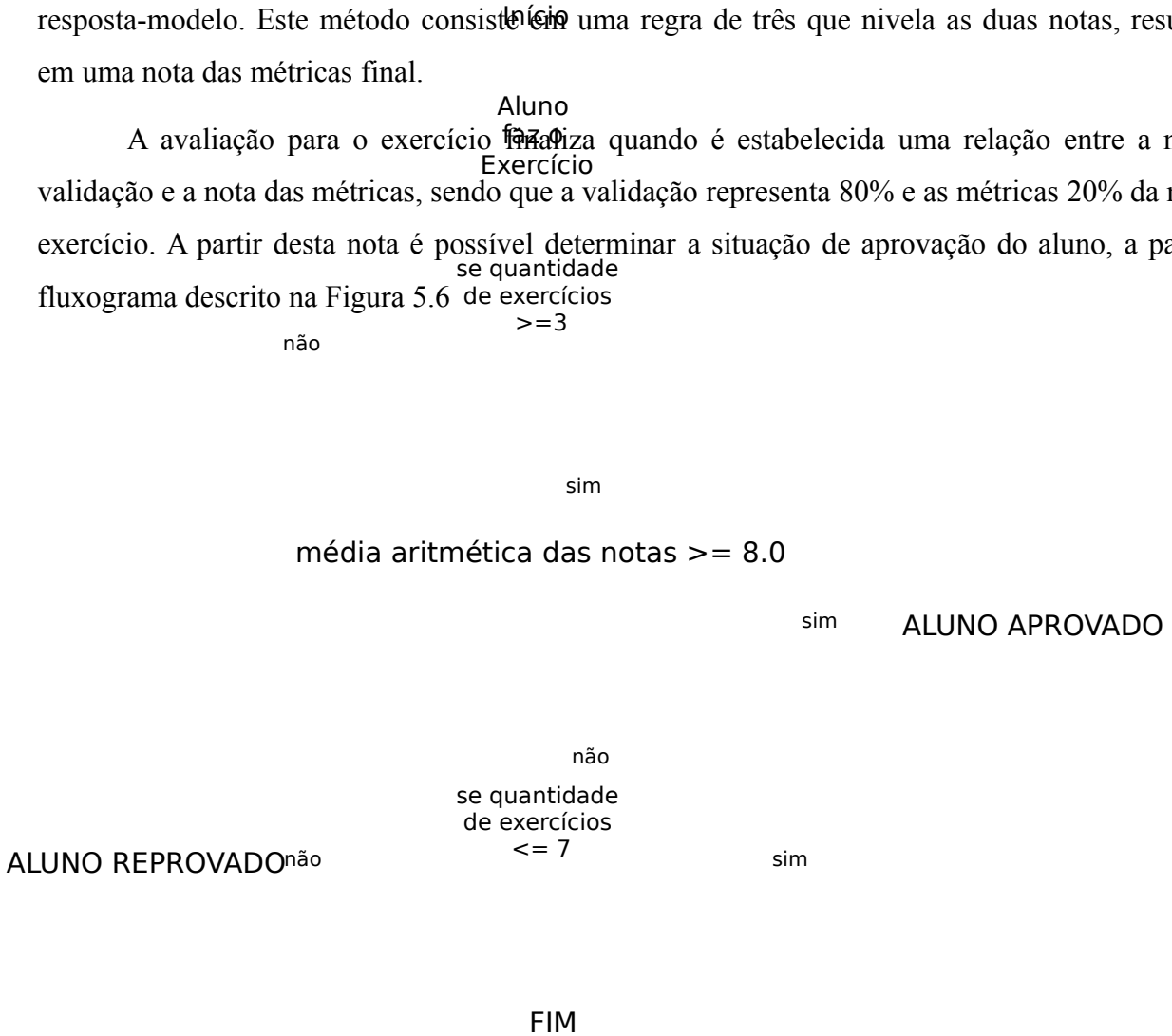

Figura 5.6: Fluxograma da situação do aluno

Conforme critérios estabelecidos empiricamente, a situação do aluno não é deduzida na correção do primeiro exercício, pois para definir a situação do aluno e concluir se este sabe ou não programação, este modelo requer no mínimo a correção de três exercícios.

