UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO APLICADA

DOUGLAS LUSA KRUG

# MÉTODO E FERRAMENTAL PARA MAPEAMENTO DA EVOLUÇÃO DE PROGRAMADORES DURANTE O DESENVOLVIMENTO DE PROGRAMAS

DISSERTAÇÃO DE MESTRADO

CURITIBA
2018

DOUGLAS LUSA KRUG

# MÉTODO E FERRAMENTAL PARA MAPEAMENTO DA EVOLUÇÃO DE PROGRAMADORES DURANTE O DESENVOLVIMENTO DE PROGRAMAS

Dissertação submetida ao Programa de Pós-Graduação em Computação Aplicada da Universidade Tecnológica Federal do Paraná como requisito parcial para a obtenção do título de Mestre em Computação Aplicada

Orientador: Prof. Dr. Laudelino Cordeiro Bastos
Coorientador: Prof. Dr. Jean Marcelo Simão

CURITIBA

2018

Dados Internacionais de Catalogação na Publicação

---

K94m
2018

Krug, Douglas Lusa
Método e ferramental para mapeamento da evolução de programadores durante o desenvolvimento de programas / Douglas Lusa Krug.-- 2018.
163 f. : il. ; 30 cm

Texto em português com resumo em inglês
Disponível também via World Wide Web
Dissertação (Mestrado) - Universidade Tecnológica Federal do Paraná. Programa de Pós-graduação em Computação Aplicada, Curitiba, 2018
Bibliografia: f. 128-131

1. Software - Desenvolvimento. 2. Gerenciamento de configurações de software. 3. Computadores - Programação. 4. Programação (Computadores). 5. Programação (Computadores) - Estudo e ensino. 6. Computação - Dissertações. I. Bastos, Laudelino Cordeiro. II. Simão, Jean Marcelo. III. Universidade Tecnológica Federal do Paraná - Programa de Pós-graduação em Computação Aplicada. IV. Título.

CDD: Ed. 23 -- 621.39

---

Biblioteca Central da UTFPR, Câmpus Curitiba
Bibliotecário: Adriano Lopes CRB-9/1429

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Diretoria de Pesquisa e Pós-Graduação
UTFPR UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
CAMPUS CURITIBA

# ATA DA DEFESA DE DISSERTAÇÃO DE MESTRADO Nº 66

DISSERTAÇÃO PARA OBTENÇÃO DO TÍTULO DE MESTRE EM COMPUTAÇÃO APLICADA
PROGRAMA DE PÓS-GRADUAÇÃO EM: COMPUTAÇÃO APLICADA
ÁREA DE CONCENTRAÇÃO: ENGENHARIA DE SISTEMAS COMPUTACIONAIS
LINHA DE PESQUISA: ENGENHARIA DE SOFTWARE

No dia 24 de agosto de 2018 às 09h reuniu-se na Sala B204 da Sede Centro a banca examinadora composta pelos pesquisadores indicados a seguir, para examinar a dissertação de mestrado do candidato Douglas Lusa Krug, intitulada: **Método e Ferramental para Mapeamento da Evolução de Programadores durante o desenvolvimento de programas**.

**Orientador: Prof. Dr. Laudelino Cordeiro Bastos**

**Coorientador: Prof. Dr. Jean Marcelo Simão**

Após a apresentação, o candidato foi arguido pelos examinadores que, em seguida à manifestação dos presentes, consideraram o trabalho de pesquisa: ( ) Aprovado. ( ) Aprovado com restrições. Revisor indicado para verificação: ________________________________. ( ) Reprovado.

Observações:

________________________________________________________________________________

________________________________________________________________________________

________________________________________________________________________________

Nada mais havendo a tratar, a sessão foi encerrada às __h__, dela sendo lavrado a presente ata, que segue assinada pela Banca Examinadora e pelo Candidato.

O candidato está ciente que a concessão do referido título está condicionada à: (a) satisfação dos requisitos solicitados pela Banca Examinadora; (b) entrega da dissertação em conformidade com as normas exigidas pela UTFPR; (c) atendimento ao requisito de publicação estabelecido nas normas do Programa; e (d) entrega da documentação necessária para elaboração do Diploma. A Banca Examinadora determina um **prazo máximo de ______ dias**, considerando os prazos máximos definidos no Regulamento Geral do Programa, para o cumprimento dos requisitos (desconsiderar caso reprovado), sob pena de, não o fazendo, ser desvinculado do Programa sem o Título de Mestre.

**Prof. Dr. Gustavo Alberto Giménez Lugo – Presidente – UTFPR** ______________________________

**Profª. Drª. Luciane Telinski Wiedermann Agner – UNICENTRO** ______________________________

**Prof. Dr. João Alberto Fabro – UTFPR** ______________________________

**Prof. Dr. Adolfo Gustavo Serra Seca Neto – UTFPR** ______________________________

**Assinatura do Candidato:** ______________________________

**Reservado à Coordenação**

**DECLARAÇÃO PARA A OBTENÇÃO DO TÍTULO DE MESTRE**

A Coordenação do Programa declara que foram cumpridos todos os requisitos exigidos pelo Programa de Pós-Graduação para a obtenção do título de Mestre.

Curitiba, _____de _______________de 20___.

Carimbo e Assinatura do(a) Coordenador(a) do Programa

**"A Ata de Defesa original está arquivada na Secretaria do PPGCA".**

Dedico este trabalho aos meus pais
Lauro e Marileusa, à minha esposa
Nariel e aos meus filhos Brenda
Luiza e Luiz Ernesto.

## AGRADECIMENTOS

Agradeço em primeiro lugar a Deus, pelas oportunidades colocadas em meu caminho e à minha família pelo apoio durante o tempo necessário para conclusão deste trabalho e pela paciência em relação à minha ausência durante o tempo dedicado a este.

Agradeço também aos orientadores, Prof. Dr. Laudelino Cordeiro Bastos e Prof. Dr. Jean Marcelo Simão, pelo amparo e direcionamento ao longo do percurso deste mestrado.

Da mesma forma, agradeço aos professores que disponibilizaram suas turmas e seu tempo para a realização dos experimentos, Prof. Elio Ribeiro Faria Júnior, Prof. Leonardo Geovany da Silva Zanin e Prof. Jeferson José Baqueta. Assim como o Prof. Alex Matheus Porn pelo auxílio na correção dos artefatos desenvolvidos.

Ao Instituto Federal do Paraná – IFPR, agradeço pela autorização da utilização das minhas horas de pesquisa para realização deste programa mestrado.

Finalmente, agradeço à banca examinadora pela leitura e avaliação desta dissertação de mestrado.

Se eu soubesse que o mundo acabaria amanhã, hoje plantaria uma árvore.

(MARTINHO LUTERO)

## RESUMO

A atividade de programação para computadores, pertinente ao domínio da Engenharia de Software, é um elemento fundamental do processo de desenvolvimento de software. A programação para computadores é inicialmente ensinada em cursos relativos à área de Ciência da Computação em disciplinas de base nomeadas como Fundamentos de Programação, Lógica de Programação e afins. Estas disciplinas são tidas como difíceis e apresentam elevado número de desistência e reprovação. Assim sendo, métodos e ferramentas que auxiliem no processo de ensino-aprendizagem são necessários. Este trabalho relata a proposição de um método e ferramental para mapeamento da evolução de programadores durante o desenvolvimento de programas, podendo ser utilizado como auxílio no processo de ensino-aprendizagem da programação para computadores. O método proposto orienta a realização da coleta e análise dos dados oriundos de eventos gerados durante a programação para computadores e, por sua vez, as ferramentas desenvolvidas permitem realizar a coleta e a consolidação dos dados para a análise do instrutor/docente. Utilizando o método e as ferramentas desenvolvidas foi possível realizar 5 experimentos com alunos de cursos técnicos e de graduação da área de Ciência da Computação e Engenharias de 3 instituições de ensino distintas. Através dos dados coletados nestes experimentos foi possível observar que o número de compilações com falha, a média de erros de compilação e o número total de erros de compilação estão relacionados com a nota atribuída ao exercício resolvido, sendo que quanto maior estes números, menor a nota obtida no exercício. Ainda, referente aos dados coletados nos experimentos, é possível realizar um agrupamento dos erros cometidos durante o desenvolvimento dos programas pelos alunos, fornecendo ao instrutor/docente um panorama do indivíduo e da turma quanto às dificuldades destes. Adicionalmente, também foi observado um impacto positivo quanto ao retorno detalhado aos alunos sobre os dados coletados durante o desenvolvimento de programas, pois na segunda coleta de dados, em comparação com a primeira, é possível observar uma menor incidência de erros de compilação cometidos durante o processo de desenvolvimento do programa.

**Palavras-chave**: Coleta de Dados. Programação de computadores. Processo de ensino-aprendizagem.

## ABSTRACT

The activity of computer programming, pertaining to the Software Engineering domain, is a fundamental element of the software development process. Computer programming is initially taught in courses related to the area of Computer Science in basic subjects named Fundamentals of Programming, Logic of Programming and others. These subjects are considered difficult and present a high index of withdrawal and failure. Therefore, methods and tools that assist in the teaching-learning process are necessary. This work reports the proposition of a method and tools for mapping the evolution of programmers during the development of computer programs, and it can be used as an aid in the computer programming teaching-learning process. The proposed method guides the collection and data analysis from events generated during computer programming and, in turn, the tools developed allow the data collection and consolidation for the instructor/teacher's analysis. Using the method and tools developed it was possible to carry out 5 experiments with students from technical and undergraduate courses in the area of Computer Science and Engineering in 3 different institutions. Based on the data collected in these experiments it was possible to observe that the number of failed compilations, the average of compilation errors and the total number of compilation errors are related to the grade assigned to the solved exercise, the higher these numbers are, the lower the obtained grade is. Also, regarding the data collected in the experiments, it is possible to group the errors performed by students during the development of the programs, giving the instructor/teacher an individual and group overview regarding their difficulties. In addition, a positive impact was also observed regarding the detailed feedback to the students about the data collected during the program development, because in the second data collection, in comparison with the first, it is possible to observe a lower incidence of compilation errors performed during the program development process.



**Keywords:** Data collection. Computer programming. Teaching-learning process.

**LISTA DE FIGURAS**

FIGURA 1 – ETAPAS DE COLETA E ANÁLISE....................................................47
FIGURA 2 – XML – CAPTURA EVENTOS.............................................................56
FIGURA 3 – XML – EVENTOS DE PROJETO.......................................................56
FIGURA 4 – TELA – AÇÃO DE SALVAR................................................................57
FIGURA 5 – XML – CÓDIGO SALVO.....................................................................57
FIGURA 6 – TELA – COMPILAÇÃO DO PROGRAMA..........................................58
FIGURA 7 – XML – EVENTO DE COMPILAÇÃO 1...............................................58
FIGURA 8 – XML – EVENTO DE COMPILAÇÃO 2...............................................58
FIGURA 9 – XML – EVENTO DE COMPILAÇÃO 3...............................................59
FIGURA 10 – XML – EVENTO DE COMPILAÇÃO 4.............................................59
FIGURA 11 – XML – EVENTO DE COLAR TEXTO................................................60
FIGURA 12 – XML – EVENTO DE ENCERRAMENTO..........................................60
FIGURA 13 – LISTA DE COMPILAÇÕES/EXECUÇÕES.......................................62
FIGURA 14 – RESUMO DE COMPILAÇÕES/EXECUÇÕES.................................63
FIGURA 15 – ERROS EM COMPILAÇÕES...........................................................63
FIGURA 16 – RESUMO DE ERROS EM COMPILAÇÕES – CLASSIFICAÇÃO.......63
FIGURA 17 – RESUMO DE ERROS EM COMPILAÇÕES......................................64
FIGURA 18 – ERROS DE LÓGICA – ARTEFATO ENTREGUE..............................64
FIGURA 19 – PROJETOS UTILIZADOS.................................................................65
FIGURA 20 – ARQUIVOS SALVOS........................................................................65
FIGURA 21 – RESUMO DE CÓDIGOS COLADOS.................................................65
FIGURA 22 – ARTEFATO FINAL.............................................................................65
FIGURA 23 – RESUMO DAS COMPILAÇÕES DA TURMA/EXERCÍCIO.................66
FIGURA 24 – RESUMO DA CLASSIFICAÇÃO DE ERROS DE COMPILAÇÃO DA TURMA/EXERCÍCIO.................................................................................................67
FIGURA 25 – RESUMO DAS NOTAS DA TURMA/EXERCÍCIO..............................67
FIGURA 26 – RESUMO DAS NOTAS DA TURMA/EXERCÍCIO..............................67
FIGURA 27 – DIAGRAMA ENTIDADE RELACIONAMENTO...................................68

**LISTA DE GRÁFICOS**


GRÁFICO 1 – ERROS DE COMPILAÇÃO – EXPERIMENTO 1 – PRIMEIRA COLETA........77

GRÁFICO 2 – ERROS DE COMPILAÇÃO – EXPERIMENTO 1 – SEGUNDA COLETA........80

GRÁFICO 3 – ERROS DE COMPILAÇÃO – EXPERIMENTO 2 – PRIMEIRA COLETA........83

GRÁFICO 4 – ERROS DE COMPILAÇÃO – EXPERIMENTO 2 – SEGUNDA COLETA........86

GRÁFICO 5 – ERROS DE COMPILAÇÃO – EXPERIMENTO 3 – PRIMEIRA COLETA........89

GRÁFICO 6 – ERROS DE COMPILAÇÃO – EXPERIMENTO 3 – SEGUNDA COLETA........92

GRÁFICO 7 – ERROS DE COMPILAÇÃO – EXPERIMENTO 4 – PRIMEIRA COLETA........95

GRÁFICO 8 – ERROS DE COMPILAÇÃO – EXPERIMENTO 4 – SEGUNDA COLETA........98

GRÁFICO 9 – ERROS DE COMPILAÇÃO – EXPERIMENTO 5 – PRIMEIRA COLETA........101

GRÁFICO 10 – ERROS DE COMPILAÇÃO – EXPERIMENTO 5 – SEGUNDA COLETA........104

GRÁFICO 11 – CORRELAÇÃO ENTRE COMPILAÇÕES COM SUCESSO E DESEMPENHO ACADÊMICO DO EXERCÍCIO – EXPERIMENTO 1 – PRIMEIRA COLETA........106

GRÁFICO 12 – CORRELAÇÃO ENTRE COMPILAÇÕES COM SUCESSO E DESEMPENHO ACADÊMICO DO EXERCÍCIO – EXPERIMENTO 1 – SEGUNDA COLETA........106

GRÁFICO 13 – CORRELAÇÃO ENTRE COMPILAÇÕES COM FALHA E DESEMPENHO ACADÊMICO DO EXERCÍCIO – EXPERIMENTO 1 – PRIMEIRA COLETA........107

GRÁFICO 14 – CORRELAÇÃO ENTRE COMPILAÇÕES COM FALHA E DESEMPENHO ACADÊMICO DO EXERCÍCIO – EXPERIMENTO 1 – SEGUNDA COLETA........107

GRÁFICO 15 – CORRELAÇÃO ENTRE A MÉDIA DE ERROS DE COMPILAÇÕES E DESEMPENHO ACADÊMICO DO EXERCÍCIO – EXPERIMENTO 1 – PRIMEIRA COLETA.....108
GRÁFICO 16 – CORRELAÇÃO ENTRE A MÉDIA DE ERROS DE COMPILAÇÕES E DESEMPENHO ACADÊMICO DO EXERCÍCIO – EXPERIMENTO 1 – SEGUNDA COLETA.....108
GRÁFICO 17 – CORRELAÇÃO ENTRE O TOTAL DE ERROS DE COMPILAÇÕES E DESEMPENHO ACADÊMICO DO EXERCÍCIO – EXPERIMENTO 1 – PRIMEIRA COLETA.....109
GRÁFICO 18 – CORRELAÇÃO ENTRE O TOTAL DE ERROS DE COMPILAÇÕES E DESEMPENHO ACADÊMICO DO EXERCÍCIO – EXPERIMENTO 1 – SEGUNDA COLETA.....109
GRÁFICO 19 – CORRELAÇÃO ENTRE COMPILAÇÕES COM SUCESSO E DESEMPENHO ACADÊMICO DO EXERCÍCIO – EXPERIMENTO 2 – PRIMEIRA COLETA.....110
GRÁFICO 20 – CORRELAÇÃO ENTRE COMPILAÇÕES COM SUCESSO E DESEMPENHO ACADÊMICO DO EXERCÍCIO – EXPERIMENTO 2 – SEGUNDA COLETA.....111
GRÁFICO 21 – CORRELAÇÃO ENTRE COMPILAÇÕES COM FALHA E DESEMPENHO ACADÊMICO DO EXERCÍCIO – EXPERIMENTO 2 – PRIMEIRA COLETA.....111
GRÁFICO 22 – CORRELAÇÃO ENTRE COMPILAÇÕES COM FALHA E DESEMPENHO ACADÊMICO DO EXERCÍCIO – EXPERIMENTO 2 – SEGUNDA COLETA.....112
GRÁFICO 23 – CORRELAÇÃO ENTRE A MÉDIA DE ERROS DE COMPILAÇÕES E DESEMPENHO ACADÊMICO DO EXERCÍCIO – EXPERIMENTO 2 – PRIMEIRA COLETA.....112
GRÁFICO 24 – CORRELAÇÃO ENTRE A MÉDIA DE ERROS DE COMPILAÇÕES E DESEMPENHO ACADÊMICO DO EXERCÍCIO – EXPERIMENTO 2 – SEGUNDA COLETA.....113
GRÁFICO 25 – CORRELAÇÃO ENTRE O TOTAL DE ERROS DE COMPILAÇÕES E DESEMPENHO ACADÊMICO DO EXERCÍCIO – EXPERIMENTO 2 – PRIMEIRA COLETA.....113

GRÁFICO 26 – CORRELAÇÃO ENTRE O TOTAL DE ERROS DE COMPILAÇÕES E DESEMPENHO ACADÊMICO DO EXERCÍCIO – EXPERIMENTO 2 – SEGUNDA COLETA....................................................................................................................114
GRÁFICO 27 – CORRELAÇÃO ENTRE COMPILAÇÕES COM SUCESSO E DESEMPENHO ACADÊMICO DO EXERCÍCIO – EXPERIMENTO 4 – PRIMEIRA COLETA....................................................................................................................116
GRÁFICO 28 – CORRELAÇÃO ENTRE COMPILAÇÕES COM SUCESSO E DESEMPENHO ACADÊMICO DO EXERCÍCIO – EXPERIMENTO 4 – SEGUNDA COLETA....................................................................................................................116
GRÁFICO 29 – CORRELAÇÃO ENTRE COMPILAÇÕES COM FALHA E DESEMPENHO ACADÊMICO DO EXERCÍCIO – EXPERIMENTO 4 – PRIMEIRA COLETA....................................................................................................................117
GRÁFICO 30 – CORRELAÇÃO ENTRE COMPILAÇÕES COM FALHA E DESEMPENHO ACADÊMICO DO EXERCÍCIO – EXPERIMENTO 4 – SEGUNDA COLETA....................................................................................................................117
GRÁFICO 31 – CORRELAÇÃO ENTRE A MÉDIA DE ERROS DE COMPILAÇÕES E DESEMPENHO ACADÊMICO DO EXERCÍCIO – EXPERIMENTO 4 – PRIMEIRA COLETA....................................................................................................................118
GRÁFICO 32 – CORRELAÇÃO ENTRE A MÉDIA DE ERROS DE COMPILAÇÕES E DESEMPENHO ACADÊMICO DO EXERCÍCIO – EXPERIMENTO 4 – SEGUNDA COLETA....................................................................................................................118
GRÁFICO 33 – CORRELAÇÃO ENTRE O TOTAL DE ERROS DE COMPILAÇÕES E DESEMPENHO ACADÊMICO DO EXERCÍCIO – EXPERIMENTO 4 – PRIMEIRA COLETA....................................................................................................................119
GRÁFICO 34 – CORRELAÇÃO ENTRE O TOTAL DE ERROS DE COMPILAÇÕES E DESEMPENHO ACADÊMICO DO EXERCÍCIO – EXPERIMENTO 4 – SEGUNDA COLETA....................................................................................................................119
GRÁFICO 35 – VARIAÇÃO ENTRE COLETAS......................................................122