A partir do terceiro exercício, as notas são coletadas com objetivo de calcular a média das notas para determinar a situação do aluno. Caso o aluno não atinja a média necessária para ser aprovado, outros exercícios serão propostos (até o máximo de 7) para que o aluno consiga a sua situação de aprovado. Desta forma, torna-se possível avaliar de forma mais abrangente o conhecimento do aluno e inferir se o mesmo possui o domínio de programação ou não.

5.5 Implantação do sistema

Este tópico aborda sobre a implantação do sistema, sendo necessário ter instalado o sistema operacional linux Ubuntu 12.04. Após, é necessário seguir os passos a seguir:

1. Após a instalação do sistema operacional, vá para o terminal e digite *sudo apt-get install netbeans* e confirme. O sistema instalará automaticamente o NetBeans e o Java.

2. O GCC não é necessário instalar pois este vem por padrão como compilador no Ubuntu, podendo prosseguir para a instalação do CUnit.

3. Para instalar o CUnit, vá ao Gerenciador de Pacotes Synaptic e procure por CUnit. Instale os pacotes *libcunit1* e *libcunit1-dev*.

4. A instalação da ferramenta RSM não é necessária, pois esta está presente na raíz do sistema encerrando a instalação neste passo.

5. Para a instalação do banco de dados, que o MySQL Server, é necessário digitar o comando *sudo apt-get install mysql-server* e definir como senha do root a definida no código-fonte do sistema.

6. Para finalizar a implantação do sistema, apenas crie uma base de dados com o nome HALYEN e crie as tabelas de acordo com a Figura 5.3

6 *Caso de Estudo*

Para testar a interface proposta, desenvolvemos um estudo de caso formado por um conjunto de exercícios gerados aleatoriamente, onde exploramos:

- Interatividade nos exercícios de programação: pois todo exercício pode ser prontamente executado pelo aluno, corrigindo erros, caso seja detectado por ele no momento da digitação.
- Avaliação inteligente: oferecemos avaliação automática com resposta rápida, para correção por parte do aluno.

Apresentamos os exercícios e pedimos que o aluno escreva uma função como possível solução a um problema oferecido. O aluno digita na caixa de texto a função que ele acredita seja a resposta do exercício e envia para o sistema através do Botão Salvar. Na Figura 6.1 apresentamos o primeiro exemplo.

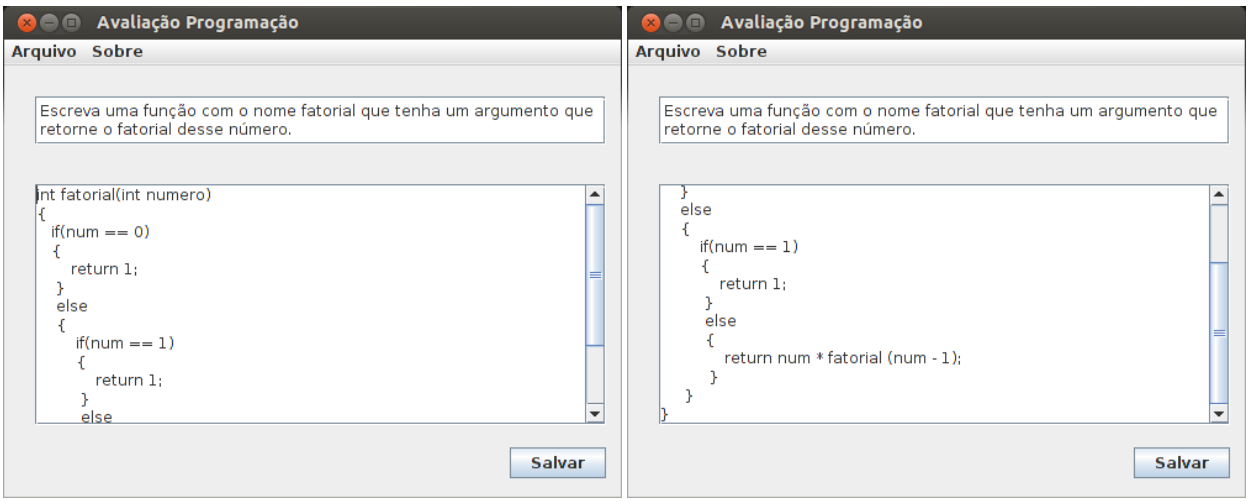

Figura 6.1: Resolução do exercício

Neste caso a função tem erro de compilação, pois a variável **num** não foi declarada. Então o sistema dispara uma mensagem de erro, vide Figura 6.2, penalizando a avaliação do aluno decrementando a nota final para este exercício e retorna para a tela do exercício.

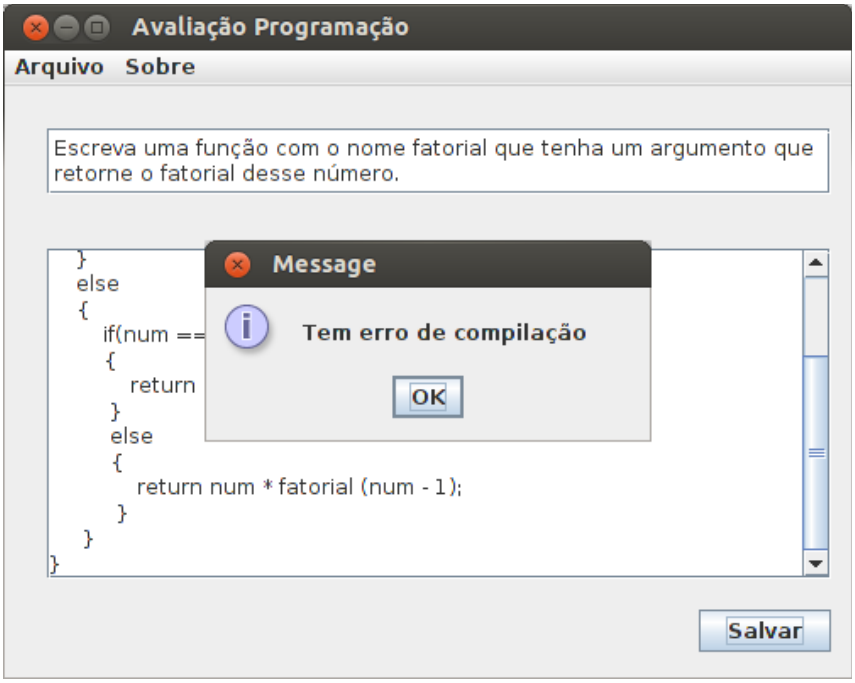

Figura 6.2: Erro de compilação

O aluno reescreve e envia a nova solução abaixo que, como não contêm erros de compilação, terá sua avaliação continuada através da Validação do Programa.

```
int fatorial(int numero)
{
  if(numero == 0)
  {
    return 1;
  }
  else
  {
    if(numero == 1)
    {
      return 1;
    }
    else
    {
      return numero*fatorial(numero-1);
    }
  }
}
```

Para modelar/validar os exercícios, testes de unidade foram escritos. Para este exercício, os testes de unidade definidos foram:

```
CU ASSERT(1 == fatorial(0));
CU ASSERT(1 == fatorial(1));
CU ASSERT(39916800 == fatorial(11));
```

O aluno tem nota 10 para esta fase, pois a implementação proposta atende a todos os requisitos.

Em seguida é calculada as métricas da solução proposta e de uma solução denominada resposta-modelo, que é uma solução que contêm os valores das métricas ideais para o exercício em questão. A resposta-modelo para este exercício é definida a seguir, onde os valores das métricas e suas comparações estão presentes na tabela 6.1.

```
int fatorial( int numero)
{
int fat,i;
if(numero == 0)
return 1;
for(i = 1; i<=numero; i = i + 1)
{
   fat = fat * i;
}
return fat;
}
```

	Número de Linhas do Programa	Complexidade Ciclomática	Quantidade de Palavras Reservadas	Nota Métricas	Notas Métricas comparada com a resposta-Modelo
Resposta do Aluno	17	3	7	9.554	9.850
Resposta Modelo	10	3	4	9.700	10

Tabela 6.1 : Métricas do exercício Fatorial

Depois de termos obtido a nota das métricas, calculamos a nota do exercício, que para este exercício o aluno obteve 9,4. Ele tiraria 9,9 pois ele acertou a lógica do exercício, mas como a sua

solução é menos eficaz que a Resposta-Modelo e ele perdeu pontos por ter enviado a resposta da primeira vez com erro de compilação, sua nota ficou afetada. Aplicando os mesmos princípios, outros exercícios foram aplicados, com objetivo de determinar se o aluno domina programação ou não.