## LISTA DE QUADROS

QUADRO 1 – PRINCIPAIS CARACTERÍSTICAS DOS TRABALHOS CONSIDERADOS....18
QUADRO 2 – CLASSIFICAÇÃO DOS TRABALHOS DE ACORDO COM O OBJETIVO DA PESQUISA....27
QUADRO 3 – CONSOLIDAÇÃO DOS TRABALHOS CONSIDERADOS....37
QUADRO 4 – CONSOLIDAÇÃO SOBRE OS DADOS COLETADOS....41
QUADRO 5 – CONSOLIDAÇÃO SOBRE MAPEAMENTO DE ERROS....42
QUADRO 6 – CONSOLIDAÇÃO SOBRE TESTE DE CORRELAÇÃO....43
QUADRO 7 – MAPEAMENTO DE EVENTOS E DADOS COLETADOS....48
QUADRO 8 – CRITÉRIOS E PONTUAÇÃO DOS EXERCÍCIOS....49
QUADRO 9 – CLASSIFICAÇÃO DE ERROS DE LÓGICA....50
QUADRO 10 – CLASSIFICAÇÃO DE ERROS DE COMPILAÇÃO....51
QUADRO 11 – INTERPRETAÇÃO DO COEFICIENTE DE CORRELAÇÃO....54
QUADRO 12 – RESUMO DOS EXPERIMENTOS REALIZADOS....71
QUADRO 13 – COEFICIENTE DE CORRELAÇÃO – EXPERIMENTO 1....105
QUADRO 14 – COEFICIENTE DE CORRELAÇÃO – EXPERIMENTO 2....110
QUADRO 15 – COEFICIENTE DE CORRELAÇÃO – EXPERIMENTO 3....114
QUADRO 16 – COEFICIENTE DE CORRELAÇÃO – EXPERIMENTO 4....115
QUADRO 17 – COEFICIENTE DE CORRELAÇÃO – EXPERIMENTO 5....120
QUADRO 18 – VARIAÇÃO ENTRE COLETAS – COMPILAÇÕES E ERROS....121

## LISTA DE TABELAS

TABELA 1 – RESUMO DE COMPILAÇÕES – EXPERIMENTO 1 – PRIMEIRA COLETA....76
TABELA 2 – CLASSIFICAÇÃO DOS ERROS DE COMPILAÇÃO – EXPERIMENTO 1 – PRIMEIRA COLETA....77
TABELA 3 – RESUMO DE ERROS DE LÓGICA – EXPERIMENTO 1 – PRIMEIRA COLETA....78
TABELA 4 – RESUMO DE NOTAS – EXPERIMENTO 1 – PRIMEIRA COLETA.......78
TABELA 5 – RESUMO DE COMPILAÇÕES – EXPERIMENTO 1 – SEGUNDA COLETA....79
TABELA 6 – CLASSIFICAÇÃO DOS ERROS DE COMPILAÇÃO – EXPERIMENTO 1 – SEGUNDA COLETA....80
TABELA 7 – RESUMO DE ERROS DE LÓGICA – EXPERIMENTO 1 – SEGUNDA COLETA....81
TABELA 8 – RESUMO DE NOTAS – EXPERIMENTO 1 – SEGUNDA COLETA……81
TABELA 9 – RESUMO DE COMPILAÇÕES – EXPERIMENTO 2 – PRIMEIRA COLETA....82
TABELA 10 – CLASSIFICAÇÃO DOS ERROS DE COMPILAÇÃO – EXPERIMENTO 2 – PRIMEIRA COLETA....82
TABELA 11 – RESUMO DE ERROS DE LÓGICA – EXPERIMENTO 2 – PRIMEIRA COLETA....83
TABELA 12 – RESUMO DE NOTAS – EXPERIMENTO 2 – PRIMEIRA COLETA.....84
TABELA 13 – RESUMO DE COMPILAÇÕES – EXPERIMENTO 2 – SEGUNDA COLETA....85
TABELA 14 – CLASSIFICAÇÃO DOS ERROS DE COMPILAÇÃO – EXPERIMENTO 2 – SEGUNDA COLETA....85
TABELA 15 – RESUMO DE ERROS DE LÓGICA – EXPERIMENTO 2 – SEGUNDA COLETA....86
TABELA 16 – RESUMO DE NOTAS – EXPERIMENTO 2 – SEGUNDA COLETA…87
TABELA 17 – RESUMO DE COMPILAÇÕES – EXPERIMENTO 3 – PRIMEIRA COLETA....88

TABELA 18 – CLASSIFICAÇÃO DOS ERROS DE COMPILAÇÃO – EXPERIMENTO 3 – PRIMEIRA COLETA....89
TABELA 19 – RESUMO DE ERROS DE LÓGICA – EXPERIMENTO 3 – PRIMEIRA COLETA....90
TABELA 20 – RESUMO DE NOTAS – EXPERIMENTO 3 – PRIMEIRA COLETA.....90
TABELA 21 – RESUMO DE COMPILAÇÕES – EXPERIMENTO 3 – SEGUNDA COLETA....91
TABELA 22 – CLASSIFICAÇÃO DOS ERROS DE COMPILAÇÃO – EXPERIMENTO 3 – SEGUNDA COLETA....91
TABELA 23 – RESUMO DE ERROS DE LÓGICA – EXPERIMENTO 3 – SEGUNDA COLETA....92
TABELA 24 – RESUMO DE NOTAS – EXPERIMENTO 3 – SEGUNDA COLETA....93
TABELA 25 – RESUMO DE COMPILAÇÕES – EXPERIMENTO 4 – PRIMEIRA COLETA....94
TABELA 26 – CLASSIFICAÇÃO DOS ERROS DE COMPILAÇÃO – EXPERIMENTO 4 – PRIMEIRA COLETA....95
TABELA 27 – RESUMO DE ERROS DE LÓGICA – EXPERIMENTO 4 – PRIMEIRA COLETA....96
TABELA 28 – RESUMO DE NOTAS – EXPERIMENTO 4 – PRIMEIRA COLETA.....96
TABELA 29 – RESUMO DE COMPILAÇÕES – EXPERIMENTO 4 – SEGUNDA COLETA....97
TABELA 30 – CLASSIFICAÇÃO DOS ERROS DE COMPILAÇÃO – EXPERIMENTO 4 – SEGUNDA COLETA....97
TABELA 31 – RESUMO DE ERROS DE LÓGICA – EXPERIMENTO 4 – SEGUNDA COLETA....98
TABELA 32 – RESUMO DE NOTAS – EXPERIMENTO 4 – SEGUNDA COLETA....99
TABELA 33 – RESUMO DE COMPILAÇÕES – EXPERIMENTO 5 – PRIMEIRA COLETA....100
TABELA 34 – CLASSIFICAÇÃO DOS ERROS DE COMPILAÇÃO – EXPERIMENTO 5 – PRIMEIRA COLETA....101
TABELA 35 – RESUMO DE ERROS DE LÓGICA – EXPERIMENTO 5 – PRIMEIRA COLETA....102
TABELA 36 – RESUMO DE NOTAS – EXPERIMENTO 5 – PRIMEIRA COLETA...102

TABELA 37 – RESUMO DE COMPILAÇÕES – EXPERIMENTO 5 – SEGUNDA COLETA..........103
TABELA 38 – CLASSIFICAÇÃO DOS ERROS DE COMPILAÇÃO – EXPERIMENTO 5 – SEGUNDA COLETA..........103
TABELA 39 – RESUMO DE ERROS DE LÓGICA – EXPERIMENTO 5 – SEGUNDA COLETA..........104
TABELA 40 – RESUMO DE NOTAS – EXPERIMENTO 5 – SEGUNDA COLETA. .105

## LISTA DE ABREVIATURAS E SIGLAS

| | |
|---|---|
| IDE | - *Integrated Development Environment* |
| QE | - Quociente de Erro |
| TI | - Tecnologia da Informação |
| XML | - *eXtensible Markup Language* |

SUMÁRIO

**1 INTRODUÇÃO....13**
1.1 JUSTIFICATIVA....14
1.2 OBJETO DE ESTUDOS....16
1.3 MOTIVAÇÃO....20
1.4 OBJETIVO GERAL E OBJETIVOS ESPECÍFICOS....22
1.4.1 Objetivos Específicos....22
1.5 ESTRUTURA DO TRABALHO....23
**2 FUNDAMENTAÇÃO TEÓRICA....24**
2.1 DIFICULDADES NO ENSINO DE PROGRAMAÇÃO....24
2.2 ERROS DE COMPILAÇÃO....25
2.3 COLETA DE DADOS DURANTE A CONSTRUÇÃO DE PROGRAMAS....26
2.3.1 Coleta de dados....28
2.3.2 Trabalhos correlatos....29
2.3.3 Resumo dos trabalhos considerados....37
2.3.4 Dados utilizados....40
2.3.5 Mapeamento de Erros....41
2.3.6 Teste de Correlação de Desempenho....43
2.4 REFLEXÃO SOBRE COLETA DE DADOS DURANTE A CONSTRUÇÃO DE PROGRAMAS....44
**3 MATERIAIS E MÉTODOS....46**
3.1 MÉTODO PARA COLETA E ANÁLISE....46
3.1.1 Coleta de Dados....48
3.1.2 Análise do Artefato Desenvolvido....49
3.1.3 Análise dos Dados Coletados....50
3.1.4 Retorno Pós-Coleta....51
3.1.5 Busca e Mapeamento de Padrões....52
3.1.6 Investigação de Correlação com Desempenho Acadêmico....53
3.2 FERRAMENTAS DESENVOLVIDAS....54
3.3 FERRAMENTA DE COLETA....55
3.3.1 Arquivo de Exportação....56
3.3.2 Dificuldades encontradas....60
3.4 FERRAMENTA DE CONSOLIDAÇÃO E ANÁLISE....61

3.4.1 Resumo das Atividades Geradas....62
3.4.2 Consolidação dos Dados do Experimento....65
3.4.3 Estrutura do Banco de Dados....68
3.5 REFLEXÃO A RESPEITO DO MÉTODO E FERRAMENTAL PROPOSTO....69
**4 EXPERIMENTOS REALIZADOS E RESULTADOS....71**
4.1 EXERCÍCIOS UTILIZADOS....72
4.1.1 Exercício 1 – Estruturas de repetição....72
4.1.2 Exercício 2 – Vetores....72
4.1.3 Exercício 3 – Busca binária....73
4.1.4 Exercício 4 – Ordenação....73
4.2 CONSOLIDAÇÃO DE DADOS DO EXPERIMENTO....74
4.3 EXPERIMENTOS 1 E 2....75
4.3.1 Resultados da primeira coleta – experimento 1....76
4.3.2 Resultados da segunda coleta – experimento 1....78
4.3.3 Resultados da primeira coleta – experimento 2....81
4.3.4 Resultados da segunda coleta – experimento 2....84
4.4 EXPERIMENTO 3....87
4.4.1 Resultados da primeira coleta – experimento 3....88
4.4.2 Resultados da segunda coleta – experimento 3....90
4.5 EXPERIMENTO 4....93
4.5.1 Resultados da primeira coleta – experimento 4....94
4.5.2 Resultados da segunda coleta – experimento 4....96
4.6 EXPERIMENTO 5....99
4.6.1 Resultados da primeira coleta – experimento 5....100
4.6.2 Resultados da segunda coleta – experimento 5....102
4.7 CORRELAÇÃO DADOS COLETADOS X DESEMPENHO ACADÊMICO....105
4.7.1 Investigação de correlação – experimento 1....105
4.7.2 Investigação de correlação – experimento 2....110
4.7.3 Investigação de correlação – experimento 3....114
4.7.4 Investigação de correlação – experimento 4....115
4.7.5 Investigação de correlação – experimento 5....119
4.8 PERCEPÇÃO DOS ALUNOS....120
4.9 REFLEXÃO SOBRE OS RESULTADOS....121
**5 CONCLUSÃO E TRABALHOS FUTUROS....125**

5.1 CONCLUSÃO....................................................................................................125
5.2 AMEAÇAS À VALIDADE DESTE TRABALHO....................................................126
5.3 TRABALHOS FUTUROS....................................................................................127
**REFERÊNCIAS....................................................................................................128**
**APÊNDICE A – EXPERIMENTO COMPLEMENTAR..............................................132**
**APÊNDICE B – PROPOSTA DE MÉTODO PARA COMPARATIVO ENTRE PARADIGMAS DE PROGRAMAÇÃO QUANTO À APRENDIZAGEM...................145**

# 1 INTRODUÇÃO

Desde o advento do computador, a sociedade vem mudando o modo de executar suas tarefas. Neste contexto, tarefas antes realizadas manualmente, com grande esforço e elevado tempo, hoje são realizadas de forma rápida e sem esforços manuais através da automatização de processos realizados via computador, inclusive na forma de software (MEIRELLES, 1994; NORTON, 1997; PRESSMAN, 2016). Por sua vez, o advento do computador gerou novas atividades profissionais, como a de desenvolvimento de programas para computadores, a qual permite elaborar programas que são a essência de um software no tocante a sua execução.

O processo de desenvolvimento de programas para computadores é composto pelas ações de escrever, depurar, manter e testar programas que podem ser executados pelos computadores. Programas de computador são instruções escritas em uma linguagem de programação, as quais são compiladas para algum tipo de código executável para o computador. Em última instância, estas instruções são compiladas ou transformadas em código binário, o que é realmente executado pela máquina ou computador.

Mais precisamente, esse código binário é resultante do processo de compilação do programa em uma linguagem de programação por um dado compilador, o qual contém o gerador de código alvo, que resulta em binário executável em uma arquitetura de computador (BROOKSHEAR, 2006). Em tempo, a transformação em código binário pode ser direta sendo este o código alvo, como ocorre tradicionalmente, ou, no caso de linguagens interpretadas, através de uma camada intermediária que então gera o código binário executável.

Isso posto, a programação de computadores permite compor soluções na forma de software, cujo código binário resultante (direta ou indiretamente) é o artefato executável na arquitetura computacional. Assim, enquanto elemento fundamental do software, a programação faz parte da Engenharia de Software, área que abrange inclusive a Tecnologia da Informação (TI). Em tempo, a Engenharia de Software em si pode ser definida como a aplicação de uma abordagem quantificável, sistemática e disciplinada para o desenvolvimento, operação e manutenção de software (BOURQUE e FAIRLEY, 2014).

Muito embora a programação de computadores esteja intimamente ligada com a área de Engenharia de Software, a luz do exposto acima, é pertinente ressaltar o viés do documento do Conselho Nacional de Desenvolvimento Científico e Tecnológico – CNPq (2012) que lista a divisão de áreas do conhecimento, subáreas e especialidades. Neste documento, observa-se que Linguagens de Programação e Engenharia de Software são especialidades distintas, vinculadas à subárea de Metodologia e Técnicas da Computação (CNPq, 2012).

Em todo caso, a programação de computadores é parte imprescindível no processo de desenvolvimento de software, o qual tem seu mercado crescendo inclusive na República Federativa do Brasil. Segundo o relatório da Associação Brasileira das Empresas de Software – ABES (2016), houve um crescimento no mercado brasileiro de desenvolvimento de software de 30,2%, comparando o ano de 2014 com o ano de 2015. Ainda, o mercado brasileiro representa 2,9% do mercado mundial e conta com 10.140 empresas dedicadas ao desenvolvimento e comercialização de software (ABES, 2016).

Comparando o ano de 2015 com o ano de 2016 o crescimento do mercado brasileiro de software foi de 0,2%, crescimento inferior ao demonstrado na comparação entre o ano de 2014 e o ano de 2015. Entretanto, o número de empresas dedicadas ao desenvolvimento e comercialização de software no Brasil aumentou para 11.237 (ABES, 2017).

Desta forma, o aumento da demanda por profissionais qualificados na área de TI, incluindo e aqui salientado o desenvolvimento de software, demandam que a formação dos futuros profissionais seja qualificada. Esta demanda também é impulsionada em função da corrente agilidade e eficiência exigida pelo mercado, inclusive no tocante ao desenvolvimento software.

## 1.1 JUSTIFICATIVA

O conhecimento em programação é parte fundamental para profissionais do mercado de TI e é parte imprescindível dos currículos de formação dos cursos relativos à Ciência da Computação e afins (ACM/IEEE-CS Joint Task Force on Computing Curricula, 2013; SBC, 2005). Cursos como Bacharelado em Sistemas de Informação, Bacharelado em Ciência da Computação, Bacharelado em Engenharia de Software, Bacharelado em Engenharia da Computação, Tecnólogo em Análise e

Desenvolvimento de Sistemas e afins têm entre as disciplinas primárias àquelas com foco em programação (SBC, 2005).

Na verdade, tendo em conta o impacto do desenvolvimento de software na mudança da sociedade e de suas atividades, há uma tendência em países desenvolvidos de trazer habilidades de programação para ensino em nível equivalente, em geral, ao ensino médio no Brasil (GRANDELL et al., 2006; Ragonis e Ben-Ari, 2005). No Brasil, isso já ocorre em cursos técnicos pertinentes e integrados ao ensino médio (MEC, 2012; IFPR, 2014). Entretanto, ainda a grande maioria da formação de massa crítica efetiva, nesse âmbito, dá-se após o ensino médio, principalmente no ensino superior.

De acordo com o guia para currículos de cursos em Ciência da Computação, desenvolvido em conjunto pela ACM e pela IEEE (2013), existem algumas áreas de conhecimento que são consideradas parte do conhecimento introdutório dos cursos com base na Ciência da Computação e afins. Dentre estas áreas de conhecimento, existem áreas que mantém diretamente o foco no ensino de programação (Fundamentos de Programação, Algoritmos e Complexidade, Linguagens de Programação e Fundamentos de Desenvolvimento de Software), conforme detalha a Força Tarefa Conjunta em Curriculum de Computação da ACM/IEEE-SC (ACM/IEEE-CS Joint Task Force on Computing Curricula, 2013).

Em resumo, o ensino de programação é de suma importância para melhor preparo dos futuros profissionais pertinentes ao desenvolvimento de software. Neste contexto, entretanto, é pertinente ressaltar que disciplinas introdutórias à programação são tidas como difíceis pelos discentes. Em tempo, não raro, elas apresentam grande índice de desistência e reprovação (ROBINS, ROUNTREE e ROUNTREE, 2003; VUJOŠEVIĆ-JANIČIĆ e TOŠIĆ, 2008; KUNKLE e ALLEN, 2016).

Parte da dificuldade encontrada pelos discentes ou estudantes é devida a diversidade de conhecimentos inter-relacionados que devem ser aprendidos ao mesmo tempo como, por exemplo, os relacionados a conceitos fundamentais de programação e os aspectos das linguagens de programação (MASON, 2012). Ao mesmo tempo, conforme relata Holmboe (1999), o aprendizado de programação necessita da conexão entre as habilidades práticas (como funciona) e entre o conhecimento conceitual (porquê funciona ou o que é).

Nesse âmbito, muito é falado sobre a taxa de aprovação e reprovação em disciplinas introdutórias de programação, mas poucos estudos trouxeram números consolidados a respeito (WATSON e LI, 2014). Um dos estudos que trazem números de fontes diversas é o de Bennedsen e Caspersen (2007), no qual relatam estudo realizado a partir de dados fornecidos por educadores de disciplinas introdutórias de programação e informam que a média de taxa de aprovação nestas disciplinas é de 67%.

Inspirados por Bennedsen e Caspersen (2007), Watson e Li (2014) realizaram uma Revisão Sistemática da Literatura e encontraram taxa semelhante ao do estudo anterior. A taxa média de aprovação nas disciplinas introdutórias de programação, conforme estudo de Watson e Li (2014), é de 67,7%.

Ainda, segundo Watson e Li (2014), a taxa média de 67,7% mantém-se durante os anos considerados para o estudo (entre 1979 e 2013) e não foram encontradas diferenças significantes entre os países considerados no estudo. Também relatam que a taxa de aprovação independe das linguagens de programação utilizadas para o ensino em disciplinas introdutórias de programação.

Desta forma, levando em conta que o ensino de programação é imprescindível na atual sociedade e, mesmo com os estudos e técnicas desenvolvidas ao longo dos anos, a taxa de aprovação e as dificuldades apresentadas mantém-se, justifica-se a realização de estudo neste âmbito, buscando propor métodos e ferramentas que visam colaborar com a melhoria do ensino e aprendizado de programação para computadores.

## 1.2 OBJETO DE ESTUDOS

Durante o ensino de programação de computadores diversas etapas estão envolvidas, iniciando com o planejamento da aula, passando por aulas ministradas pelo docente em conjunto com a prática realizada pelo aluno por meio do desenvolvimento de exercícios e, não menos importante, a avaliação do aprendizado e conhecimento do aluno.

A etapa de avaliação do aprendizado e do conhecimento do aluno em programação para computadores é realizada, muitas vezes, levando em conta apenas o artefato final, que é o programa para computador produzido por este. Assim sendo, não tem sido sistematicamente obtido, de forma direta e histórica, a

evolução e as dificuldades enfrentadas pelo aluno durante o desenvolvimento do artefato.

É verdade que este acompanhamento pode até ser realizado através de observação direta realizada pelo professor, porém este tipo de observação é inviável quando se têm diversos alunos desenvolvendo uma atividade ao mesmo tempo. Como exemplo, dentre tantos outros, o laboratório para a disciplina de Fundamentos de Programação 1, do curso de Engenharia de Computação da Universidade Tecnológica Federal do Paraná, tem em torno de 40 alunos para um único professor.

Uma das formas de realizar o acompanhamento da evolução e das dificuldades do aluno durante o desenvolvimento de programas para computador é através da coleta automatizada de dados oriundos de eventos (edição de código, compilação, execução, testes etc.) gerados durante este processo. Em verdade, o interesse por dados dos eventos gerados durante o processo de programação está aumentando, conforme relatam Ihantola et al. (2015).

Ainda, conforme relatam Ihantola et al. (2015), os trabalhos encontrados estão relacionados com abordagens para auxiliar o estudante ou o professor, como correções automáticas dos algoritmos para fornecer retorno, identificar estudantes propensos a desistir da disciplina ou, ainda, extrair padrões e estratégia de programação que podem ser utilizados para informar aos estudantes sobre suas escolhas.

Em tempo, esta coleta de dados pode ser realizada de diversas maneiras, entre elas a instrumentação do ambiente de desenvolvimento utilizado, a implementação de sistemas de versão de código e, ainda, o registro das teclas pressionadas.

A coleta pode levar em conta eventos específicos, como a compilação ou a gravação do arquivo ou, ainda, registrar todas as teclas pressionadas e os movimentos de mouse.

Um dos pontos em comum em grande parte dos trabalhos é a coleta de erros gerados no processo de compilação, como nos trabalhos de Iepsen, Bercht e Reategui (2011), Jadud (2006), Jadud e Dorn (2015), Rodrigo e Baker (2009), Ahmadzadeh, Elliman e Higgins (2005), Munson e Schilling (2015), Norris et al. (2008) e Murphy et al. (2009).

Entretanto, esses trabalhos, todos relatados por Ihantola et al. (2015), apresentam diversos objetivos de pesquisa além da coleta de erros de programação.

Por exemplo, conforme cada pesquisa, há objetivos outros como verificação de estratégia de programação, mapeamento de comportamento de programação, mapeamento de risco de abandono e utilização do ambiente de desenvolvimento.

Há também uma diversidade de características entre os trabalhos que relatam coleta de dados referente à programação, como o ambiente de programação utilizado, a linguagem de programação utilizada e o público utilizado para coleta dos dados. Sendo assim, no Quadro 1 está descrito um resumo com as principais características dos trabalhos pertinentes considerados para revisão bibliográfica desta pesquisa.

QUADRO 1 – PRINCIPAIS CARACTERÍSTICAS DOS TRABALHOS CONSIDERADOS

| Autores | Principais Características | Ambiente/Linguagem | Público |
|---|---|---|---|
| Silva et. al (2014) | Foco na análise automática de algoritmos;<br>Considera apenas o resultado final do algoritmo. | Python | Estudantes de graduação – Uma instituição – Localização: Brasil |
| Iepsen, Bercht e Reategui (2011) | Foco na detecção de comportamentos que indicam frustração;<br>Coleta dados oriundos de eventos de programação (compilação, quantidade de erros, tempo de desenvolvimento). | Pseudocódigo | Estudantes de graduação – Uma instituição – Localização: Brasil |
| Blikstein (2011) | Realização de prova de conceito para utilização de *learning analytics.* | NetLogo | Estudantes de graduação – Uma instituição (apenas 9 estudantes) – Localização: Estados Unidos da América |
| Blikstein et al. (2014) | Aplicação de *machine learning* em dados coletados durante o desenvolvimento;<br>Coleta o código completo a cada evento de salvar. | IDE Eclipse / Linguagem Java | Estudantes de graduação – Uma instituição – Localização: Estados Unidos da América |
| Jadud (2006) | Apresenta método e ferramental para explorar o comportamento de compilação dos estudantes;<br>Coleta o código e a saída do compilador a cada evento de compilação;<br>Devido à característica da IDE apenas um erro de compilação é apresentado e coletado por vez. | IDE BlueJ / Linguagem Java | Estudantes de graduação – Uma instituição – Localização: Reino Unido |
| Jadud e Dorn (2015) | Amplia a coleta do trabalho anterior para um número superior de usuários; | IDE BlueJ / Linguagem Java | 27.698 usuários da plataforma BlueJ de 10 diferentes |

| Autores | Principais Características | Ambiente/Linguagem | Público |
|---|---|---|---|
| | Não tem controle a respeito de quem são estes usuários. | | países – não é possível atribuir o nível de conhecimento dos usuários |
| Rodrigo e Baker (2009) | Detecção de frustração dos estudantes com base no local de edição e repetição dos erros de compilação. | IDE BlueJ / Linguagem Java | Estudantes de graduação – Uma instituição – Localização: Filipinas |
| Ahmadzadeh, Elliman e Higgins (2005) | Foco na análise do padrão de compilação e depuração de código;<br>Parte do estudo coleta e mapeia erros de compilação em variados exercícios, outra parte do estudo foca no padrão de depuração. | IDE JCreator / Linguagem Java | Estudantes de graduação – Uma instituição – Localização: Reino Unido |
| Thomas, Karahasanovic e Kennedy (2005) | Investigação a respeito impacto dos padrões de digitação na performance de programação. | Java e ADA | Estudantes de graduação – Duas instituições – Localização: Noruega e Reino Unido |
| Munson e Schilling (2015) | Verifica a ordem em que os erros de compilação apresentados são solucionados pelo programador;<br>Coleta os erros de compilação e o local de edição do código. | IDE CodeWork / Linguagem Java | Estudantes de graduação – Uma instituição – Localização: Estados Unidos da América |
| Norris et al. (2008) | Coleta dados oriundos de eventos de programação;<br>Os dados não são demonstrados de forma coletiva, apenas 3 exemplos de comportamentos são demonstrados. | IDE BlueJ / Linguagem Java | Estudantes de graduação – Uma instituição (apenas detalhou os dados de 3 estudantes) – Localização: Estados Unidos da América |
| Murphy et al. (2009) | Coleta dados oriundos de eventos de programação;<br>Os dados coletados não foram divulgados. | IDE BlueJ e Eclipse / Linguagem Java | Estudantes de graduação – Uma instituição – Localização: Estados Unidos da América |
| McKeogh e Exton (2004) | Monitora o comportamento do programador no ambiente de programação;<br>Coleta as ações e interações realizadas com a IDE. | IDE Eclipse / Linguagem Java | Não realizou coleta de dados |
| Yoon e Myers (2011) | Descreve ferramenta para coleta de dados;<br>Resultados focados no comportamento do programador na IDE. | IDE Eclipse / Linguagem Java | Estudantes de graduação – Uma instituição (apenas 12 estudantes) – Localização: Estados Unidos da América |

FONTE: O autor (2018).

Os trabalhos considerados, que são detalhados no capítulo 2, relatam ferramentas e coletas de dados oriundos da programação para computadores, cada um com o seu objetivo e nível de detalhamento do que é coletado e também de como os dados são consolidados. Dentre os trabalhos considerados, é possível observar que existe uma tendência quanto à escolha da linguagem de programação Java, sendo que dos 14 trabalhos apenas 4 utilizam outras linguagens. Também quanto ao público de coleta de dados é possível destacar que apenas 2 trabalhos utilizam estudantes de instituições diferentes, sendo que em 1 destes trabalhos não é possível identificar a origem dos dados.

Nos trabalhos considerados para a revisão não foi possível encontrar trabalhos que mapeiem padrões de comportamento durante a programação para computadores quanto à compilação e execução do artefato e também quanto aos erros de compilação gerados durante o desenvolvimento. Também não foram encontrados trabalhos que relacionem os dados coletados com mapeamento de (possíveis) erros de lógica encontrados no artefato desenvolvido. Ainda, não é possível encontrar trabalhos que relacionem os dados coletados a partir de eventos de programação (número de compilações e erros de compilação) com uma nota atribuída ao artefato desenvolvido.

Sendo assim, o objeto de estudos deste trabalho é a coleta de dados de eventos gerados durante o desenvolvimento de programas para computador, visando mapeá-los de maneira estruturada. Busca-se ainda, a partir desses dados estruturados, mapear padrões de comportamento dos estudantes de programação. Isto seria útil para diversas atividades, como a utilização dessas informações como elemento de apoio ao professor no entendimento do desempenho dos alunos e das dificuldades apresentadas por estes durante o desenvolvimento dos programas. Neste caso, isto teria certamente o intuito de auxiliar o professor na elaboração da sua estratégia, materiais e métodos para ensino de programação.

## 1.3 MOTIVAÇÃO

A motivação inicial deste trabalho teve sua semente no âmbito dos paradigmas de programação, seus subparadigmas e suas linguagens de programação. De uma maneira muito resumida, pode-se classificar os paradigmas em dois grandes grupos, o imperativo e o declarativo. No imperativo estariam os

subparadigmas procedimental e orientado a objetos com suas linguagens, enquanto no declarativo estariam os subparadigmas funcional e lógico com suas linguagens que incluem sistemas baseados em regras. Certamente, esta classificação não seria rígida, havendo sim interseções (BANASZEWSKI, 2009; RONSZCKA, 2012).

Na verdade, a motivação inicial deste trabalho teve origem na indagação a respeito de como realizar comparativos entre paradigmas de programação, no tocante as suas linguagens, quanto à facilidade de programação. Esta indagação, para o autor do trabalho, surgiu durante estudos relacionados a linguagens e paradigmas de programação, especialmente do chamado Paradigma Orientado a Notificações (PON) cuja linguagem de programação herda características lógico-declarativas em termos de expressão (SIMÃO e STADZISZ, 2008; SANTOS, SIMÃO e FABRO, 2017; RONSZCKA et al., 2017).

Nestes estudos, organizados por grupos de estudos da Universidade Tecnológica Federal do Paraná – UTFPR, que estudam inclusive o PON, são realizadas, entre outras atividades, desenvolvimento de programas em diferentes paradigmas de programação, sendo o mesmo programa desenvolvido com a linguagem do PON e com outro paradigma de programação.

Durante o desenvolvimento de uma das atividades pelo autor do trabalho, foi observada a facilidade de desenvolvimento com o PON com sua linguagem lógico-declarativa, relatando uma facilidade maior do que com o paradigma imperativo-procedimental. Porém, esta facilidade era subjetiva, partindo da observação e experiência do autor deste trabalho durante o desenvolvimento. Pertinente mencionar que esta maior facilidade de linguagens lógico-declarativas perante paradigmas imperativos também é observada por outros pesquisadores do grupo de estudos através de suas observações durante o desenvolvimento de atividades semelhantes e aprofundamento teórico (XAVIER, 2014; FERREIRA, 2015; SANTOS, 2017; PORDEUS, 2017).

Neste contexto, foi iniciado pelo autor do trabalho uma pesquisa a respeito de comparativo entre paradigmas de programação via coleta de dados de desenvolvimento, particularmente entre uma linguagem imperativo-procedimental e outra lógico-declarativa. O comparativo entre paradigmas foi inicialmente proposto, conforme relatado no Apêndice B, permitindo alguma experimentação inicial. Entretanto, o principal resultado foi a percepção da falta de método e ferramental

apropriado para a coleta de dados para comparar linguagens mesmo que de um mesmo paradigma.

Ao bem da verdade, após reflexões junto ao grupo de pesquisa pertinente e a docentes do PPGCA/UTFPR, percebeu-se que haveria a falta de método e ferramental apropriado até mesmo para acompanhar, via coleta de dados, a evolução de um indivíduo em apenas uma dada linguagem de programação. Assim, após confirmação junto ao estado da arte, essa problemática mais fundamental motivou o objeto de pesquisa desse presente trabalho.

Finalmente, tem-se também como motivação de realização deste trabalho a possibilidade de aliar as atividades de pesquisa e as profissionais do autor deste trabalho enquanto docente de programação no Instituto Federal do Paraná – IFPR. O fato de focar a pesquisa em um tema ainda mais pertinente ao ensino de programação para computadores, buscando o desenvolvimento de métodos e ferramentas úteis ao auxílio no desenvolvimento de suas atividades, permite que a dissertação se encontre no âmbito da Computação Aplicada que enseja a sigla PPGCA.

## 1.4 OBJETIVO GERAL E OBJETIVOS ESPECÍFICOS

O objetivo geral deste trabalho é propor um método e ferramental para coletar e analisar dados de eventos gerados durante o desenvolvimento de programas para computador, mapeando padrões de comportamento de programação de cada indivíduo de um grupo observado, com intuito de permitir auxiliar na análise da sua evolução em programação, procurando assim trazer possibilidades de aplicação disso no processo de ensino-aprendizagem.

### 1.4.1 Objetivos Específicos

Os objetivos específicos do trabalho são:

a) Investigar quais eventos gerados durante a programação para computadores possam indicar padrões de comportamento de programação;
b) Propor e implementar método para coleta e análise de dados gerados durante a programação para computadores que possibilitem o mapeamento de padrões de comportamento de programação;

c) Coletar dados durante a programação para computadores, à luz de método proposto, por meio da proposição de ferramenta desenvolvida especificamente para tal;
d) Consolidar e analisar os dados coletados por meio da proposição de uma ferramenta desenvolvida especificamente para mapear os dados pertinentes;
e) Investigar possíveis correlações entre os dados coletados e o desempenho acadêmico atribuído à tarefa desenvolvida durante a coleta de dados.

## 1.5 ESTRUTURA DO TRABALHO

O presente trabalho está estruturado da seguinte forma: o Capítulo 2 apresenta a fundamentação teórica do trabalho, o Capítulo 3 descreve os materiais e métodos utilizados, o Capítulo 4 apresenta os experimentos realizados e os dados coletados, finalmente, o Capítulo 5 apresenta as conclusões obtidas através da análise e execução dos experimentos.

# 2 FUNDAMENTAÇÃO TEÓRICA

Com o objetivo de dar base à discussões e descrever o cenário de pesquisa atual, o presente capítulo apresenta a fundamentação teórica relacionada a um breve relembrar das dificuldades do tocante ao ensino de programação e principalmente à coleta de dados durante o processo de desenvolvimento de programas para computadores.

## 2.1 DIFICULDADES NO ENSINO DE PROGRAMAÇÃO

A programação de computadores faz parte dos currículos de cursos da área de Computação, seja em nível superior ou técnico profissionalizante. Durante o decorrer dos cursos, ela pode ser abordada em diversos níveis, do mais básico ao mais aprofundado. Neste âmbito uma das disciplinas introdutórias da grande parte dos currículos destes cursos é uma disciplina chamada Introdução a Programação, dentre outros nomes similares ou afins (ACM/IEEE-CS Joint Task Force on Computing Curricula, 2013; SBC, 2005).

Nesta disciplina introdutória é onde grande parte dos estudantes tem seu primeiro contato com a programação de computadores. Assim, tal disciplina serve como base para conceitos subsequentes mais aprofundados de programação. Portanto, tal disciplina é de suma importância para os futuros profissionais no âmbito da Engenharia de Software (ES), uma vez que ela fundamenta os conhecimentos primários de programação que são *sine qua non* para os demais conhecimentos necessários para a fase de ES chamada de desenvolvimento.

Conforme Kunkle e Allen (2016), diversos autores concordam que o ensino de programação não é tarefa fácil, de fato, muitos estudantes experimentam dificuldades quando aprendem a programar. Robins, Rountree e Rountree (2003) também relatam que aprender a programar é difícil e que programadores novatos sofrem devido a uma ampla quantidade de dificuldades de entendimento dos conceitos de programação. Conforme Mason (2012), uma das razões encontradas na literatura do porque aprender programação é tão difícil está relacionada à quantidade de conceitos inter-relacionados como os conceitos fundamentais de programação e aspectos das linguagens de programação que devem ser aprendidos ao mesmo tempo.

A respeito da taxa de aprovação em disciplinas introdutórias de programação, Watson e Li (2014) encontraram uma taxa de 67,7% de aprovação, independente da localização geográfica dos alunos e da linguagem de programação utilizada. O estudo realizado por Watson e Li (2014) corrobora o estudo anterior realizado por Bennedsen e Caspersen (2007), que encontrou taxa de 67% de aprovação. Em suma, disciplinas de programação são tidas como difíceis e geralmente têm grandes taxas de desistências.

Isto dito, existem diversos métodos que podem ser utilizados para o ensino de programação, como fluxogramas, algoritmos em pseudocódigo e mesmo a utilização direta de uma linguagem de programação (BINI, 2010; BORGES, 2000; NOBRE e MENEZES, 2002). Independente do método utilizado, é necessário que haja a utilização da teoria relacionada com a prática. Holmboe (1999) enfatiza que um bom entendimento de programação de computadores requer habilidades práticas e entendimento conceitual e, ainda, a conexão entre os dois.

A utilização de métodos e ferramentas para auxiliar o ensino de programação sempre fez parte de estudos e experimentos. Isto se deve inclusive às suas múltiplas variáveis como paradigmas de programação, linguagens de programação e forma de aprendizado. Assim, provavelmente, este assunto deve permanecer objeto de estudo por muitos anos.

## 2.2 ERROS DE COMPILAÇÃO

Um dos pontos levados em consideração durante o processo de ensino-aprendizagem de programação para computadores é a ocorrência de erros de compilação. Erros de compilação ocorrem antes da efetiva execução do programa e tem em sua causa fatos como, por exemplo, comandos escritos de forma incorreta ou fora de ordem.

Embora existam algumas classificações gerais para tipos de erros de compilação, não existe um consenso e um padrão a respeito. Em um âmbito geral, os erros de compilação são classificados como léxicos, sintáticos ou semânticos. Conforme o Lawrence Livermore National Laboratory (2007), erros léxicos são aqueles relacionados ao uso de caracteres ilegais, por exemplo, acentos ou símbolos inválidos. Já os erros sintáticos são encontrados durante a análise do código do programa, no processo de compilação, e são causados por instruções

gramaticalmente incorretas. Exemplos de erros sintáticos comuns são: ausência de operador; uso de mais de um operador na mesma linha; duas instruções sem a separação do finalizador de instrução (ponto e vírgula); parênteses incompletos (abertura sem fechamento); e uso de palavras reservadas.

Por sua vez, erros semânticos têm o efeito durante a execução do código, embora alguns possam ser encontrados no momento da compilação. Erros semânticos não tem relação com a forma como as instruções são construídas, mas sim com o conteúdo destas. Exemplos de erros semânticos comuns são: tipo de variável incorreto; tamanho de variável; variável não declarada; e subscritos fora do alcance.

Quando fala-se sobre erros de compilação é importante deixar claro que, conforme McCall (2016), a ocorrência de um erro não é necessariamente devido a um equívoco do programador por falta de conhecimento, em algumas situações o erro pode ocorrer devido a um simples erro de digitação ou erro na escrita de um comando.

Ainda, conforme McCall (2016) existem os erros lógicos na construção do programa que, por sua vez, ocorrem devido à implementação incorreta de um algoritmo ou outro erro que não está diretamente relacionado a conhecimentos da linguagem de programação. Este tipo de erro é detectado a partir da análise do programa ou em tempo de execução, quando o programa não realiza as tarefas conforme o esperado. Desta forma, erros lógicos estão fora do escopo da classificação de erros de compilação.

## 2.3 COLETA DE DADOS DURANTE A CONSTRUÇÃO DE PROGRAMAS

Durante o processo tradicional de ensino-aprendizagem de programação, na maioria das vezes, apenas o código final produzido pelo estudante é levado em conta. Entretanto, o interesse por informações que podem ser coletadas durante o processo de codificação vem aumentando. Neste quadro, Ihantola et al. (2015) relatam uma revisão sistemática da literatura a respeito de mineração de dados educacionais e *Learning Analytics* aplicados no aprendizado de programação.

Conforme Ihantola et al. (2015) há um crescimento substantivo a respeito da coleta de dados referente ao aprendizado de programação. A revisão relatada pelos autores leva em conta o período de 2005 até 2015 e considera 76 trabalhos. Dentre

estes trabalhos, a grande maioria (81%) apresentou dados coletados a partir de uma única instituição e 34% dos trabalhos consideraram dados de disciplinas distintas.