O próximo exercício foi o da série de Fibonnaci. A figura contêm o enunciado do problema, junto com a resposta do aluno.

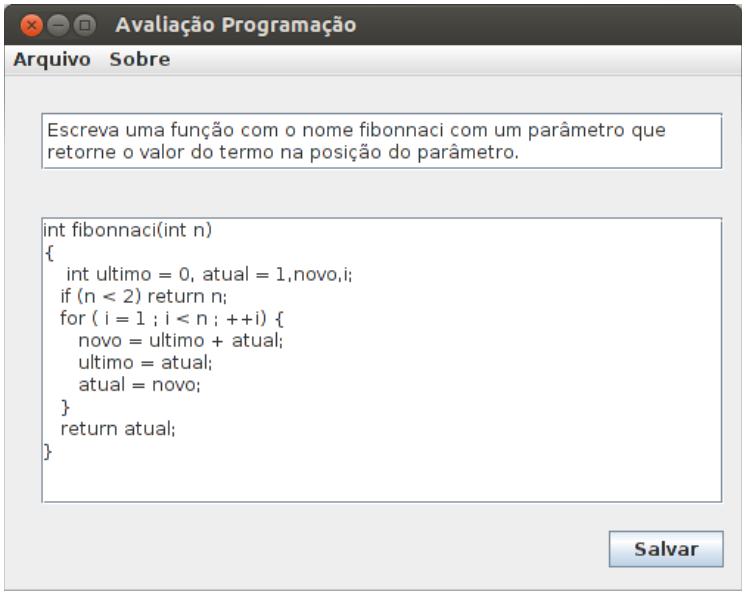

Figura 6.3 : Enunciado do exercício Fibonnaci com solução do aluno

Os seguintes testes foram realizados e cada assertiva valia 2.0 pontos.

```
CU_ASSERT(1 == fibonnaci(1));
CU_ASSERT(1 == fibonnaci(2));
CU_ASSERT(5 == fibonnaci(5));
CU_ASSERT(13 == fibonnaci(7));
CU_ASSERT(34 == fibonnaci(9));
```

Como o aluno neste exercício também atendeu a todos os requisitos, a sua nota para esta fase de avaliação é 10.0.

Em seguida as métricas do programa do aluno são calculadas em conjunto com a resposta-modelo presente abaixo, resultando nas métricas contidas na tabela 6.2 .

Resposta - modelo

```
int fibonnaci(int termo)
{
  int resultado,anterior,i;
  if((termo == 1) || (termo == 2))
  {
    return 1;
  }
  else
  {
    resultado = 1;
    anterior = 1;
    for(i = 3 ; i <= termo; i++)
    {
      resultado = resultado + anterior;
      anterior = resultado - anterior;
    }
    return resultado;
  }
}
```

	Número de Linhas do Programa	Complexidade Ciclomática	Quantidade de Palavras Reservadas	Nota Métricas	Nota Métricas comparada com a resposta-Modelo
Resposta do Aluno	10	3	4	12.004	8.623
Resposta Modelo	18	4	5	13.921	10

Tabela 6.2 : Métricas do Exercício Fibonacci

Como é possível observar a nota das métricas do aluno é de 8.623 , pois o iterador da resposta do aluno faz mais comparações que a resposta modelo.

Com base nas notas da Validação e das métricas, o aluno obteve a nota 9.7, mas para determinar se o aluno conhece ou não programação, é necessário que ele faça no mínimo três exercícios. Então é aplicado o terceiro exercício, que é o da lanchonete, cujo enunciado e resposta do aluno encontra-se na figura 6.4 .