Ihantola et al. (2015) relata que a linguagem que mais foi utilizada pelos trabalhos considerados foi Java, sendo utilizada em 48% dos trabalhos. Python foi utilizada em 11% dos trabalhos e C++ em 7%. 9% dos trabalhos utilizaram mais do que uma linguagem e 18% não especificaram qual linguagem foi utilizada.

Durante a revisão realizada por Ihantola et al. (2015) os trabalhos foram classificados de acordo com o objetivo da pesquisa conforme listado no Quadro 2.

QUADRO 2 – CLASSIFICAÇÃO DOS TRABALHOS DE ACORDO COM O OBJETIVO DA PESQUISA

| Objetivo da pesquisa | Quantidade de trabalhos |
|---|---|
| Habilidade e conhecimento | 3 |
| Estado afetivo | 1 |
| Comportamento do estudante | 7 |
| Dificuldades | 4 |
| Risco de abandono e performance | 8 |
| Análise de algoritmo | 3 |
| *Feedback* automático | 7 |
| Atribuição de nota automática | 2 |
| Utilização do ambiente de desenvolvimento | 5 |
| Testes | 3 |
| Comportamento de programação | 8 |
| Erros de programação | 11 |
| Padrões de programação | 2 |
| Processo de programação | 2 |
| Progresso de programação | 2 |
| Estratégia de programação | 10 |
| Métricas de programação | 3 |
| Comportamento de testes | 2 |

FONTE: Ihantola et al. (2015).

No geral, os trabalhos encontrados por Ihantola et al. (2015) estão relacionados com abordagens para auxiliar o estudante ou o professor, como correções automáticas para fornecer retorno, identificar estudantes propensos a desistir do curso ou, ainda, extrair padrões e estratégia de programação que podem ser utilizados para informar aos estudantes sobre suas escolhas. Alguns trabalhos ainda são dirigidos a análises pós-coleta, como correlação com desempenho acadêmico e outras variáveis do contexto.

Dentre os trabalhos relatados por Ihantola et al. (2015) alguns foram escolhidos e detalhados na próxima subseção por serem pertinentes aos objetivos desta pesquisa. Os trabalhos escolhidos são os escritos por Jadud (2006),

Ahmadzadeh, Elliman e Higgins (2005), Thomas, Karahasanovic e Kennedy (2005) e Norris et al. (2008).

Além dos trabalhos citados no parágrafo imediatamente anterior, outros trabalhos também foram considerados para a revisão bibliográfica desta pesquisa por estarem relacionados aos objetivos da mesma, sendo os trabalhos escritos por Silva et. al (2014), Iepsen, Bercht e Reategui (2011), Blikstein (2011), Blikstein et al. (2014), Jadud e Dorn (2015), Rodrigo e Baker (2009), Munson e Schilling (2015), Murphy et al. (2009), McKeogh e Exton (2004) e Yoon e Myers (2011). Estes trabalhos foram encontrados através de buscas em bases de artigos como IEEE Xplore[1], ACM Digital Library[2] e Periódicos da Capes[3] utilizando os termos de busca programação, compilação, eventos e aprendizagem, em diferentes combinações, sendo pesquisados em português e inglês.

Desta forma, com base nos trabalhos considerados as próximas subseções tratam a respeito da coleta de dados, trabalhos correlatos, dados coletados e uma breve reflexão a respeito do assunto.

### 2.3.1 Coleta de dados

A coleta de dados gerados durante o processo de construção de um programa pode ser feita de diversas formas, sendo que a granularidade dos dados também varia de situação para situação.

Ihantola et al. (2015) relata quatro abordagens distintas para coleta automática de dados:

- Sistemas de atribuição automática de notas: ferramentas que coletam e validam (corrigem) o exercício, consideram o código produzido a partir da submissão (envio) realizado pelo estudante, normalmente quando o estudante entende que o código está pronto para validação, verificando o conteúdo do mesmo e atribuindo uma nota de forma automática. Uma limitação para esta forma de coleta de dados está na falta de visibilidade do que acontece entre as distintas submissões, pois apenas o código final é considerado.

---

[1] https://ieeexplore.ieee.org/Xplore/home.jsp

[2] https://dl.acm.org/

[3] http://www.periodicos.capes.gov.br/

- Instrumentação do ambiente de desenvolvimento: ferramentas utilizadas para coletar eventos individuais realizados pelo estudante no ambiente de desenvolvimento. Normalmente os eventos considerados são compilações, execuções e o evento de salvar.

- Sistemas de controle de versões: considera apenas os diferentes estados de um código, sem levar em conta os eventos relacionados ao processo de desenvolvimento.

- Registro de teclas (*Key logging*): o nível mais fino de granularidade, onde a utilização de teclas (ou ausência de utilização) é considerado. Nesta forma o ambiente de desenvolvimento é dotado de ferramentas que funcionam como um *keylogger*, capturando cada ação do teclado individualmente.

### 2.3.2 Trabalhos correlatos

Nesta subseção estão relatados os trabalhos considerados para a pesquisa de acordo com os objetivos da mesma, descrevendo informações para embasar o projeto proposto. Estes trabalhos estão, inicialmente, relatados de forma descritiva, considerados os principais pontos e consolidados ao final da seção, junto à reflexão a respeito dos mesmos.

Silva et. al (2014) descrevem em seu trabalho a construção de um arcabouço para a análise, de forma automática, de algoritmos desenvolvidos em disciplinas introdutórias de programação. Através do arcabouço proposto é possível classificar os alunos em alguns grupos: alunos que não conseguiram ultrapassar a barreira sintática; alunos que não conseguiram utilizar os construtos (termo utilizado por Silva et al. (2014) para definir estruturas de controle) da linguagem da maneira correta; e alunos que não conseguiram propor soluções estruturalmente coesas com a solução esperada pelo professor.

A construção do arcabouço proposto por Silva et al. (2014) tem como principal característica a utilização em conjunto de diferentes tipos de analisadores de código (e.g., analisador sintático, analisador semântico, analisador estrutural, analisador de construtos). Tal utilização em conjunto tem a função de prover uma análise mais abrangente, pois permite que diferentes aspectos de um código possam ser verificados.

O arcabouço utiliza o código do programa completo para fins de avaliação. A validação do arcabouço foi realizada com a análise de 816 códigos escritos na linguagem Python, desenvolvidos por alunos de um curso de Sistemas de Informação. O resultado com maior destaque do trabalho é a possibilidade de identificação dos estudantes em diversos grupos, destacando-se: (1) estudantes com problemas a nível sintático; (2) estudantes com problemas na utilização de estruturas de controle; e (3) estudantes com problemas em construir soluções que mantenham um nível mínimo de conformidade com as expectativas do professor da disciplina.

Iepsen, Bercht e Reategui (2011) descrevem o desenvolvimento e validação de uma ferramenta que pode ser utilizada durante as aulas de programação para detectar o nível de frustração do aluno. Conforme os autores, algumas das observações comportamentais que podem ser obtidas através de um sistema computacional são o tempo que o aluno leva para realizar uma tarefa, o número de erros de compilação que ele comete durante a execução de uma atividade, a solicitação de ajuda, o uso do botão voltar e as palavras utilizadas em comentários de código.

A ferramenta relatada por Iepsen, Bercht e Reategui (2011) foi desenvolvida para plataforma web e é preparada para a programação utilizando pseudocódigo. A ferramenta coleta algumas informações para análise, entre elas, número de compilações com erros, número total de erros de compilação, tempo entre o início e a última compilação do programa e o número de compilações sem erro de sintaxe. Os erros de compilação coletados não foram classificados e relatados no trabalho escrito por Iepsen, Bercht e Reategui (2011).

Foi realizado um estudo de caso com 58 alunos do curso de Tecnologia em Análise e Desenvolvimento de Sistemas. Através deste estudo foi possível observar diferentes evidências que podem estar relacionadas ao estado de frustração do aluno, como elevado número de tentativas de compilação de um programa sem sucesso, elevado número de programas com erro de compilação e/ou grande quantidade de tempo utilizada para resolver um algoritmo.

Blikstein (2011) relata um caso de estudo exploratório a respeito da possibilidade de utilização de *Learning Analytics* e mineração de dados educacionais para inspecionar o comportamento e a aprendizagem dos estudantes em ambientes de desenvolvimento de software, nos quais métodos tradicionais de verificação não

captam a evolução do estudante. O objetivo principal relatado por Blikstein (2011) é realizar uma prova de existência a respeito de que a utilização de *logs* gerados automaticamente a partir da programação realizada por estudantes pode ser utilizada para inferir padrões a respeito de como os estudantes programam, possibilitando projetar de forma melhor os materiais e as estratégias para o ensino de programação.

Para esta prova de existência relatada por Blikstein (2011) foi utilizado o ambiente NetLogo, no qual é possível capturar eventos como teclas pressionadas, cliques do mouse, alteração em variáveis e alterações no código. Foram coletadas atividades de nove estudantes.

Em suma, Blikstein (2011) indica que a frequência de compilação do código, em conjunto com o tamanho deste código, habilita traçar uma aproximação razoável de um estilo e um padrão de codificação. Ainda, conforme o autor, entendendo melhor o estilo de codificação e o comportamento de cada aluno é possível ter uma nova visão do processo cognitivo do estudante, fornecendo um panorama do processo de programação e como isso afeta o entendimento do estudante a respeito de uma linguagem de programação e habilidades para solucionar problemas.

Blikstein et al. (2014) relatam em seu trabalho a utilização de captura automática de eventos gerados durante a programação de computadores por 370 estudantes de curso de graduação, aplicando técnicas de aprendizado de máquina para traçar o progresso dos estudantes.

Conforme Blikstein et al. (2014) existem algumas lacunas na literatura com relação ao estudo de dados coletados durante a programação de computadores, sendo eles: (a) grande parte dos experimentos são pequenos, contando com codificação manual e observação; (b) estudos com grande quantidade de dados não capturam dados no nível de detalhe necessário para construir complexos modelos do processo de programação dos estudantes; e (c) as tarefas propostas aos estudantes são comprimidas na forma de pequenos exercícios em vez de problemas de programação que refletem o mundo real.

Blikstein et al. (2014) relata coleta de dados realizada no ambiente de desenvolvimento Eclipse, coletando todo o código gerado a cada evento de salvar ou compilar o programa. Conforme as observações relatadas por Blikstein et al. (2014) os resultados dos estudos apontam que o tamanho do código em cada atualização não está associado à performance acadêmica na disciplina, porém as

mudanças no padrão de como os estudantes realizam alterações no código do programa no decorrer da disciplina estão relacionadas com o desempenho acadêmico do aluno, sendo que alunos que mudam seu padrão de alteração têm maior desempenho acadêmico.

Jadud (2006) descreve em seu trabalho um experimento realizado com alunos de um curso de graduação em Ciência da Computação, durante os anos de 2003 (62 alunos), 2004 (56 alunos) e 2005 (68 alunos). Neste experimento é utilizada a coleta de dados durante o processo de programação em Java, utilizando o ambiente de desenvolvimento BlueJ.

No trabalho relatado por Jadud (2006) foram utilizados apenas os computadores da universidade em que o experimento foi aplicado e, a cada evento de compilação do programa desenvolvido pelos alunos, foi coletado o código completo e a saída do compilador. Conforme Jadud (2006), a partir dos dados coletados é possível identificar o tipo de erro em cada compilação (se houver), o tempo que o estudante levou entre uma compilação e outra, a diferença do número de caracteres entre uma compilação e outra e a localização no programa onde o código foi editado entre as compilações. Ressalta-se que, conforme Jadud (2006), a ferramenta BlueJ informa apenas um erro de compilação por vez, mesmo que o programa contenha mais erros no momento da compilação do código.

Nas coletas dos experimentos realizados por Jadud (2006) foram encontrados erros de compilação classificados nas seguintes categorias: utilização de variável desconhecida; ausência de ponto e vírgula; parênteses esperados; método desconhecido; erro de aplicação de método; início ilegal de expressão; classe desconhecida; identificador esperado; tipos incompatíveis; classe ou interface esperada; violação de acesso esperado; e falta de retorno.

A partir dos dados coletados, Jadud (2006) utilizou uma métrica chamada Quociente de Erro (QE), na qual verifica os pares de compilação e atribui pontuação de acordo com a presença ou não de erro de compilação, além do tipo e da localização do erro no código. Isto permite a verificação se o erro encontrado na última compilação foi resolvido ou não. Verifica-se ainda se o mesmo erro está sendo repetido compilação após compilação.

Os dados de 56 estudantes foram utilizados por Jadud (2006) para análise de correlação entre o Quociente de Erro e o desempenho acadêmico durante o ano. Embora entre os dados analisados exista uma correlação entre o QE e o

desempenho acadêmico, não foi possível afirmar firmemente a existência desta correlação.

Em outro trabalho, Jadud e Dorn (2015) aplicam a métrica QE para 27.698 usuários do BlueJ, coletando dados através do projeto BlackBox. Nesta análise não é possível verificar qual é o tipo do usuário, se é um novato ou um especialista em programação. Houve um pré-filtro dos dados para análise, sendo que os dados analisados são apenas de usuários do hemisfério norte.

Conforme Jadud e Dorn (2015), não foi possível verificar uma distribuição normal da métrica QE entre os usuários do BlueJ. Entretanto, é possível observar uma semelhança entre usuários do mesmo país.

Fazendo referências ao trabalho de Jadud (2006), Rodrigo e Baker (2009) relatam a utilização de dados coletados em plano de fundo durante o desenvolvimento de programas de computadores para detectar frustração dos estudantes durante o desenvolvimento.

Rodrigo e Baker (2009) também utilizam a interface BlueJ e a linguagem de programação Java. A inspiração do trabalho de Jadud (2006) é oriunda da utilização de pares de compilação com o mesmo erro de compilação na mesma localização do programa ou ainda edições consecutivas na mesma localização do programa. Esta sequência, segundo os autores, pode indicar a frustração do estudante durante o desenvolvimento do programa. Em conclusão ao estudo, Rodrigo e Baker (2009) informam que embora seja possível indicar padrão de frustração com base nos dados coletados, são necessárias diversas sessões para coleta dos dados e formação deste padrão. Apesar dos erros de compilação terem sido utilizados no trabalho, os mesmos não foram classificados e também não foram divulgados.

Ahmadzadeh, Elliman e Higgins (2005) descrevem análise a respeito dos padrões de depuração de programas por estudantes novatos de Ciências da Computação. O estudo foi dividido em duas partes, sendo a primeira com o foco na coleta do código e erros de compilação gerados durante a solução de exercícios. Já na segunda parte os estudantes receberam um código pronto com alguns erros colocados propositalmente para serem corrigidos.

No estudo relatado por Ahmadzadeh, Elliman e Higgins (2005) foi utilizada a linguagem Java com a IDE JCreator e o compilador Jikes. Para o primeiro estudo foi possível notar padrão no tipo de erros de compilação cometidos pelos estudantes, indicando possíveis deficiências do grupo de estudantes, demonstrando assim onde

o instrutor poderia trabalhar de forma mais focada. Apenas os erros semânticos foram classificados, pois foram os tipos de erro com maior incidência na amostra de dados coletada. As categorias a seguir foram consideradas: campo não encontrado; uso de variável não estática dentro de método estático; incompatibilidade de tipo; uso de variável não inicializada; chamada de método com argumentos incorretos; e nome de método não encontrado.

Já o foco do segundo estudo relatado por Ahmadzadeh, Elliman e Higgins (2005) foi verificar a habilidade de depuração dos estudantes através da correção de erros deixados propositalmente em programas preparados pelos instrutores. Com o resultado dos experimentos foi possível concluir que os estudantes considerados bons depuradores de código são também bons programadores, porém, bons programadores não são necessariamente bons depuradores de código.

Thomas, Karahasanovic e Kennedy (2005) descrevem estudo que objetiva encontrar correlação entre a velocidade de digitação de estudantes de programação e a performance de desenvolvimento dos programas. Neste experimento foram utilizadas as linguagens Java e Ada. Com a linguagem Java foi realizado o experimento em um ambiente controlado e com a linguagem Ada de forma mais livre. Para ambas as linguagens o estudo relatado por Thomas, Karahasanovic e Kennedy (2005) não encontrou correlação entre a velocidade de digitação ou latência e a performance de programação, medida através de atribuição de notas às alterações realizadas em programas.

Munson e Schilling (2015) relatam em seu estudo o acompanhamento do comportamento de estudantes de programação quanto à solução dos erros de compilação encontrados durante o processo de criação de programas de computador. Em seu estudo utilizam a linguagem Java em um ambiente chamado CodeWork.

Uma das observações realizadas por Munson e Schilling (2015) é de que em quase metade das vezes os estudantes corrigem o primeiro erro reportado pelo compilador, considerado pelos autores a maneira correta de corrigir os erros encontrados durante a compilação de programas. Outra observação encontrada pelos autores é de que o desempenho acadêmico dos estudantes nos exercícios de programação está associado ao hábito de solucionar o primeiro erro de compilação encontrado antes de prosseguir com o desenvolvimento do programa. Finalmente, Munson e Schilling (2015) relatam que foi possível observar em seu estudo que os

estudantes que levam maior tempo entre uma compilação e outra apresentam maiores notas no exercício proposto. Os erros de compilação encontrados nos experimentos de Munson e Schilling (2015) não foram classificados e relatados.

Norris et al. (2008) descreve a utilização de um *plugin* chamado ClockIt em conjunto com a Interface de Desenvolvimento (IDE) BlueJ para coletar dados a respeito do desenvolvimento de programas por novatos em programação, a linguagem de programação observada é a linguagem Java.

O *plugin* citado por Norris et al. (2008) coleta eventos como compilação, abertura e fechamento de arquivos. Os eventos podem ser acompanhados pelos estudantes, podendo estes verificar as informações sobre o seu desempenho, identificando seus eventos, verificando a quantidade de código escrito entre as compilações. No trabalho é descrito também uma interface web que contém os dados de diversos estudantes de forma consolidada.

No experimento relatado por Norris et al. (2008) foram coletadas informações de 75 estudantes de Ciência da Computação, durante pequenos exercícios realizados em laboratório. A partir da coleta de dados algumas observações foram feitas no trabalho, especificando situação de três estudantes. O estudante número um, que teve a maior nota, levou mais tempo trabalhando no experimento, porém não escreveu a maior quantidade de código. O estudante número dois levou quase o mesmo tempo que o estudante um e produziu mais código, porém grande parte deste código não é funcional, este estudante tirou uma nota regular. O último estudante levou o menor tempo, escreveu a menor quantidade de código e teve a pior nota. Os erros de compilação cometidos pelos estudantes não foram classificados e relatados.

Murphy et al. (2009) relatam em seu trabalho o desenvolvimento da ferramenta chamada Retina, composta por *plugin* que coleta informações durante o desenvolvimento de programas para computador, ferramenta de análise para o instrutor e ferramenta de verificação de dados para o estudante. O *plugin* relatado por Murphy et al. (2009) é desenvolvido para captar as informações de programas escritos na linguagem Java e funciona nas IDEs BlueJ e Eclipse. O foco de coleta deste *plugin* está na compilação e erros indicados neste processo. Apesar disso, os erros de compilação encontrados nas coletas relatadas por Murphy et al. (2009) não foram relatados.

Na ferramenta desenvolvida para análise pelo professor é possível verificar o desempenho individual de cada estudante e o da classe de forma consolidada. Cada estudante pode verificar seu desempenho através de uma página web desenvolvida para isso. Além disso, caso tenha interesse, o estudante pode usufruir de dicas em tempo real através de aplicativo de mensagens instantâneas interfaceado com a ferramenta Retina.

Conforme relatado por Murphy et al. (2009), a ferramenta Retina foi avaliada com exercícios de programação desenvolvidos por 48 alunos. Através dos dados coletados pela ferramenta foi possível verificar o desempenho da turma e de forma individual, sendo estes dados aproveitados pelos instrutores para ajustes nas suas aulas. A correlação entre os dados coletados e o desempenho acadêmico do estudante foi validada e, entretanto, não foram encontrados dados significantes.

McKeogh e Exton (2004) relatam em seu trabalho o desenvolvimento da ferramenta chamada Eclipse Watcher, um *plugin* que coleta dados durante o desenvolvimento de programas para computador, integrado à IDE Eclipse. Os dados coletados pela ferramenta relatada por McKeogh e Exton (2004) são movimentos do mouse, teclas pressionadas, navegação pelo código e edição do código. O foco dos dados coletados está no mapeamento do tempo utilizado com cada um dos tipos de ações coletados.