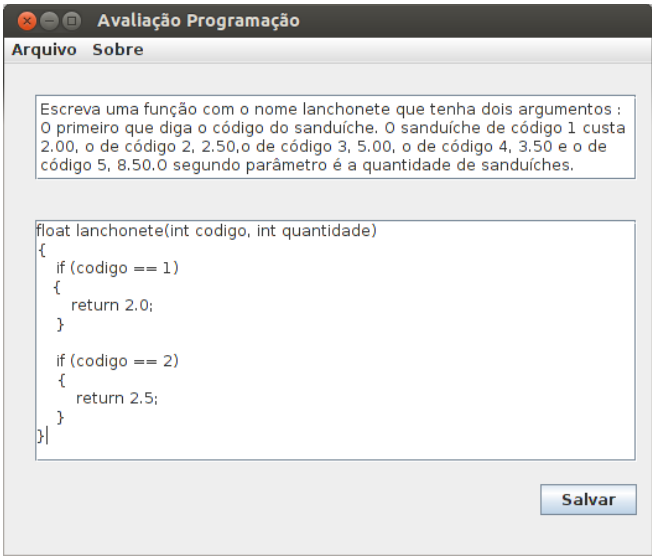

Figura 6.4 : Enunciado e solução do aluno para o exercício da lanchonete.

Estes testes foram realizados e cada assertiva valia 1.0 ponto.

```
CU_ASSERT(2.00 == lanchonete(1,1));
CU_ASSERT(6.00 == lanchonete(1,3));
CU_ASSERT(2.50 == lanchonete(2,1));
CU_ASSERT(10.00 == lanchonete(2,4));
CU_ASSERT(5.00 == lanchonete(3,1));
CU_ASSERT(25.00 == lanchonete(3,5));
CU_ASSERT(3.50 == lanchonete(4,1));
CU_ASSERT(7.00 == lanchonete(4,2));
CU_ASSERT(8.50 == lanchonete(5,1));
CU_ASSERT(93.50 == lanchonete(5,11));
```

Como puderam observar, o aluno não implementou boa parte dos requisitos, obtendo com isso uma nota de validação baixa, do valor de 2.0 .

Passando para a fase das métricas, a seguinte resposta-modelo é usada como base, resultando nos valores das métricas contida na tabela 6.3.

Resposta - modelo

```
float lanchonete(int codigo, int quantidade)
{
  if(codigo == 1)
  {
    return 2 * quantidade;
  }
  if(codigo == 2)
  {
    return 2.50 * quantidade;
  }
  if(codigo == 3)
  {
    return 5 * quantidade;
  }
  if(codigo == 4)
  {
    return 3.50 * quantidade;
  }
  if(codigo == 5)
  {
    return 8.50 * quantidade;
  }
}
```

	Número de Linhas do Programa	Complexidade Ciclomática	Quantidade de Palavras Reservadas	Nota Métricas	Nota Métricas comparada com a resposta-Modelo
Resposta do Aluno	10	3	4	9.999	10
Resposta Modelo	21	6	10	9.999	10

Tabela 6.3 : Métricas do Exercício Lanchonete

Apesar do aluno ter obtido nota máxima para a fase de métricas, o cálculo da nota do exercício derruba a nota do aluno para este exercício para 3.6;

Neste momento o programa já deveria avaliar se o aluno conhece programação, mas a sua média é 7.6, não sendo o suficiente para passar na avaliação, precisando de mais exercícios.

O próximo exercício foi o do múltiplo, onde o aluno deve determinar se o número dividendo é múltiplo do número divisor.

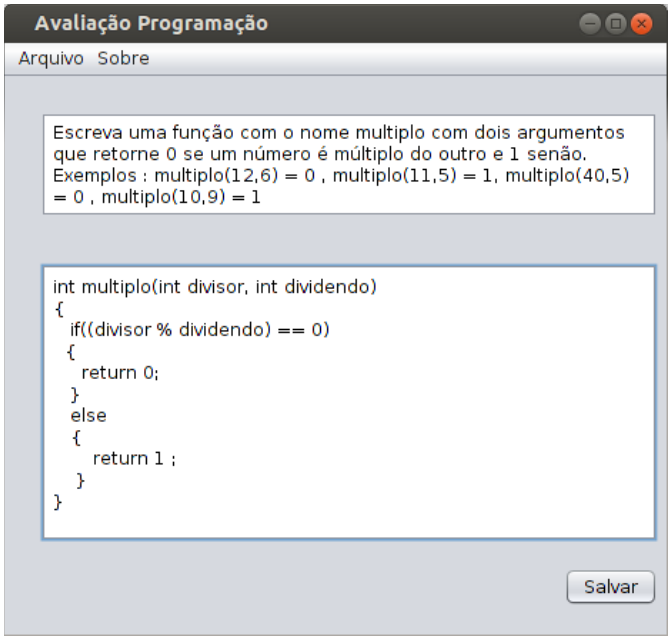

Figura 6.5 : Enunciado e solução do aluno do exercício múltiplo

Os seguintes testes foram realizados e cada assertiva valia 2.0 pontos.

```
CU_ASSERT(0 == multiplo(1,1));
CU_ASSERT(1 == multiplo(1,100));
CU_ASSERT(0 == multiplo(8,4));
CU_ASSERT(1 == multiplo(3,2));
CU_ASSERT(0 == multiplo(80,20));
```

O aluno percebeu a essência do exercício e conseguiu satisfazer todos os requisitos necessários e obteve a nota máxima (10) para a validação dos exercícios.

Em seguida são calculadas as métricas com base na solução do aluno e na resposta-modelo que são apresentadas na tabela 6.4.

Resposta - modelo

```
int multiplo(int divisor, int dividendo)
{
    if((divisor % dividendo) == 0)
    {
       return 0;
    }
    else
    {
```

```
        return 1;
    }
}
```

	Número de Linhas do Programa	Complexidade Ciclomática	Quantidade de Palavras Reservadas	Nota Métricas	Nota Métricas comparada com a resposta-Modelo
Resposta do Aluno	10	2	4	10.153	10
Resposta Modelo	10	2	4	10.153	10

Tabela 6.4 : Métricas do exercício Múltiplo

Como o aluno obteve os mesmos valores de métrica que a resposta-modelo, então ele obteve a nota 10 para a avaliação das métricas.

Em cima das notas de validação e das métricas, o aluno obteve a nota máxima para esse exercício, é feita um novo cálculo da média, que passa a ser de 8.198. Como a média necessária mínima para afirmar que o aluno saiba programação é 8.0, então o aluno passou no teste demonstrando as suas habilidades na programação.

7 Conclusões

O ensino de introdução a programação está regularmente presente nas grades curriculares dos cursos de graduação da computação como uma das principais matérias do curso. Assim, a correção automática de um exercício de programação é um problema que deve ser solucionado através da junção de alguns métodos. A ação do aluno é o centro do processo, onde o trabalho dos testes de unidade e da regressão linear múltipla permite um método de avaliação individualizado.

Como uma solução com mais linhas e mais estruturas de repetição comumente apresenta qualidade inferior de uma solução considerada correta e ideal, as métricas de engenharia de software revelaram-se um meio eficaz para avaliação de um software. As técnicas de regressão linear múltipla e do método dos mínimos quadrados complementam esta abordagem, oferecendo uma nota a partir de um conjunto de métricas.

Através da análise dos resultados da pesquisa pôde-se concluir que os coeficientes técnicos foram calculados com êxito, pois apesar da previsão de erros ter sido elevada em algumas respostas no 1º experimento, no 2º experimento pôde-se perceber que a previsão do erro nas questões foi bem menor, devido aos ajustes realizados para a aplicação do 2º experimento.

O sistema visa fornecer mais um meio de apoio ao professor. O modelo de avaliação dos exercícios dos alunos, utilizando testes de unidade e regressão linear múltipla, oferece um ambiente agradável, interativo e mais confortável para o aluno. Portanto conclui-se que os objetivos definidos no início do trabalho foram cumpridos com êxito.