Yoon e Myers (2011) relatam o desenvolvimento da ferramenta chamada FLUORITE, um *plugin* para a IDE Eclipse. A ferramenta captura eventos com granularidade fina, por exemplo, inserção de caracteres, alteração de código selecionado, remoção de caracteres indicando o que foi removido e comandos utilizados na IDE. Para possibilitar o acompanhamento da evolução do código o *plugin* captura também o código inicial quando o arquivo é aberto.

Os dados capturados pela ferramenta descrita por Yoon e Myers (2011) são salvos em um arquivo XML, o qual pode ser importado posteriormente para a ferramenta de análise, que por sua vez pode gerar relatórios sobre padrão de edição de código, evolução do tamanho do código e distribuição de teclas pressionadas.

### 2.3.3 Resumo dos trabalhos considerados

Uma consolidação a respeito dos trabalhos relatados nesta seção é apresentada no Quadro 3. Na próxima subseção será relatado a respeito dos dados utilizados durante a coleta em plano de fundo.

QUADRO 3 – CONSOLIDAÇÃO DOS TRABALHOS CONSIDERADOS

| Autores | Objetivo | Forma de Coleta | Resultados | Público |
|---|---|---|---|---|
| Silva et. al (2014) | Propor um arcabouço para análise estrutural de algoritmos de forma automática. | Coleta apenas o artefato final; Linguagem de programação: Python. | Foi possível mapear grupos por padrão de alunos que não conseguiram romper os erros sintáticos, alunos com problemas nas estruturas de controle e alunos que não conseguiram entregar solução com nível de conformidade; Os erros não foram classificados; Não foi realizado teste de correlação com desempenho acadêmico. | Estudantes de graduação – Uma instituição – Localização: Brasil |
| Iepsen, Bercht e Reategui (2011) | Propor método e ferramenta para verificar o estado de frustração dos alunos durante o desenvolvimento de programas para computador. | Ferramenta WEB desenvolvida para coletar eventos de compilação (incluindo erros) e informações pró-ativas do aluno indicando frustração (botão “Preciso de Ajuda” e “Estou Frustrado”); Linguagem de programação: Pseudocódigo. | Foi possível realizar um mapeamento dos erros de compilação em conjunto com o acionamento dos botões “Preciso de Ajuda” e “Estou Frustrado”; Não foi realizado teste de correlação com desempenho acadêmico. | Estudantes de graduação – Uma instituição – Localização: Brasil |
| Blikstein (2011) | Estudo experimental para coleta de dados de eventos de programação e inferir padrões de programação dos estudantes através da utilização de *learning analytics.* | Instrumentação de IDE para coleta de eventos de programação (compilações, execuções, teclas pressionadas, botões utilizados e alterações de código); Linguagem de programação: NetLogo. | Indica que é possível traçar perfis de programação para computadores a partir dos dados gerados; Os erros gerados não foram mapeados. | Estudantes de graduação – Uma instituição (apenas 9 estudantes) – Localização: Estados Unidos da América |
| Blikstein et al. (2014) | Utilizar métodos de *machine learning* | Instrumentação de IDE – coletas de | Conclui que é possível mapear padrões de | Estudantes de |

| Autores | Objetivo | Forma de Coleta | Resultados | Público |
|---|---|---|---|---|
| | para mapear padrões de programação e prever o desempenho acadêmico através destes. | alteração do código (tamanho e frequência), realizadas a cada compilação ou evento de salvar; IDE Eclipse / Linguagem Java | programação, porém estes não estão relacionados com desempenho acadêmico. | graduação – Uma instituição – Localização: Estados Unidos da América |
| Jadud (2006) | Explorar o comportamento dos programadores sobre a compilação de programas. | Instrumentação de IDE – coleta os eventos de compilação e a saída do compilador; IDE BlueJ / Linguagem Java | Coleta as compilações e verifica os erros entre compilações, calculando o quociente de erro (proposto pelo trabalho). Realizada o teste de correlação entre o QE e o desempenho acadêmico, não foi encontrada correlação; A ferramenta limita a coleta de apenas um erro por compilação; | Estudantes de graduação – Uma instituição – Localização: Reino Unido |
| Jadud e Dorn (2015) | Amplia a coleta do trabalho anterior para um número superior de usuários. | Instrumentação de IDE – coleta os eventos de compilação e a saída do compilador; IDE BlueJ / Linguagem Java | Não foi possível verificar uma distribuição normal da métrica QE entre os usuários do BlueJ. Entretanto, é possível observar uma semelhança entre usuários do mesmo país. | 27.698 usuários da plataforma BlueJ de 10 diferentes países – não é possível atribuir o nível de conheciment o dos usuários |
| Rodrigo e Baker (2009) | Detectar frustração dos estudantes durante a programação para computadores com base na localização de edição de código e a repetição dos erros de compilação. | Instrumentação da IDE – coletando compilação e erros de compilação; IDE BlueJ / Linguagem Java. | É possível gerar um mapeamento dos dados de compilação para gerar indícios de frustração; Não foi realizado o mapeamento dos tipos de erros; Não foi realizado teste de correlação com desempenho acadêmico. | Estudantes de graduação – Uma instituição – Localização: Filipinas |
| Ahmadzadeh , Elliman e Higgins (2005) | Analisar padrão de compilação e depuração durante a programação para computadores. | Instrumentação de IDE – coleta resultado da compilação e erros; IDE JCreator / Linguagem Java. | Foram realizados 2 experimentos. No primeiro experimento, foi testada a correlação entre o tempo entre as compilações e o rendimento acadêmico, foram encontrados indícios de correlação. No segundo experimento foi verificada a lógica de depuração de | Estudantes de graduação – Uma instituição – Localização: Reino Unido |

| Autores | Objetivo | Forma de Coleta | Resultados | Público |
|---|---|---|---|---|
| | | | programas, através de erros colocados no programa para serem corrigidos;<br>Não foi realizado o mapeamento dos erros no artefato final. | |
| Thomas, Karahasanovic e Kennedy (2005) | Investigação a respeito impacto dos padrões de digitação na performance de programação. | Instrumentação da IDE através de *keylogging*;<br>Linguagens de programação: Java e ADA | Encontrado indícios de correlação positiva entre velocidade de digitação e performance de programação. | Estudantes de graduação – Duas instituições – Localização: Noruega e Reino Unido |
| Munson e Schilling (2015) | Analisar a resposta dos estudantes aos erros de compilação. | Instrumentação de IDE – coleta alterações de código e mensagens de compilação;<br>IDE CodeWork / Linguagem Java. | Indica que existe correlação entre o desempenho acadêmico e a correção do primeiro erro indicado a cada compilação;<br>Não mapeia os erros encontrados, limitado a verificar se o primeiro erro indicado pelo compilador foi corrigido. | Estudantes de graduação – Uma instituição – Localização: Estados Unidos da América |
| Norris et al. (2008) | Mapear dados de programação para entender o comportamento dos estudantes | Instrumentação de IDE;<br>IDE BlueJ / Linguagem Java | Indica que é possível realizar a coleta e relata os dados coletados de 3 estudantes. | Estudantes de graduação – Uma instituição (apenas detalhou os dados de 3 estudantes) – Localização: Estados Unidos da América |
| Murphy et al. (2009) | Coletar dados de eventos de programação para fornecer retorno aos alunos e instrutor. | Instrumentação de IDE – coleta eventos de compilação;<br>IDE BlueJ e Eclipse / Linguagem Java. | Testou correlação entre o tempo gasto e o desempenho acadêmico do exercício, correlação não encontrada. Existe uma correlação fraca entre o tempo e o desempenho no semestre;<br>Os dados coletados e os valores das correlações não foram informados. | Estudantes de graduação – Uma instituição – Localização: Estados Unidos da América |
| McKeogh e Exton (2004) | Monitorar o comportamento do programador no ambiente de programação | Instrumentação da IDE – monitora todas as ações realizadas na IDE;<br>IDE Eclipse / Linguagem Java | Foi apresentada a ferramenta e os possíveis usos na educação e na indústria. | Não realizou coleta de dados |

| Autores | Objetivo | Forma de Coleta | Resultados | Público |
|---|---|---|---|---|
| Yoon e Myers (2011) | Descrever ferramenta para coleta de dados do programador com interação com a IDE. | Instrumentação da IDE – coleta eventos de programação e ações realizadas na IDE;<br>IDE Eclipse / Linguagem Java. | O artigo foi baseado na apresentação da ferramenta, concluindo a respeito da forma de coleta;<br>Resultados divulgados focam apenas no padrão de edição de código. | Estudantes de graduação – Uma instituição (apenas 12 estudantes) – Localização: Estados Unidos da América |

FONTE: O autor (2018).

### 2.3.4 Dados utilizados

A partir dos trabalhos relatados na subseção anterior é possível verificar a diversidade de dados coletados durante o desenvolvimento de programas de computador e utilizados para mapear padrões e comportamentos.

Em alguns trabalhos a granularidade do que é coletado é maior, por exemplo, Thomas, Karahasanovic e Kennedy (2005) apresentam ferramenta que coleta cada tecla pressionada buscando correlação com a velocidade de digitação. Também apresentam granularidade fina os trabalhos de Blikstein (2011), Blikstein et al. (2014), McKeogh e Exton (2004) e Yoon e Myers (2011) que coletam teclas pressionadas e cliques do mouse, entre outros dados.

Um dos pontos em comum em grande parte dos trabalhos é a coleta de erros gerados no processo de compilação, como nos trabalhos de Iepsen, Bercht e Reategui (2011), Jadud (2006), Jadud e Dorn (2015), Rodrigo e Baker (2009), Ahmadzadeh, Elliman e Higgins (2005), Munson e Schilling (2015), Norris et al. (2008) e Murphy et al. (2009).

Também alguns trabalhos relatam a coleta do código completo a cada evento de salvar ou compilar o programa gerado, entre eles estão os trabalhos de Silva et. al (2014), Blikstein (2011), Blikstein et al. (2014), Jadud (2006), Jadud e Dorn (2015), Rodrigo e Baker (2009) e Ahmadzadeh, Elliman e Higgins (2005).

A partir dos dados coletados é possível gerar métricas como frequência/velocidade de digitação, tempo ocioso, erros gerados a cada compilação do programa, frequência de compilação, tamanho da alteração entre as compilações, número de compilações com sucesso, número de compilações com falha e tempo de desenvolvimento do programa.

Uma visão consolidada a respeito dos dados coletados está demonstrada no Quadro 4.

QUADRO 4 – CONSOLIDAÇÃO SOBRE OS DADOS COLETADOS

| Autores | Descrição/Objetivo | Teclas pressionadas e movimentos do mouse | Erros de compilação | Código Completo |
|---|---|---|---|---|
| Silva et. al (2014) | Arcabouço para a análise de algoritmos | | | X |
| Iepsen, Bercht e Reategui (2011) | Detecção de nível de frustração | | X | |
| Blikstein (2011) | Estudo exploratório sobre uso de *Learning Analytics* | X | | X |
| Blikstein et al. (2014) | Coleta de eventos durante programação para computadores | X | | X |
| Jadud (2006) | Coleta de eventos durante a programação – Métrica Quociente de Erro | | X | X |
| Jadud e Dorn (2015) | Replicação do trabalho Jadud (2006) | | X | X |
| Rodrigo e Baker (2009) | Detecção de nível de frustração | | X | X |
| Ahmadzadeh, Elliman e Higgins (2005) | Análise sobre o padrão de depuração de programas | | X | X |
| Thomas, Karahasanovic e Kennedy (2005) | Busca de correlação entre padrão de digitação e desempenho acadêmico | X | | |
| Munson e Schilling (2015) | Solução de erros de compilação | | X | |
| Norris et al. (2008) | Coleta de eventos durante programação para computadores | | X | |
| Murphy et al. (2009) | Coleta de eventos durante programação para computadores | | X | |
| McKeogh e Exton (2004) | Coleta de eventos durante programação para computadores | X | | |
| Yoon e Myers (2011) | Coleta de eventos durante programação para computadores | X | | |

FONTE: O autor (2018).

### 2.3.5 Mapeamento de Erros

Conforme relatado nas subseções anteriores, um dos pontos em comum em grande parte dos trabalhos é a coleta dos erros durante o processo de programação. Desta forma, o Quadro 5 lista uma consolidação dos trabalhos a respeito dos erros relacionados ao processo de programação, separados em erros de compilação e erros presentes no artefato final.

QUADRO 5 – CONSOLIDAÇÃO SOBRE MAPEAMENTO DE ERROS

| **Autores** | **Descrição/Objetivo** | **Erros de Compilação** | **Erros Presentes no Artefato Final** |
|---|---|---|---|
| Silva et. al (2014) | Arcabouço para a análise de algoritmos | Não coletados. | Artefatos classificados de acordo com a presença de erros sintáticos, presença de erros em estruturas de controle e solução entregue sem um nível de conformidade de acordo com o esperado pelo instrutor. |
| Iepsen, Bercht e Reategui (2011) | Detecção de nível de frustração | Erros coletados, porém não categorizados. | Não considerados. |
| Blikstein (2011) | Estudo exploratório sobre uso de *Learning Analytics* | Erros coletados, porém não categorizados. | Não considerados. |
| Blikstein et al. (2014) | Coleta de eventos durante programação para computadores | Erros coletados, porém não categorizados. | Não considerados. |
| Jadud (2006) | Coleta de eventos durante a programação – Métrica Quociente de Erro | Coletados, porém apenas o primeiro erro de compilação é relatado/coletado pela ferramenta. | Não considerados. |
| Jadud e Dorn (2015) | Replicação do trabalho Jadud (2006) | Coletados, porém apenas o primeiro erro de compilação é relatado/coletado pela ferramenta. | Não considerados. |
| Rodrigo e Baker (2009) | Detecção de nível de frustração | Erros coletados, porém não categorizados. | Não considerados. |
| Ahmadzadeh, Elliman e Higgins (2005) | Análise sobre o padrão de depuração de programas | Erros coletados, categorizados em léxicos, sintáticos e semânticos. | Não considerados. |
| Thomas, Karahasanovic e Kennedy (2005) | Busca de correlação entre padrão de digitação e desempenho acadêmico | Não coletados. | Não considerados. |
| Munson e Schilling (2015) | Solução de erros de compilação | Erros coletados, porém não categorizados. | Não considerados. |
| Norris et al. (2008) | Coleta de eventos durante programação para computadores | Erros coletados, porém não categorizados. | Não considerados. |
| Murphy et al. (2009) | Coleta de eventos durante programação para computadores | Erros coletados, porém não categorizados. | Não considerados. |
| McKeogh e Exton (2004) | Coleta de eventos durante programação para computadores | Não coletados. | Não considerados. |

| Autores | Descrição/Objetivo | Erros de Compilação | Erros Presentes no Artefato Final |
|---|---|---|---|
| Yoon e Myers (2011) | Coleta de eventos durante programação para computadores | Erros coletados, porém não categorizados. | Não considerados. |

FONTE: O autor (2018).

### 2.3.6 Teste de Correlação de Desempenho

O público utilizado para a coleta de dados nos trabalhos considerados é, em grande parte, de instituições de ensino. Desta forma, é comum a realização do teste de correlação dos dados coletados com o desempenho acadêmico atribuído ao exercício. Sendo assim, é apresentado no Quadro 6 uma consolidação a respeito dos testes de correlação descritos nos trabalhos considerados. Na sequência, a próxima subseção apresenta uma breve reflexão sobre coleta de dados durante construção de programas para computador.

QUADRO 6 – CONSOLIDAÇÃO SOBRE TESTE DE CORRELAÇÃO

| Autores | Descrição/Objetivo | Teste de Correlação |
|---|---|---|
| Silva et. al (2014) | Arcabouço para a análise de algoritmos | Não realizado. |
| Iepsen, Bercht e Reategui (2011) | Detecção de nível de frustração | Não realizado. |
| Blikstein (2011) | Estudo exploratório sobre uso de *Learning Analytics* | Não realizado. |
| Blikstein et al. (2014) | Coleta de eventos durante programação para computadores | Correlação testada entre o desempenho acadêmico e os padrões de programação relativos à forma de alteração/manutenção do código. Correlação não encontrada. |
| Jadud (2006) | Coleta de eventos durante a programação – Métrica Quociente de Erro | Correlação testada entre o desempenho acadêmico e a métrica de Quociente de Erro (proposta por Jadud (2006)). Correlação não encontrada. |
| Jadud e Dorn (2015) | Replicação do trabalho Jadud (2006) | Não realizado. |
| Rodrigo e Baker (2009) | Detecção de nível de frustração | Não realizado. |
| Ahmadzadeh, Elliman e Higgins (2005) | Análise sobre o padrão de depuração de programas | Correlação testada entre o desempenho acadêmico e o intervalo entre as compilações, encontrado indício de correlação positiva entre as duas variáveis. |
| Thomas, Karahasanovic e Kennedy (2005) | Busca de correlação entre padrão de digitação e desempenho acadêmico | Correlação testada entre o desempenho acadêmico e a velocidade de digitação, encontrada correlação positiva entre as duas variáveis. |
| Munson e Schilling (2015) | Solução de erros de compilação | Correlação testada entre o desempenho acadêmico e a correção do primeiro erro de compilação apresentado, encontrada correlação positiva entre as duas variáveis. |

| Autores | Descrição/Objetivo | Teste de Correlação |
|---|---|---|
| Norris et al. (2008) | Coleta de eventos durante programação para computadores | Não realizado. |
| Murphy et al. (2009) | Coleta de eventos durante programação para computadores | Correlação testada entre o desempenho acadêmico e o tempo gasto para o desenvolvimento do programa. Correlação não encontrada. |
| McKeogh e Exton (2004) | Coleta de eventos durante programação para computadores | Não realizado. |
| Yoon e Myers (2011) | Coleta de eventos durante programação para computadores | Não realizado. |

FONTE: O autor (2018).

## 2.4 REFLEXÃO SOBRE COLETA DE DADOS DURANTE A CONSTRUÇÃO DE PROGRAMAS

Conforme relatado nos trabalhos anteriormente mencionados, é possível coletar dados durante o processo de desenvolvimento de programas para computador. É possível verificar também que os dados coletados podem ser utilizados para mapear padrões de aprendizado e de comportamento dos programadores.

A maioria dos trabalhos (84,6%) utilizam uma única linguagem e são aplicados com alunos em uma única instituição de ensino. Alguns trabalhos são aplicados para um público grande, como em Jadud e Dorn (2015), porém não é possível classificar o nível de programação de cada participante, nem se estes são estudantes ou profissionais de TI. Ainda, a título de curiosidade, os trabalhos mencionados, em sua grande maioria (84,6%), coletam dados de instituições localizadas no hemisfério norte, evidenciando a falta de coleta de dados em países do hemisfério sul, como o Brasil.

Conforme revisão escrita por Ihantola et al. (2015), as linguagens mais utilizadas são as linguagens do Paradigma Orientado a Objetos, como Java, Python e C++. Linguagens exclusivamente do Paradigma Imperativo-Procedimental não são citadas. Este dado é reforçado também pelos demais trabalhos considerados na revisão, sendo que é possível observar que existe uma tendência quanto à escolha da linguagem de programação Java, sendo que dos 14 trabalhos apenas 4 utilizam outras linguagens.

Quanto à coleta e classificação de erros relacionados ao processo de programação, 11 dos 14 trabalhos realizam a coleta de erros de compilação, porém, destes 11 trabalhos apenas 3 realizam uma classificação/categorização dos erros de compilação coletados. Já a respeito dos erros presentes nos artefatos entregues, apenas 1 trabalho considera este tipo de erro.

Ainda, 6 dos 11 trabalhos testam a correlação entre o desempenho acadêmico e informações coletadas durante o desenvolvimento de programas para computador. Nestes 6 trabalhos não são relatados testes de correlação entre o desempenho acadêmico e informações relativas às compilações realizadas e aos erros de compilações encontrados.

Nos trabalhos considerados para a revisão não foi possível encontrar trabalhos que mapeiem e disponibilizem os dados a respeito de padrões de comportamento durante a programação para computadores quanto à compilação e execução do artefato e também quanto aos erros de compilação gerados durante o desenvolvimento. Também não foram encontrados trabalhos que relacionem os dados coletados com mapeamento de (possíveis) erros de lógica encontrados no artefato desenvolvido. Ainda, não é possível encontrar trabalhos que relacionem os dados coletados a partir de eventos de compilação (com exceção ao intervalo entre compilações) com uma nota atribuída ao artefato desenvolvido.

Finalmente, levando em consideração o que foi relatado neste capítulo, o próximo capítulo apresenta os materiais e métodos deste trabalho.

# 3 MATERIAIS E MÉTODOS

Neste capítulo serão apresentados os materiais e métodos utilizados no desenvolvimento do trabalho. Inicialmente será apresentado o método proposto pelo trabalho e na sequência será relatado a respeito das ferramentas desenvolvidas para suportar o método.

## 3.1 MÉTODO PARA COLETA E ANÁLISE

Para viabilizar a coleta e análise dos dados oriundos de eventos gerados durante a programação para computadores um método está sendo proposto. Nesta seção será apresentado o método utilizado para realização da coleta e análise dos dados.

O método para coleta e análise é composto por 6 etapas, ilustradas na Figura 1 e explicadas a seguir:

1 – Coleta de dados: realização da coleta de dados em plano de fundo através de ferramenta desenvolvida para este fim. A coleta de dados é realizada durante a resolução de um exercício de programação pré-definido pelo pesquisador;

2 – Análise do artefato desenvolvido: correção do programa de computador gerado pelo estudante durante a etapa de coleta de dados. Nesta etapa é atribuída uma nota de 0 a 100 de acordo com o desenvolvimento do programa e identificado se existem erros na solução entregue;

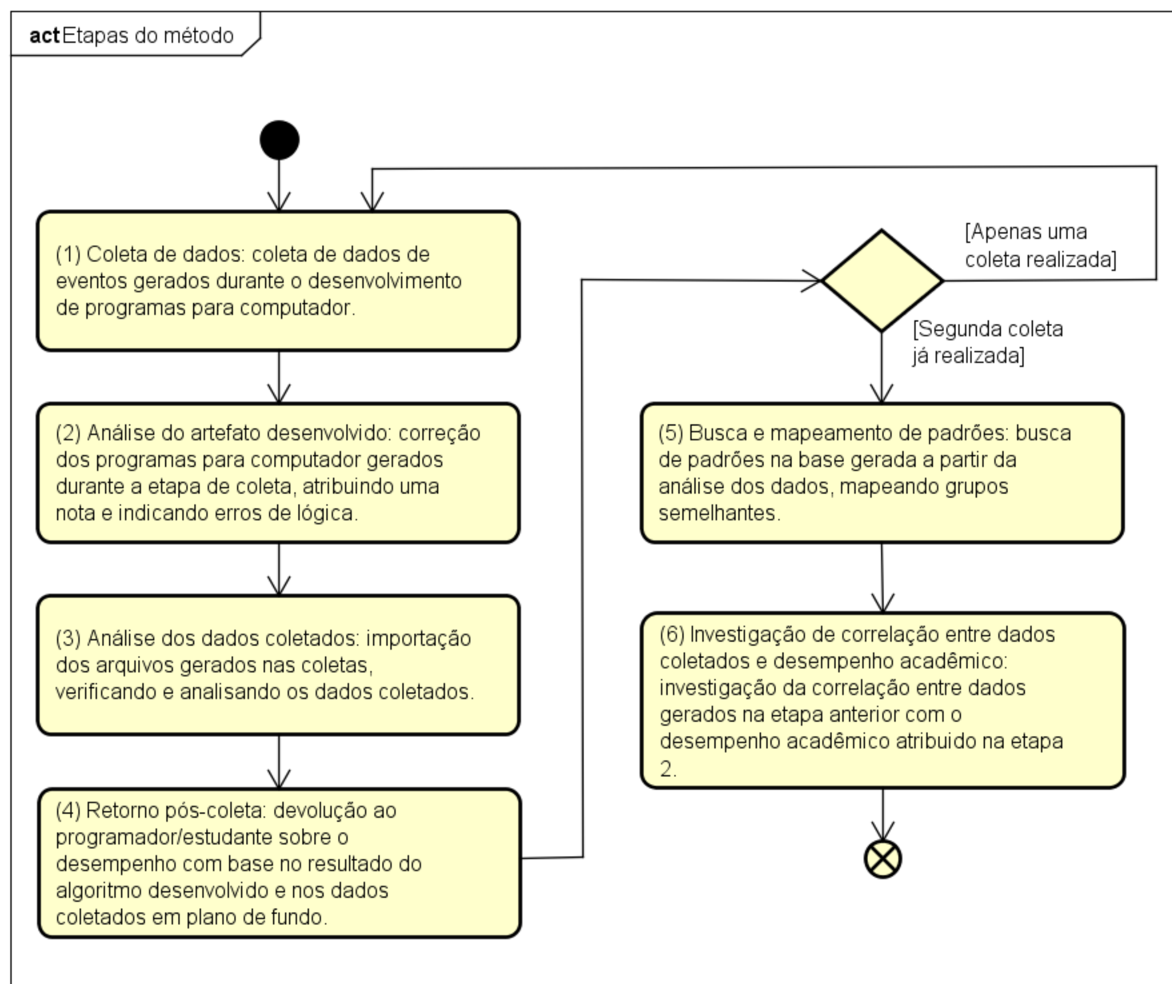
3 – Análise dos dados coletados: importação do arquivo de coleta para a ferramenta de análise, gerando o relatório consolidado de análise para auxílio ao instrutor/professor e retorno ao programador/estudante;

4 – Retorno pós-coleta: devolutiva ao programador/estudante sobre o desempenho durante o desenvolvimento do programa para computador, composto pela nota atribuída ao exercício e pelas informações geradas pela ferramenta de análise;

5 – Busca e mapeamento de padrões: busca por padrões entre os programadores/estudantes na base gerada a partir da análise dos dados, mapeando possíveis grupos;

6 – Investigação de correlação entre dados coletados e desempenho acadêmico: teste de correlação entre os dados coletados e o desempenho acadêmico atribuído ao exercício.

FIGURA 1 – ETAPAS DE COLETA E ANÁLISE




FONTE: O autor (2018).

De acordo com o método proposto, para cada grupo de alunos que participarem do experimento devem ser realizadas 2 coletas de dados, sendo que a segunda é realizada após o retorno aos estudantes com informações coletadas em plano de fundo. A realização de 2 coletas, em um primeiro momento, apresenta-se como necessária para a comparação entre as coletas e verificação da evolução dos estudantes. Entretanto, nada obsta que sejam realizadas mais do que 2 coletas para o mesmo grupo de alunos.

### 3.1.1 Coleta de Dados

Na primeira etapa do método proposto é realizada a coleta de dados oriundos de eventos gerados durante a programação para computadores. Os eventos a serem monitorados foram escolhidos de acordo com os dados a serem coletados e, por sua vez, a escolha dos dados coletados baseou-se no resultado da revisão bibliográfica relatada no capítulo anterior. Nesta revisão observa-se que, dos 14 trabalhos relatados, 8 coletam informações a respeito dos eventos de compilação e possíveis erros de compilação. Ainda, o mapeamento dos erros cometidos pelos estudantes durante o desenvolvimento do programa pode indicar a forma como o mesmo evolui durante o desenvolvimento de programas para computador, informação esta que não é possível encontrar apenas no artefato final gerado pelo estudante.

Além da especificidade do erro (tipo, linha de código e posição) informações como a quantidade de compilações, compilações com falha, compilações com sucesso e erros repetidos entre as compilações, podem indicar a forma de comportamento do aluno durante o desenvolvimento de programas.

Levando em conta os objetivos do trabalho e a análise dos trabalhos correlatos, foi optado por não utilizar ferramenta de *keylogger* como forma de coleta, proposto inicialmente conforme relatado no Apêndice B. Para os eventos observados e as análises propostas a coleta com *keylogger* não seria viável.

Desta forma, os eventos e os dados coletados estão listados no Quadro 7 para melhor visualização.

QUADRO 7 – MAPEAMENTO DE EVENTOS E DADOS COLETADOS

| Evento | Dados Coletados | Observações |
|---|---|---|
| Inicialização da IDE | Data e hora do evento. | Inicializa o monitoramento de todos os eventos. |
| Criação de um projeto | Nome do projeto;<br>Data e hora do evento. | Permite acompanhar o projeto em que o estudante está trabalhando. |
| Ativação de um projeto | Nome do projeto;<br>Data e hora do evento. | Permite verificar se o estudante utilizou outros projetos para referência. |
| Salvar arquivo | Nome do arquivo;<br>Número de caracteres;<br>Número de linhas;<br>Conteúdo do arquivo;<br>Data e hora do evento. | Permite acompanhar a evolução do projeto a cada ação de salvar o projeto/arquivo que está sendo trabalhado. |
| Compilação | Nome do projeto;<br>Nome do arquivo;<br>Número de caracteres; | Permite acompanhar cada compilação, mapeando o status da compilação e os erros gerados a cada evento. |

| Evento | Dados Coletados | Observações |
|---|---|---|
| | Número de linhas;<br>Conteúdo do arquivo;<br>Log de compilação;<br>Status da compilação;<br>Quantidade de erros;<br>Quantidade de mensagens;<br>Data e hora do evento. | |
| Colar texto no arquivo | Texto colado;<br>Posição de inserção do texto;<br>Nome do arquivo;<br>Número de caracteres;<br>Número de linhas;<br>Data e hora do evento. | Permite verificar reutilização de blocos de código, seja de origem do próprio programa que está em desenvolvimento ou de origem externa. |
| Encerramento da IDE | Data e hora do evento. | Encerra o monitoramento dos eventos e armazena os dados em arquivo XML. |

FONTE: O autor (2018).

### 3.1.2 Análise do Artefato Desenvolvido

Na etapa de análise do artefato desenvolvido, cada artefato entregue deve ser corrigido com o intuito de verificar se o que foi desenvolvido atende ao que foi solicitado e identificar possíveis erros presentes no artefato entregue. No processo de análise do artefato uma nota deve ser atribuída, permitindo desta forma testar a correlação dos dados coletados com o desempenho acadêmico do aluno no exercício realizado.

Para padronizar a atribuição da nota de desempenho acadêmico entre distintos exercícios, foi estabelecido uma série de critérios e pontuações para a correção dos artefatos. Estes critérios e pontuações estão descritas no Quadro 8.

QUADRO 8 – CRITÉRIOS E PONTUAÇÃO DOS EXERCÍCIOS

| Descrição | Pontuação |
|---|---|
| Artefato final atende ao solicitado utilizando os recursos corretos | 100 |
| Artefato final atende ao solicitado, porém não utiliza os recursos corretos | 80 |
| Artefato final atende parcialmente ao que foi solicitado apresentando poucos erros (possível corrigir com até 3 alterações) | 60 |
| Artefato final apresenta alguns erros (possível corrigir com até 6 alterações) | 40 |
| Artefato final apresenta muitos erros (necessário mais do que 6 alterações para correção) | 20 |
| Artefato final em desacordo com o que foi solicitado | 0 |

FONTE: O autor (2018).

Caso o artefato entregue não atenda ao que foi solicitado, o método prevê que seja atribuído ao artefato uma indicação de qual tipo de erro está presente no

artefato entregue, sendo levados em consideração erros considerados como erros de lógica para desenvolvimento da solução. Entende-se como erro de lógica o erro que impede que o programa realize o que foi solicitado devido à implementação incorreta da solução. As classificações de erros de lógica utilizados neste trabalho estão listadas no Quadro 9 e foram escolhidas com base na experiência do autor do trabalho, podendo ser ampliadas conforme necessidade futura.

QUADRO 9 – CLASSIFICAÇÃO DE ERROS DE LÓGICA

| Classificação | Descrição |
|---|---|
| Erro de lógica | O artefato entregue está em desacordo com o que foi solicitado de forma geral. |
| Erro com operador lógico | O artefato entregue apresenta erro na utilização de operador lógico, por exemplo, operador “E” ou “OU”. |
| Erro com operador relacional | O artefato entregue apresenta erro na utilização de operador relacional, por exemplo, sinal de igualdade ou diferença. |
| Erro de ordem/sequência de comandos | O artefato entregue apresenta erro na ordem ou na sequência dos comandos utilizados para a construção da resolução do exercício. |
| Erro aritmético | O artefato entregue apresenta erro nas operações aritméticas necessárias para a resolução do exercício. |
| Erro com estruturas de repetição | O artefato entregue apresenta erro na utilização das estruturas de repetição necessárias para a resolução do exercício. |
| Erro com estruturas de decisão | O artefato entregue apresenta erro na utilização das estruturas de decisão necessárias para a resolução do exercício. |

FONTE: O autor (2018).

É possível que um artefato seja classificado com mais do que um tipo de erro de lógica, assim como é possível que um artefato seja classificado apenas com o tipo “Erro de lógica” pois a solução entregue está em desacordo de forma geral, não sendo possível classificar em uma outra categoria.

### 3.1.3 Análise dos Dados Coletados

Esta etapa do método tem como objetivo prover ao instrutor/professor uma visão dos dados coletados a partir de eventos gerados durante o desenvolvimento do exercício. Os dados são gerados através dos eventos mapeados (ver o Quadro 7) e podem fornecer informações auxiliares ao instrutor/docente de como o indivíduo desenvolveu o programa solicitado.

Nesta etapa, além dos dados coletados via ferramenta de coleta, é possível atribuir também uma classificação aos erros de compilação cometidos pelo

indivíduo, possibilitando, desta forma, agrupar os erros de compilação por tipo. Nesta classificação é importante que seja levado em conta o código-fonte do programa no momento da compilação e não apenas a mensagem gerada pelo compilador. Os tipos utilizados pelo método estão listados no Quadro 10.

QUADRO 10 – CLASSIFICAÇÃO DE ERROS DE COMPILAÇÃO

| Classificação | Descrição |
|---|---|
| Ausência ou sobra de finalizador (;) | Erro gerado pela ausência do finalizar de comando (;) ou utilização deste em local incorreto. |
| Ausência ou sobra de parênteses, chaves ou colchetes | Erro gerado pela ausência de separadores, por exemplo, parênteses, chaves ou colchetes, ou pela utilização destes em local incorreto. |
| Comando escrito incorretamente (sintaxe incorreta ou parâmetros incorretos) | Erro gerado pela sintaxe incorreta de comandos da linguagem de programação ou pela utilização incorreta de parâmetros. |
| Variável utilizada sem estar declarada | Erro gerado pelo uso de variável sem declará-la previamente no programa. |
| Variável declarada incorretamente ou valor inicial atribuído incorretamente | Erro gerado pela declaração incorreta da variável ou pela atribuição incorreta de valores na declaração desta. |
| Variável declarada, porém, utilizada com o nome escrito incorretamente | Erro gerado pela utilização de variável com nome escrito incorretamente. |
| Tipo de dados incorreto ou inexistente | Erro gerado pela utilização de tipo de dados incorretos, na atribuição de valores às variáveis, ou inexistentes na declaração das variáveis. |
| Comando/expressão incompleta (else sem if, while sem do, falta de parâmetros) | Erro gerado pela utilização incompleta de comandos, por exemplo, utilização do else sem if precedente ou utilização do for sem parâmetros corretos. |
| Conectivo de comparação incorreto | Erro gerado pela utilização de conectivos de comparação incorretos, por exemplo, “=<” em vez de “<=”. |
| Atribuição de valores incorreta | Erro gerado pela atribuição incorreta de valores, por exemplo, “a + b = c” em vez de “c = a + b”. |

FONTE: O autor (2018).

Os erros foram escolhidos com base na revisão da literatura realizada e também levando em conta a experiência do pesquisador. Novas classificações podem ser utilizadas de acordo com a necessidade futura.

### 3.1.4 Retorno Pós-Coleta

Após a realização da coleta, análise do artefato final e análise dos dados coletados, o método prevê que seja gerado um relatório individual para os alunos. O relatório individual é composto pelas informações a seguir:

a) Nota: nota atribuída ao exercício conforme a tabela de pontuação utilizada para referência da correção (ver o Quadro 8);

b) Tempo de desenvolvimento: horário de início do desenvolvimento, horário de término do desenvolvimento do exercício e o tempo total gasto;

c) Compilações e execuções: compilações e execuções realizadas durante o desenvolvimento do exercício, indicando o resultado e a quantidade de erros de compilação, quando for o caso;

d) Resumo de compilações e execuções: resumo quantitativo referente as compilações e execuções realizadas, indicando o total por tipo (compilação e execução) e por resultado (sem efeito, sucesso e falha), média de erros de compilação, média de tempo entre as compilações e a indicação se a última compilação foi realizada com sucesso;

e) Erros de compilação: erros de compilação encontrados durante o desenvolvimento, indicando a mensagem de erro, o código do programa, a classificação do erro e se o mesmo é repetido, ou seja, se o mesmo (mesmo erro na mesma linha) já ocorreu anteriormente;

f) Resumo de erros de compilação: quadro resumo com os erros de compilação de acordo com a classificação, indicando a quantidade e o percentual de vezes que este erro ocorreu;

g) Erros de lógica: indicação a respeito dos erros de lógica encontrados no artefato entregue;

h) Programa: código-fonte do artefato entregue para correção.

### 3.1.5 Busca e Mapeamento de Padrões

Nesta etapa do método deve ser realizado o agrupamento dos dados coletados de forma individual, mapeando possíveis padrões entre o grupo de indivíduos participantes da coleta. Esta busca e mapeamento de padrões gera informações que possam auxiliar o instrutor/docente no planejamento das instruções como um todo, fornecendo informações auxiliares do desempenho do grupo de programadores/alunos.

O agrupamento dos dados coletados previsto pelo método está dividido em 4 grupos de informações:

Dados de execuções e compilações – agrupamento das informações por: a) número de execuções realizadas; b) número de compilações realizadas; c) número de compilações com sucesso; d) número de compilações com falha; e) número de

compilações sem efeito; f) média de erros de compilações; g) quantidade total de erros de compilações; h) número de erros repetidos entre as compilações; e i) número de coletas com a última compilação com sucesso.

Classificação dos erros de compilação – agrupamento dos números de erros de compilação encontrados nas análises individuais, classificados por tipo de erro (ver o Quadro 10).

Notas atribuídas ao artefato – agrupamento das notas (ver o Quadro 8) de desempenho acadêmico atribuídas aos artefatos entregues.

Classificação de erro no artefato – agrupamento da classificação dos erros encontrados nos artefatos entregues por tipo de erro lógico (ver o Quadro 9) encontrado.

O agrupamento destas informações fornece também ao instrutor/docente parâmetros para verificar a evolução de determinado grupo de alunos em coletas distintas, acompanhando assim a evolução deste grupo.

### 3.1.6 Investigação de Correlação com Desempenho Acadêmico

Para investigar uma possível correlação entre o desempenho acadêmico atribuído aos exercícios realizados com os dados coletados durante a realização do exercício, o método prevê a utilização do cálculo do coeficiente de correlação que, conforme Larson e Farber (2004), é uma medida do grau e da direção de uma relação linear entre duas variáveis, *x* e *y*, onde *x* é a variável independente ou variável explanatória e *y* a variável dependente ou resposta. O coeficiente de correlação amostral (símbolo *r*) é calculado através da fórmula:

$$r=\frac{n\sum xy-\left(\sum x\right)\left(\sum y\right)}{\sqrt{n\sum x^2-\left(\sum x\right)^2}\sqrt{n\sum y^2-\left(\sum y\right)^2}}$$

O intervalo de variação do coeficiente de correlação vai de -1 a 1. Se *x* e *y* tiverem forte correlação linear positiva, *r* estará próximo de 1. Se *x* e *y* tiverem forte correlação negativa (inversamente proporcional), *r* estará próximo de -1. Se não existir correlação linear ou ainda se a correlação linear for fraca, *r* estará próximo de 0 (LARSON e FARBER, 2004).

Com base no valor do coeficiente de correlação calculado é possível indicar o grau de correlação entre as variáveis. No Quadro 11 estão listadas as faixas de correlação conforme a interpretação utilizada neste trabalho.

QUADRO 11 – INTERPRETAÇÃO DO COEFICIENTE DE CORRELAÇÃO

| Faixa | Interpretação |
|---|---|
| De 0,00 até 0,20 | Correlação muito fraca |
| De 0,20 até 0,40 | Correlação fraca |
| De 0,40 até 0,60 | Correlação moderada |
| De 0,60 até 0,80 | Correlação forte |
| De 0,80 até 1,00 | Correlação muito forte |

FONTE: O autor (2018).

Também é possível investigar correlação entre duas variáveis utilizando um gráfico chamado mapa de dispersão. Conforme Larson e Farber (2004) um mapa de dispersão pode ser usado para determinar se existe uma correlação linear (uma reta) entre duas variáveis. Em um mapa de dispersão, os pares ordenados ($x$, $y$) representam pontos em um plano coordenado. A variável independente $x$ é medida sobre o eixo horizontal, enquanto a variável dependente $y$ é medida sobre o eixo vertical.