Poucos trabalhos similares foram encontrados, sendo mais difícil ainda encontrar estudos que já apresentassem resultados de uso na prática.

Durante o desenvolvimento deste trabalho de conclusão de curso, foram adquiridos novos conhecimentos sobre avaliação automática de exercícios, testes de unidade e regressão linear múltipla, que não fazem parte da grade curricular do curso de graduação de Ciência da Computação. Também, foram aplicados conhecimentos, adquiridos nas disciplinas de engenharia de software e programação orientada a objetos.

Portanto esta proposta apresentou-se apropriada, pois visa trazer benefícios para os atuais sistemas de avaliação de exercícios dos alunos, melhorando a prática dos alunos e facilitando a correção por parte do professor. Este produto poderá ser utilizado nos cursos de graduação que possuam computação/programação como base da grade curricular.

8 Trabalhos Futuros

Como continuação do desenvolvimento do sistema, alguns passos devem ser elaborados e/ou melhorados.

- Testar a aplicação em outras turmas da ciência da computação.
- Definir o conhecimento de Programação Orientada a Objetos que deverá ser ensinado.
- Escolher os exercícios de Programação Orientada a Objetos que deverão ser aplicados em cada tópico da ementa, definindo também os testes para eles.

Referências Bibliográficas

ABRAN, A. *Software Metrics and Software Metrology.* IEEE computer society e Wiley, 2010.

AHO, A. V.; SETHI, R.; ULLMAN, J.D. *Compiladores: Princípios, Técnicas e Ferramentas. LTC*, 1995.

ALA-MUTKA, K.: *A survey of automated assessment approaches for programming assignments. Computer Science Education*, 2005.

BARROS, E. A. C. et. al. *Métodos de Estimação em Regressão Linear Múltipla: Aplicação a Dados Clínicos.* Revista Colombiana Estadística, 2008.

BECK, K. *Test-Driven Development:* BY EXAMPLE, *Addison-Wesley*,2002.

BRANDÃO, H. A. et. al. *xUnit: Testes unitários automatizados*. Disponível em: <http:// paginas. fe.up.pt/~aaguiar/es/artigos%20finais/es_final_6.pdf>, Data de acesso : 20/07/2012 , 2005.

CARD D. N. *What makes a software measure suscessful*, *American Programmer*, 1991, Pages 2 - 8.

CLAIR, J. S. *CUnit Progammers Guide.* Disponível em: <http://cunit.sourceforge.net/doc/ introduction.html> Data de acesso: 10/09/2012, 2005

COSTA, A. et.al. *Utilizando métricas para dimensionar um software*. Disponível em: <http:// www.csi.uneb.br/engenharia_de_software/anexos/Artigo-MetricasdeSoftware.pdf>, Data de acesso: 20/07/2012 , 2011

DOUCE, C.; LIVINGSTONE, D.; ORWELL, J. *Automatic test-based assessment of programming: a review*, *acm* , 2005.

FORSYTHE, G. E.; WIRTH, N. *Automatic grading programs*. Commun , 1965.

GUO, P. J. A Scalable Mixed-Level *Approach to Dynamic Analysis of C and C++ Programs, Massachusets Institute of Technology*, 2006.

HARB, M.P.A.; BRITO, S., SILVA, A., FAVERO, E. L.,; LINO, A. *Avaliação automática de consultas sql em ambiente virtual de ensino-aprendizagem. Anais da 2a Conferência Ibérica de Sistemas e Tecnologias de Informação (CISTI)*, 2007.

HEARST, M. A. *The Debate on Automated Essay Grading* . IEEE Inteligent Systems , 2000.

HELENE,O. *Método dos mínimos quadrados com formalismo matricial : guia do usuário*. Editora Livraria da Física, 2006.

HOLLINGSWORTH, J. *Automatic Graders For Programming Classes. Communication acm Vol 3*, 1960

HUTCHESON M. L. *Software Testing Fundamentals: Methods and Metrics. Indianapolis Wisley.* 2003.

KARAVIRTA, V; IHANTOLA, P. *Automatic Assessment of JavaScript Exercises . Aalto University, Finland*, 2011.