É proposto no método que, inicialmente, seja investigada a correlação entre alguns dos dados coletados em plano de fundo durante a solução dos exercícios, como variável independente, e o desempenho acadêmico atribuído ao exercício, como variável dependente. Os dados escolhidos para investigar a correlação são: a) número de compilações com falha; b) número de compilações com sucesso; c) média de erros por compilação); e d) número total de erros.

## 3.2 FERRAMENTAS DESENVOLVIDAS

Para que fosse possível realizar a coleta dos eventos gerados durante a programação para computadores e a análise dos dados coletados, suportando o método proposto, duas ferramentas foram concebidas e desenvolvidas, uma ferramenta de coleta na forma de *plugin* integrada a um Ambiente Integrado de Desenvolvimento (*Integrated Development Environment* – IDE) e uma ferramenta de consolidação e análise de dados.

A escolha por utilizar ferramentas distintas, ou seja, de forma modular, justifica-se visando a coleta dos dados em IDEs diferentes, possibilitando o reuso do

método e da ferramenta de análise desenvolvida nesta pesquisa com outras linguagens ou, ainda, por outros pesquisadores.

Desta forma, o sistema de coleta e análise de eventos gerados durante a programação para computadores é composto por 2 módulos. O primeiro módulo é o de coleta de dados, um *plugin* desenvolvido para a ferramenta Code::Blocks[4], ferramenta utilizada pelos estudantes que participaram dos experimentos. O segundo módulo é o de consolidação e análise de dados, desenvolvido como uma página web, utilizando a linguagem PHP e banco de dados MySQL.

As ferramentas de coleta e análise estão descritas nas seções a seguir.

## 3.3 FERRAMENTA DE COLETA

Para a coleta de dados dos eventos gerados durante o desenvolvimento de programas para computador optou-se por utilizar uma ferramenta integrada ao Ambiente Integrado de Desenvolvimento (*Integrated Development Environment* – IDE) na forma de *plugin*.

A IDE escolhida para ser utilizada nos experimentos desta pesquisa é a IDE Code::Blocks, uma IDE *open source*, voltada para o desenvolvimento de programas nas linguagens C, C++ e Fortran. A escolha do Code::Blocks para o experimento baseia-se no fato desta IDE ser amplamente utilizada como ambiente de desenvolvimento para o ensino de programação e também pelo fato desta IDE ser a utilizada pelos alunos envolvidos nos experimentos que serão relatados no próximo capítulo.

A IDE Code::Blocks permite o desenvolvimento de *plugins* para o seu ambiente, utilizando a linguagem de programação C++. Pela característica de desenvolvimento e manutenção da IDE (*open source*), a documentação está espalhada em fóruns mantidos pela comunidade desenvolvedora e usuária da ferramenta. Da mesma forma, os *plugins* desenvolvidos pela comunidade e disponibilizados junto com a ferramenta estão disponíveis para serem utilizados como modelo.

Para permitir a coleta de dados de forma transparente, o *plugin* desenvolvido neste trabalho foi programado para ser ativado automaticamente no momento da abertura da IDE e, a partir de então, os eventos descritos no método (vide Quadro 7)

---

[4] http://www.codeblocks.org/

são monitorados e os dados são coletados. Quando a IDE é encerrada estes dados são consolidados em um arquivo no formato *eXtensible Markup Language* (XML) que será utilizado para importação na ferramenta de análise.

### 3.3.1 Arquivo de Exportação

Para armazenar os dados coletados durante o desenvolvimento do programa para computador pelos alunos, permitindo a posterior importação para o sistema de análise, foi escolhido o formato XML. O arquivo gerado funciona como um repositório intermediário entre o *plugin* de coleta e o sistema de consolidação e análise.

A forma e o formato escolhido permitem que novos programas de coleta sejam desenvolvidos, visando a reutilização deste método em outras IDEs. Sendo assim o padrão do arquivo XML será relatado a seguir.

Ao iniciar a IDE é gerado o nó principal do arquivo XML, chamado “CapturaEventos”, registrando a data e hora de inicialização da IDE, conforme ilustrado na Figura 2.

FIGURA 2 – XML – CAPTURA EVENTOS

```xml
<?xml version="1.0"?>
<CapturaEventos TimeStamp="2018-01-18 15:14:58">
```

FONTE: O autor (2018).

Ao criar um novo projeto são gerados 2 eventos, o evento “ProjectActivated” e o “NewProject”. É necessário registrar os dois eventos devido ao comportamento da IDE Code::Blocks, pois apenas utilizando o evento “NewProject” o nome do projeto ainda não está registrado.

Caso o evento “ProjectActivated” aparece sozinho, significa que um projeto já existente foi aberto na IDE. Os eventos estão ilustrados na Figura 3.

FIGURA 3 – XML – EVENTOS DE PROJETO

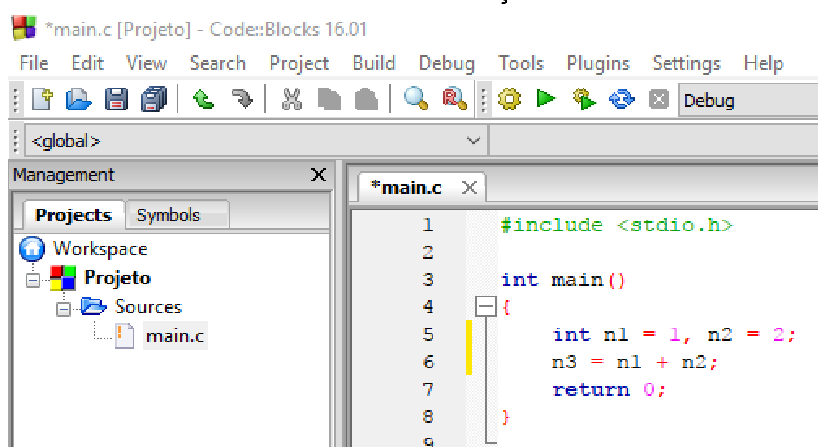
```xml
<ProjectEvent TimeStamp="2018-01-18 15:15:16" Type="ProjectActivated">
    <ProjectName>Projeto</ProjectName>
</ProjectEvent>
<ProjectEvent TimeStamp="2018-01-18 15:15:16" Type="NewProject">
    <ProjectName>ND</ProjectName>
</ProjectEvent>
```

FONTE: O autor (2018).

Ao salvar um arquivo o evento “FileSaved” é gerado. Neste evento são registrados o nome do arquivo, o número de caracteres, o número de linhas e o

código do programa. A ação de salvar ilustrada na Figura 4 gerou o evento XML ilustrado na Figura 5.

FIGURA 4 – TELA – AÇÃO DE SALVAR




FONTE: O autor (2018).

FIGURA 5 – XML – CÓDIGO SALVO

```
<FileEvent TimeStamp="2018-01-18 15:16:18" Type="FileSaved">
    <FileName>C:\Temp\Projeto\main.c</FileName>
    <FileLength>92</FileLength>
    <FileLines>9</FileLines>
  - <CodeSnapshot>
      - <![CDATA[
          #include <stdio.h>

          int main()
          {
             int n1 = 1, n2 = 2;

             n3 = n1 + n2;
             return 0;
          }
        ]]>
    </CodeSnapshot>
</FileEvent>
```

FONTE: O autor (2018).

Quando a compilação é executada, dois eventos são registrados o “BuildStart” e o “BuildFinished”. O “BuildStart” registra apenas o nome do projeto e o momento em que a compilação foi iniciada. Por sua vez, o “BuildFinishd” registra o nome do projeto, o nome do arquivo, a quantidade de caracteres, a quantidade de linhas, o código do programa, as mensagens geradas pelo compilador, o status da compilação, a quantidade de erros, a quantidade de mensagens e os detalhes (tipo de mensagem, linha e código-fonte) de cada mensagem.

O registro da compilação ilustrada na Figura 6 pode ser verificada nas Figuras 7, 8 e 9.

FIGURA 6 – TELA – COMPILAÇÃO DO PROGRAMA

FONTE: O autor (2018).

FIGURA 7 – XML – EVENTO DE COMPILAÇÃO 1


```
<BuildEvent TimeStamp="2018-01-18 15:16:21" Type="BuildStarted">
    <ProjectName>Projeto</ProjectName>
</BuildEvent>
<BuildEvent TimeStamp="2018-01-18 15:16:23" Type="BuildFinished">
    <ProjectName>Projeto</ProjectName>
    <FileName>C:\Temp\Projeto\main.c</FileName>
    <FileLength>92</FileLength>
    <FileLines>9</FileLines>
  - <CodeSnapshot>
    - <![CDATA[
          #include <stdio.h>

          int main()
          {
            int n1 = 1, n2 = 2;

            n3 = n1 + n2;
            return 0;
          }
      ]]>
    </CodeSnapshot>
```


FONTE: O autor (2018).

FIGURA 8 – XML – EVENTO DE COMPILAÇÃO 2


```
<BuildLog>
  - <![CDATA[
        ||=== Build: Debug in Projeto (compiler: GNU GCC Compiler) ===|

        C:\Temp\Projeto\main.c||In function 'main':|

        C:\Temp\Projeto\main.c|6|error: 'n3' undeclared (first use in this function)|

        C:\Temp\Projeto\main.c|6|note: each undeclared identifier is reported only once for each function it appears in|

        ||=== Build failed: 1 error(s), 0 warning(s) (0 minute(s), 1 second(s)) ===|

    ]]>
</BuildLog>
```


FONTE: O autor (2018).

FIGURA 9 – XML – EVENTO DE COMPILAÇÃO 3


```
    <BuildStatus>Failed</BuildStatus>
    <BuildErrors>1</BuildErrors>
    <BuildMessagesNumber>5</BuildMessagesNumber>
  - <BuildMessages>
    - <BuildMessage Number="0">
        <File/>
        <Line/>
        <Message>=== Build: Debug in Projeto (compiler: GNU GCC Compiler) ===</Message>
      </BuildMessage>
    - <BuildMessage Number="1">
        <File>C:\Temp\Projeto\main.c</File>
        <Line/>
        <Message>In function 'main':</Message>
      </BuildMessage>
    - <BuildMessage Number="2">
        <File>C:\Temp\Projeto\main.c</File>
        <Line>6</Line>
        <Message>error: 'n3' undeclared (first use in this function)</Message>
        <Code> n3 = n1 + n2; </Code>
      </BuildMessage>
    - <BuildMessage Number="3">
        <File>C:\Temp\Projeto\main.c</File>
        <Line>6</Line>
        <Message>note: each undeclared identifier is reported only once for each function it appears in</Message>
        <Code> n3 = n1 + n2; </Code>
      </BuildMessage>
    - <BuildMessage Number="4">
        <File/>
        <Line/>
        <Message>=== Build failed: 1 error(s), 0 warning(s) (0 minute(s), 1 second(s)) ===</Message>
      </BuildMessage>
    </BuildMessages>
</BuildEvent>
```


FONTE: O autor (2018).

FIGURA 10 – XML – EVENTO DE COMPILAÇÃO 4


```
<BuildEvent TimeStamp="2018-01-18 15:17:07" Type="BuildStarted">
    <ProjectName>Projeto</ProjectName>
</BuildEvent>
<BuildEvent TimeStamp="2018-01-18 15:17:10" Type="BuildFinished">
    <ProjectName>Projeto</ProjectName>
    <FileName>C:\Temp\Projeto\main.c</FileName>
    <FileLength>96</FileLength>
    <FileLines>9</FileLines>
  - <CodeSnapshot>
    - <![CDATA[
        #include <stdio.h>

        int main()
        {
           int n1 = 1, n2 = 2, n3;

           n3 = n1 + n2;
           return 0;
        }
      ]]>
    </CodeSnapshot>
  - <BuildLog>
    - <![CDATA[
        ||=== Build: Debug in Projeto (compiler: GNU GCC Compiler) ===|

        C:\Temp\Projeto\main.c||In function 'main':|

        C:\Temp\Projeto\main.c|5|warning: variable 'n3' set but not used [-Wunused-but-set-variable]|

        ||=== Build finished: 0 error(s), 1 warning(s) (0 minute(s), 3 second(s)) ===|

      ]]>
    </BuildLog>
    <BuildStatus>Finished</BuildStatus>
    <BuildErrors>0</BuildErrors>
</BuildEvent>
```


FONTE: O autor (2018).

Quando o resultado da compilação não apresenta erros, não houve alterações no código para uma nova compilação ou é apenas uma execução, o “BuildStatus” é registrado respectivamente como “Finished”, “Run” e “Nothing”. Nestas situações as mensagens não são salvas. Um exemplo pode ser verificado na Figura 10.

Caso um trecho de código seja colado no programa, seja ele proveniente do mesmo programa ou de uma fonte externa, o evento “PasteEvent” é registrado. Neste evento são registrados o texto colado, a posição em caracteres onde ele foi colado, o nome do arquivo, a quantidade de caracteres do arquivo e o número de linhas. Um exemplo está ilustrado na Figura 11.

FIGURA 11 – XML – EVENTO DE COLAR TEXTO

```
<PasteEvent TimeStamp="2018-01-18 15:20:00">
    <TextPasted>printf ("%d", n3);</TextPasted>
    <PastePosition>109</PastePosition>
    <FileName>C:\Temp\Projeto\main.c</FileName>
    <FileLength>144</FileLength>
    <FileLines>11</FileLines>
</PasteEvent>
```

FONTE: O autor (2018).

Ao encerrar a IDE o evento “IDEClosed” é gerado, registrando a data e a hora em que isso ocorre. Neste momento o arquivo XML é salvo. O evento está ilustrado na Figura 12.

FIGURA 12 – XML – EVENTO DE ENCERRAMENTO

```
    <IDEClosed TimeStamp="2018-01-18 15-20-17"/>
</CapturaEventos>
```

FONTE: O autor (2018).

Ao encerrar a IDE o arquivo XML é salvo no diretório “C:\Users\Public\Temp” do computador, ficando disponível para cópia e importação no sistema de análise.

### 3.3.2 Dificuldades encontradas

Algumas dificuldades foram encontradas para o desenvolvimento da ferramenta de coleta de dados. Estas dificuldades estão relatadas nesta subseção.

A principal dificuldade encontrada foi com a falta de documentação para o desenvolvimento de *plugins* para a IDE Code::Blocks. Apesar do código fonte dos *plugins* embutidos na ferramenta estarem disponíveis, pouca ou nenhuma

documentação é encontrada a respeito do desenvolvimento destes. Este fato dificulta um tanto o entendimento do código que está disponível.

Devido a característica do desenvolvimento de *plugins*, sendo estes desenvolvidos por diferentes pessoas, não há um padrão da forma de desenvolvimento. Este fato dificulta o entendimento quando se utilizam diferentes exemplos.

Para registrar o resultado da compilação foi necessário realizar a cópia das informações geradas pela IDE para a área de transferência do computador e programado o acesso e o tratamento do conteúdo pelo *plugin*. Este procedimento foi necessário devido à impossibilidade de acesso direto ao log gerado pela IDE.

Ainda, foi enfrentada uma dificuldade quanto à distribuição do *plugin*, pois para que o desenvolvimento fosse possível era necessário compilar uma nova versão da ferramenta na máquina onde o *plugin* foi desenvolvido e a versão do compilador utilizado deve ser a mesma que é distribuída com a ferramenta para o desenvolvimento.

Caso este detalhe não seja observado o *plugin* funcionará na máquina de desenvolvimento, porém não será possível utilizá-lo em outros computadores. Este detalhe parece simples, porém a falta de documentação a respeito o torna um empecilho, principalmente quando se utilizam máquinas com diferentes IDEs, com diferentes compiladores.

## 3.4 FERRAMENTA DE CONSOLIDAÇÃO E ANÁLISE

Para realizar a importação, consolidação e análise dos dados, uma segunda ferramenta foi desenvolvida. Esta ferramenta foi desenvolvida para importar os arquivos XML gerados durante o desenvolvimento de programas e, a partir desses dados, gerar informações sobre as atividades coletadas em plano de fundo.

Na ferramenta desenvolvida é possível importar o arquivo XML gerado vinculando-o a uma atividade e a um aluno específico. Para isso as atividades e os alunos devem estar previamente cadastrados na ferramenta.

A partir dos dados importados, o professor pode verificar um resumo das atividades geradas pelo aluno durante o desenvolvimento da atividade. Este resumo está disponível na ferramenta e será relatado na próxima subseção.

### 3.4.1 Resumo das Atividades Geradas

Para que o professor possa verificar um resumo das atividades geradas pelo estudante durante o desenvolvimento de uma atividade específica foi desenvolvida uma tela de consolidação dos dados. A partir desta tela do professor pode obter um panorama a respeito do comportamento do aluno durante o desenvolvimento da solução para a atividade aplicada.

Os dados são agrupados em seções na tela, sendo elas: a) Lista de Compilações/Execuções; b) Resumo de Compilações/Execuções; c) Erros em Compilações; d) Resumo de Erros em Compilações de acordo com a classificação de erros; e) Resumo de Erros em Compilações; f) Erros de Lógica no Artefato Entregue; g) Projetos Utilizados; h) Arquivos Salvos; i) Códigos Colados; e j) Programa Final.

Na seção de “Compilações e Execuções” é exibida uma lista com os eventos de compilação e execução disparadas pelo aluno. Nesta lista é possível verificar o momento de início e término da compilação, o intervalo entre as compilações, o tipo da compilação (compilação, execução ou sem efeito), a quantidade de erros gerados, a quantidade de mensagens geradas e o nome do projeto. Na Figura 13 está ilustrado um exemplo da lista de compilações/execuções.

FIGURA 13 – LISTA DE COMPILAÇÕES/EXECUÇÕES

| Resultado Individual - Experimento: Exercício Exemplo | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Aluno: Aluno Exemplo - Nota atribuída: | | | | | | | | |
| Início: 2018-01-18 15:15:16 - Término: 2018-01-18 15:18:08 - Tempo Gasto: 00:02:52 | | | | | | | | |
| Compilações/Execuções | | | | | | | | |
| Sequência | Início | Fim | Intervalo | Tipo | Resultado | Erros | Mensagens | Projeto |
| 1 | 2018-01-18 15:16:21 | 2018-01-18 15:16:23 | 00:01:05 | Compilação | Falha | 1 | 5 | Projeto |
| 2 | 2018-01-18 15:17:07 | 2018-01-18 15:17:10 | 00:00:44 | Compilação | Sucesso | 0 | 0 | Projeto |
| 3 | 2018-01-18 15:17:41 | 2018-01-18 15:17:44 | 00:00:31 | Compilação | Sucesso | 0 | 0 | Projeto |
| 4 | 2018-01-18 15:17:44 | 2018-01-18 15:17:52 | 00:00:00 | Execução | Execução | 0 | 0 | Projeto |

FONTE: O autor (2018).

Na sequência é exibido um resumo das compilações/execuções, no qual é possível observar a quantidade de compilações, a quantidade de execuções, a quantidade de tentativa de compilação sem efeito, a quantidade de compilações com sucesso, a quantidade de compilações com falha, a média de erros entre as compilações, o intervalo médio de tempo entre as compilações e qual é o resultado da última compilação realizada (Sucesso ou Falha). Na Figura 14 está ilustrado um exemplo do resumo das compilações/execuções.

FIGURA 14 – RESUMO DE COMPILAÇÕES/EXECUÇÕES

| Resumo de Compilações/Execuções | | | | | | | |
|---|---|---|---|---|---|---|---|
| Compilações | Execuções | Sem efeito | Sucesso | Falha | Média de Erros | Intervalo Médio | Última Compilação |
| 3 | 1 | 0 | 2 | 1 | 1.00 | 00:00:37 | Sucesso |

FONTE: O autor (2018).

Para as compilações que terminaram com erros o sistema lista um quadro relatando os erros em compilações. Na lista de "Erros em Compilações" é informada qual a compilação em que ocorreu o erro, o número da linha do arquivo que apresentou problema, a mensagem de erro gerada pelo compilador, o código do programa em que ocorreu o erro, a classificação do erro, e se o erro é repetido ou não, ou seja, se o mesmo erro já ocorreu em compilações anteriores. Na Figura 15 está ilustrado um exemplo da lista de erros em compilações.

FIGURA 15 – ERROS EM COMPILAÇÕES

| Erros em Compilações | | | | | | |
|---|---|---|---|---|---|---|
| Seq. Compilação | Linha | Arquivo | Mensagem | Código | Classificação | Repetido |
| 1 | 6 | C:\Temp\Projeto\main.c | error: 'n3' undeclared (first use in this function) | n3 = n1 + n2; | Variável utilizada sem estar declarada | Não |

FONTE: O autor (2018).