JANICIC, M. V.; NIKOLIC, M. ; TOSIC, D. ; KUNCAC, V. *Software Verification and Graph Similatiry for Automated Evaluation of Students´ Assignments* , Swiss National Science Foundation, 2012.

JOY, M.; LUCK, M. *Effective electronic for on-line assessment* . 6th Annual Conference on the Teaching of Computing, 1998.

LANZA, M.; MARINESCU, R. *Object-Oriented Metrics in Practice : Using Software Metrics to Characterize, Evaluate and Improve The Design of Object-Oriented Systems* . Springer, 2006.

LINO, A. D. P. *LABSQL : Laboratório de Ensino de SQL* . UFPA , 2007.

MASSOL , V. ; HUSTED ,T. *JUnit in Action* . Manning , 2003.

MCCABE, T. J. *A Complexity Measure* . IEEE , 1976.

MEDEIROS, C. L. ; DAZZI, R. L. S. *Aprendendo Algoritmos com auxilio da web* . UNIVALI. 2002.

MESZAROS G. *xUnit Test Patterns: Refactoring Test Code* . Addison Wisley, USA, 2007.

[MOREIRA,2009a] **MOREIRA, M. P. ; FAVERO, E. L**. *Um ambiente para ensino de programação*, Universidade Federal do Pará, 2009.

[MOREIRA,2009b**]** **MOREIRA, M. P.; FAVERO E. L.** *Um Ambiente para Ensino de Programação com Feedback Automático de Exercícios*, Anais do XVII Workshop sobre Educação em Informática (WEI) , Bento Gonçalves - RS, 2009, pp. 429-438.

MYERS G. J. *The Art of Software Testing . Indianapolis Wisley*, 2a edition, 2004

NGHI, T. *A Web-Based Programming Environment for Novice Programmers.* Queensland University of Technology, 2007.

ODEKIRK-HASH, E. *Providing automatic feedback to novice programmers.* Master Thesis, University of Utah, 2001.

OSHEROVE, R. *The Art of Unit Testing : with examples in .NET* . Manning , 2009.

PRESSMAN, R. S. Engenharia de Software . MAKRON , 1995.

[RAHMAN,2007a] **RAHMAN, K.A. et.al** *Development of an automated assessment for c programming exercises using pseudocodes comparison technique* , *Citra*, Selangor, 2007.

[RAHMAN,2007b] **RAHMAN,K.A. , NORDIN,M. J.** *A Review on the Static Analysis Approach in the Automated Programming Assessment Systems* , *CSEIJ*, 2007.

REEK, K. A. *The TRY system : or how to avoid testing student programs. SIGCSE Bull , 1989.*

ROBERTSON, L. A. *A Simple Program Design: A Step-by-Step Approach* . Thomson Course Technology, 2004.

SHAMSI, F.A. ; ELNAGAR, A. *An Intelligent Assessment Tool for Students' Java Submissions in Introductory Programming Courses, Journal of Intelligent Learning Systems and Applications*, 2012.

STALLMAN, R. M. *Using and porting the gnu compiler collection* Disponível em: <http://www. skyfree.org/linux/references/gcc-v3.pdf> Data de acesso: 26/06/2012, 2001.

SYMEONIDIS, P. *Automated Assessment of Java Programming Coursework for Computer Science Education* . University of Nottingham , 2006

TAHCHIEV, P .; LEME F. ; MASSOL, V. ; GREGORY G. *JUnit in Action . Manning Publications Co.*, Greenwich, 2003.

TECHNOLOGIES, M. S. *Resource standard metrics c/c++, java and c# metrics*, Disponível em : < http://msquaredtechnologies.com/m2rsm/index.htm>, Data de Acesso : 20/11/2011, 2009

VILLELA, R.T.N.B. *Ferramentas para análise estática de códigos JAVA* ,*UFMG*, 2008

XURU. *Xuru's Website*. Disponível em: <www.xuru.org/Index.asp>. Data de acesso: 05/04/2013, 2008

ZUSE H. *The history of software measurement*. Disponível em: <http://irb.cs.tu-berlin.de/~zuse/metrics/3-hist.html> Data de acesso: 05/07/2012 , 1996