A partir da classificação de erros é apresentado um resumo dos erros de compilação. Neste quadro é apresentada a classificação atribuída ao erro, a quantidade de vezes que erros desta classificação foram repetidos e o percentual que esta quantidade de erros representa perante o total de erros de compilações. Um exemplo está ilustrado na Figura 16.

FIGURA 16 – RESUMO DE ERROS EM COMPILAÇÕES – CLASSIFICAÇÃO

| Resumo de Erros em Compilações - Classificação | | |
|---|---|---|
| Descrição | Quantidade | Percentual |
| Ausência ou sobra de finalizador (;) | 0 | 0.0 |
| Ausência ou sobra de parênteses, chaves ou colchetes | 0 | 0.0 |
| Comando escrito incorretamente (sintaxe incorreta ou parâmetros incorretos) | 0 | 0.0 |
| Variável utilizada sem estar declarada | 1 | 100.0 |
| Variável declarada incorretamente ou valor inicial assignado incorretamente | 0 | 0.0 |
| Variável declarada mas utilizada com o nome escrito incorretamente | 0 | 0.0 |
| Tipo de dados incorreto ou inexistente | 0 | 0.0 |
| Comando/expressão incompleta (else sem if, while sem do, falta de parâmetros) | 0 | 0.0 |
| Conectivo de comparação incorreto | 0 | 0.0 |
| Atribuição de valores incorreta | 0 | 0.0 |

FONTE: O autor (2018).

Ressalta-se que a classificação de erros é manual, realizada pelo professor, a partir da análise de cada erro e com auxílio do código-fonte do artefato gerado. A classificação utilizada teve base na literatura descrita no capítulo anterior e também na experiência do autor do trabalho. Entretanto, é possível que esta classificação

seja modificada de acordo com a necessidade, adicionando, alterando e excluindo classificações conforme necessidade.

Com base nos erros gerados nas compilações é também apresentado um quadro com o “Resumo de Erros em Compilações”. Neste quadro é apresentado o número total de erros em compilações, a quantidade de erros repetidos e o percentual de erros repetidos entre as compilações. Um exemplo está ilustrado na Figura 17.

FIGURA 17 – RESUMO DE ERROS EM COMPILAÇÕES

| Resumo de Erros em Compilações | | |
|---|---|---|
| Total | Erros Repetidos | % de Erros Repetidos |
| 1 | 0 | 0.0 |

FONTE: O autor (2018).

A partir do artefato entregue, é possível realizar uma classificação dos erros de lógica apresentados. Esta classificação permite ao professor avaliar também onde estão os possíveis problemas do aluno quanto à aplicação da lógica na construção dos programas para computador. Ressalta-se que a classificação é manual, realizada pelo professor, tendo como base o artefato final entregue pelo aluno.

Os erros utilizados nesta ferramenta foram escolhidos com base na experiência do autor deste trabalho e, assim como a classificação de erros de compilação, podem ser adicionados, alterados e excluídos conforme necessidade. O quadro de classificação dos erros de lógica está ilustrado na Figura 18.

FIGURA 18 – ERROS DE LÓGICA – ARTEFATO ENTREGUE

| Erros de Lógica - Artefato Entregue | |
|---|---|
| Descrição | Presente |
| Erro de Lógica | Não |
| Erro com operador lógico | Não |
| Erro com operador relacional | Não |
| Erro de ordem/sequência de comandos | Não |
| Erro aritmético | Não |
| Erro com estruturas repetição | Não |
| Erro com estruturas de decisão | Não |

FONTE: O autor (2018).

Para acompanhar os projetos e arquivos utilizados são apresentadas duas listas, uma com os projetos utilizados durante o monitoramento das atividades e outra lista com os eventos de salvar. Estas listas estão ilustradas nas Figuras 19 e 20.

FIGURA 19 – PROJETOS UTILIZADOS

| Projetos Utilizados | | |
|---|---|---|
| Nome | Ação | Horário |
| Projeto | ProjectActivated | 2018-01-18 15:15:16 |

FONTE: O autor (2018).

FIGURA 20 – ARQUIVOS SALVOS

| Arquivos Salvos | | | |
|---|---|---|---|
| Nome | Horário | Caracteres | Linhas |
| C:\Temp\Projeto\main.c | 2018-01-18 15:16:18 | 92 | 9 |
| C:\Temp\Projeto\main.c | 2018-01-18 15:17:00 | 96 | 9 |
| C:\Temp\Projeto\main.c | 2018-01-18 15:17:41 | 120 | 10 |

FONTE: O autor (2018).

Na sequência é apresentado um quadro com os eventos de inserção de código colado no programa. Neste quadro é listado o momento de inserção, a posição no arquivo em que o código foi inserido, o texto colado, o nome do arquivo, a quantidade de caracteres do arquivo após a inserção do código e o número de linhas do arquivo após a inserção do código. Este quadro é ilustrado na Figura 21.

FIGURA 21 – RESUMO DE CÓDIGOS COLADOS

| Códigos Colados | | | | | |
|---|---|---|---|---|---|
| Horário | Posição | Texto | Arquivo | Caracteres | Linhas |
| 2018-01-18 15:18:00 | 109 | printf ("%d", n3); | C:\Temp\Projeto\main.c | 144 | 11 |

FONTE: O autor (2018).

Finalmente, é apresentado o código final do programa gerado pelo aluno para que o professor possa ter acesso também via o sistema de análise. A apresentação do código final está ilustrada na Figura 22.

FIGURA 22 – ARTEFATO FINAL

Programa

```
#include <stdio.h>
int main()
{
int n1 = 1, n2 = 2, n3;
n3 = n1 + n2;
printf ("%d", n3);
return 0;
}
```

FONTE: O autor (2018).

### 3.4.2 Consolidação dos Dados do Experimento

Com o intuito de fornecer ao professor uma visão geral de como a turma se comportou durante um exercício específico, foram disponibilizadas na ferramenta de consolidação dos dados informações agrupadas da turma para o exercício

analisado. Estas informações permitem que o professor verifique pontos fortes e fracos da turma durante a resolução do exercício, possibilitando ajustes pontuais quanto às dificuldades da turma.

Para fornecer uma visão a respeito do comportamento de compilação da turma, foi disponibilizada na ferramenta um quadro com o resumo das compilações e execuções, no qual é possível verificar informações sobre: a) as execuções realizadas; b) as compilações executadas; c) as compilações com sucesso; d) as compilações com falha; e) as compilações sem efeito; f) a média de erros das compilações; g) a quantidade total de erros das compilações; h) a quantidade de erros repetidos entre as compilações; e i) a quantidade de alunos que tiveram sucesso na última compilação.

Neste quadro é possível observar o número de alunos com ocorrência no item indicado, o percentual dos alunos que realizaram o item perante o total dos alunos, a média da turma e a média para os alunos com ocorrência. O quadro com o resumo das compilações está ilustrado na Figura 23.

FIGURA 23 – RESUMO DAS COMPILAÇÕES DA TURMA/EXERCÍCIO

| Resultado - Compilações - Alunos: 35 | | | | |
|---|---|---|---|---|
| Descrição | Alunos com ocorrência | % Alunos com ocorrência | Média da turma | Média alunos com ocorrência |
| Execuções realizadas | 32 | 91.4 | 20.49 | 22.41 |
| Compilações executadas | 35 | 100.0 | 37.34 | 37.34 |
| Compilações com sucesso | 34 | 97.1 | 19.37 | 19.94 |
| Compilações com falha | 35 | 100.0 | 14.74 | 14.74 |
| Compilações sem efeito | 29 | 82.9 | 3.23 | 3.90 |
| Média de erros | 35 | - | 1.74 | 1.74 |
| Quantidade total de erros | 35 | - | 29.71 | 29.71 |
| Erros repetidos entre compilações | 30 | 85.7 | 15.63 | 18.23 |
| Última compilação com sucesso | 30 | 85.71 | - | - |

FONTE: O autor (2018).

Objetivando fornecer o panorama de erros de compilação da turma, foi criado um quadro com as informações consolidadas da classificação de erros de compilação. Neste quadro é possível observar para cada classificação de erro de compilação a quantidade de alunos que cometeram este erro ao menos uma vez, o percentual de alunos da turma que cometeram este erro, a média da turma de vezes em que erros desta classificação foram encontrados e a média para os alunos que cometeram o erro. O quadro com o resumo das classificações de erro está ilustrado na Figura 24.

FIGURA 24 – RESUMO DA CLASSIFICAÇÃO DE ERROS DE COMPILAÇÃO DA TURMA/EXERCÍCIO

| Classificação dos Erros de Compilação | | | | |
|---|---|---|---|---|
| **Descrição** | **Alunos com ocorrência** | **% alunos com ocorrência** | **Média da turma** | **Média alunos com ocorrência** |
| Ausência ou sobra de finalizador (;) | 28 | 80.0 | 6.63 | 8.29 |
| Ausência ou sobra de parênteses, chaves ou colchetes | 21 | 60.0 | 6.17 | 10.29 |
| Comando escrito incorretamente (sintaxe incorreta ou parâmetros incorretos) | 15 | 42.9 | 5.83 | 13.60 |
| Variável utilizada sem estar declarada | 27 | 77.1 | 4.91 | 6.37 |
| Variável declarada incorretamente ou valor inicial assignado incorretamente | 4 | 11.4 | 0.14 | 1.25 |
| Variável declarada, porém, utilizada com o nome escrito incorretamente | 0 | 0.0 | 0.00 | 0.00 |
| Tipo de dados incorreto ou inexistente | 0 | 0.0 | 0.00 | 0.00 |
| Comando/expressão incompleta (else sem if, while sem do, falta de parâmetros) | 9 | 25.7 | 0.91 | 3.56 |
| Conectivo de comparação incorreto | 2 | 5.7 | 0.20 | 3.50 |
| Atribuição de valores incorreta | 11 | 31.4 | 2.06 | 6.55 |

FONTE: O autor (2018).

A partir da atribuição da nota à solução entregue, com base nos critérios informados na subseção anterior, foi criado um quadro resumo a respeito do desempenho acadêmico no exercício. Neste quadro é possível observar a média da turma para o exercício, a quantidade de alunos por nota e o percentual da turma que atingiu a nota no exercício. O quadro está ilustrado na Figura 25.

FIGURA 25 – RESUMO DAS NOTAS DA TURMA/EXERCÍCIO

| Resumo de Notas - Média da turma: 39.43 | | |
|---|---|---|
| **Nota** | **Quantidade de alunos** | **% Turma** |
| Nota 100 | 1 | 2.9 |
| Nota 80 | 1 | 2.9 |
| Nota 60 | 7 | 20.0 |
| Nota 40 | 15 | 42.9 |
| Nota 20 | 9 | 25.7 |
| Nota 0 | 2 | 5.7 |

FONTE: O autor (2018).

Finalmente, também é possível observar o panorama da turma quanto aos erros lógicos do artefato entregue. No quadro ilustrado na Figura 26 é possível observar, para cada erro de lógica, a quantidade de alunos da turma que cometeram este erro no artefato final e também o percentual da turma.

FIGURA 26 – RESUMO DAS NOTAS DA TURMA/EXERCÍCIO

| Erro Lógico | | |
|---|---|---|
| **Descrição** | **Quantidade de alunos** | **% Turma** |
| Erro de Lógica | 33 | 94.3 |
| Erro com operador lógico | 1 | 2.9 |
| Erro com operador relacional | 0 | 0.0 |
| Erro de ordem/sequência de comandos | 9 | 25.7 |
| Erro aritmético | 33 | 94.3 |
| Erro com estruturas repetição | 19 | 54.3 |
| Erro com estruturas de decisão | 0 | 0.0 |

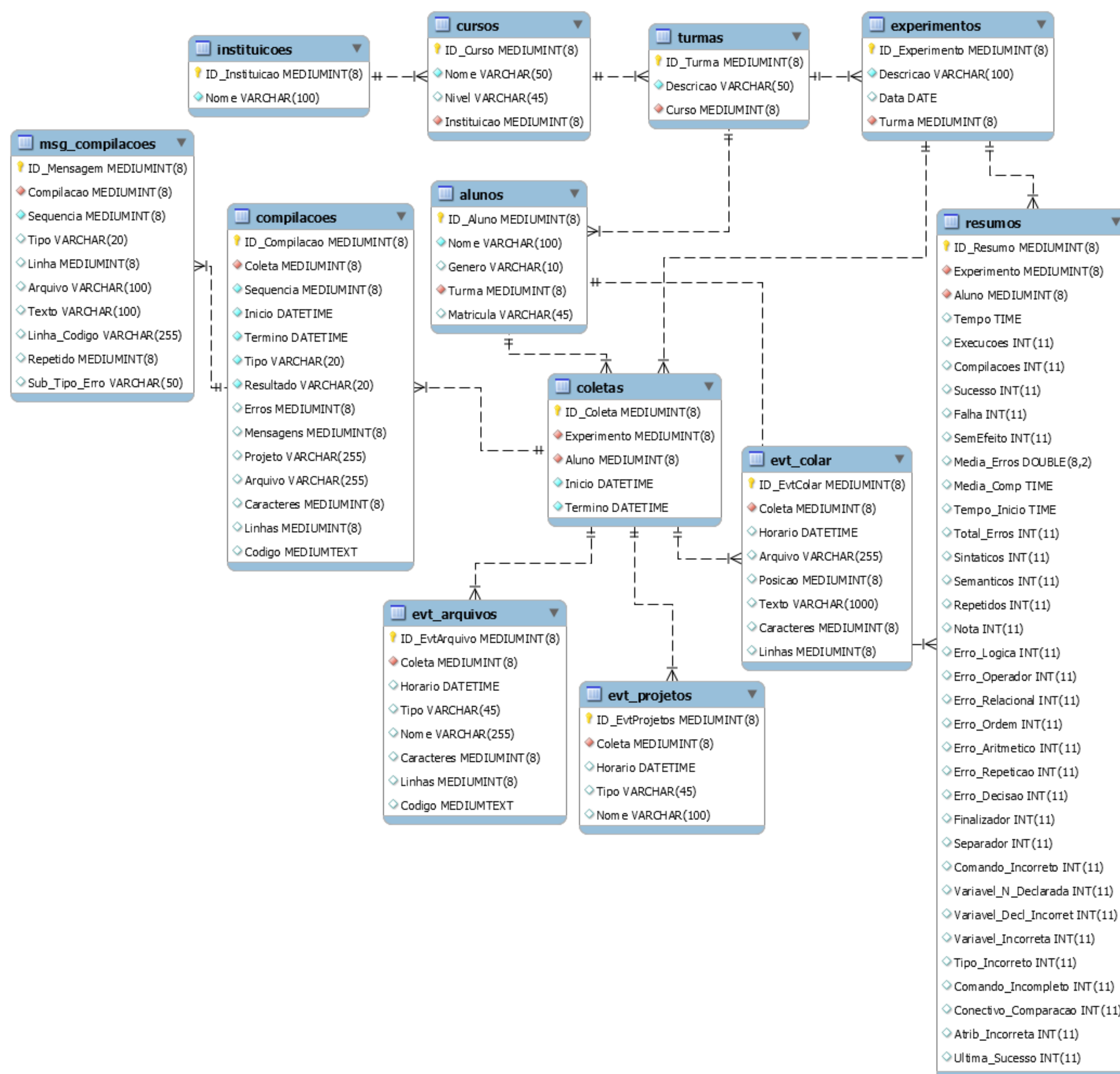
FONTE: O autor (2018).

### 3.4.3 Estrutura do Banco de Dados

A ferramenta de análise necessita armazenar os dados coletados em um banco de dados, consolidando as coletas para posterior análise. O Sistema Gerenciador de Banco de Dados escolhido para armazenar os dados é o MySQL.

Para armazenar os dados foi criado um banco de dados, que tem sua estrutura ilustrada na Figura 27.

FIGURA 27 – DIAGRAMA ENTIDADE RELACIONAMENTO




FONTE: O autor (2018).

As entidades “instituicoes”, “cursos” e “turmas” armazenam os dados de identificação das turmas que participaram das coletas de dados, relacionando com os cursos e instituições de ensino. Por sua vez, a entidade alunos armazena os

dados dos alunos participantes do experimento. Obrigatoriamente um aluno deve estar vinculado a uma turma.

Na entidade “experimentos” são armazenadas os dados de cada experimento realizado. Cada experimento deve estar vinculado a uma turma. O cadastro do experimento é necessário para o registro das coletas, sendo que os dados relativos as coletas são armazenados na entidade “coletas”, sendo cada coleta relacionada a um experimento e a um aluno.

Os eventos coletados são armazenados em entidades específicas. Os eventos de compilações, operações com arquivos, operações com projetos e colagem de código são armazenados, respectivamente, nas entidades “compilacoes”, “evt_arquivos”, “evt_projetos” e “evt_colar”. Ainda referente as compilações, cada mensagem originada nos eventos de compilação são armazenadas na entidade “msg_compilacoes”.

Por último, a entidade “resumos” é a responsável por armazenar as informações consolidadas das coletas relacionadas aos alunos participantes dos experimentos.

## 3.5 REFLEXÃO A RESPEITO DO MÉTODO E FERRAMENTAL PROPOSTO

Neste capítulo foram apresentados o método e ferramental propostos e utilizados nesta pesquisa. Este método e ferramentas podem ser utilizados como auxílio no processo de ensino-aprendizagem de programação, fornecendo ao professor dados a respeito do processo de desenvolvimento de programas para computador de cada aluno e também da turma, de forma consolidada.

O método apresentado visa estabelecer os parâmetros para coleta e análise dos dados oriundos de eventos gerados durante a programação para computadores, servindo como diretriz para este trabalho e também podendo ser utilizado em trabalhos futuros ou até por outros pesquisadores.

As ferramentas desenvolvidas e apresentadas neste capítulo foram concebidas para suportar o método e foram desenvolvidas em módulos. O primeiro módulo funciona como um *plugin* para IDE Code::Blocks, sendo ativado automaticamente quando a IDE é aberta e monitora eventos durante o processo de criação de programas, ao encerrar a IDE é gerado um arquivo no formato XML para ser importado no segundo módulo. O segundo módulo serve para importar e

consolidar os dados coletados pelo *plugin*, através deste módulo é possível visualizar os dados coletados durante o desenvolvimento de programas para computador, seja por aluno ou de forma consolidada, verificando o desempenho da turma.

Desta forma, com base no método e ferramental apresentados, o próximo capítulo relata os experimentos aplicados e o resultado oriundo destes.

# 4 EXPERIMENTOS REALIZADOS E RESULTADOS

Neste capítulo são relatados os experimentos realizados utilizando o método e as ferramentas relatadas no capítulo anterior. Foram realizados 5 experimentos com turmas de cursos relacionados com a área de Tecnologia da Informação e Engenharias, em nível médio e superior. Os dados sobre os experimentos estão resumidos no Quadro 12 e detalhados nas subseções a seguir.

QUADRO 12 – RESUMO DOS EXPERIMENTOS REALIZADOS

| # | Nível | Curso | Série | Disciplina | Tipo Instituição | Alunos |
|---|---|---|---|---|---|---|
| 1 | Médio | Técnico em Informática | 1ª | Lógica e Linguagem da Programação | Pública | 35 |
| 2 | Médio | Técnico em Informática | 1ª | Lógica e Linguagem da Programação | Pública | 35 |
| 3 | Superior | Sistemas de Informação | 1ª | Estrutura de Dados | Privada | 5 |
| 4 | Superior | Sistemas de Informação, Engenharia de Computação e Engenharia Eletrônica | 1ª | Fundamentos de Programação 1 | Pública | 14 |
| 5 | Superior | Engenharia Mecânica | 2ª | Computação 2 | Pública | 8 |

FONTE: O autor (2018).

Todos os experimentos foram realizados durante as aulas das disciplinas citadas no Quadro 12, em laboratórios de informática das instituições, sendo que cada aluno utilizou um computador de forma individual. Durante a realização das atividades os alunos foram orientados a não utilizar internet, mas puderam consultar seus materiais de apoio (cadernos e exercícios resolvidos anteriormente) e não puderam utilizar aparelho celular, *tablet* ou outros dispositivos.

Para cada grupo de alunos foram realizadas duas coletas, com tempo máximo de 51 minutos[5], sendo a segunda coleta após devolutiva das informações processadas a respeito da primeira coleta, conforme descrito nas etapas do método (ver a Figura 1). Apenas foram considerados para os experimentos os alunos que participaram das duas coletas. Sendo assim, ficaram de fora 3 alunos para o experimento 1, 5 alunos para o experimento 2, 4 alunos para o experimento 3, 9 alunos para o experimento 4 e 3 alunos para o experimento 5. A quantidade de alunos utilizada nos experimentos está apresentada no Quadro 12.

[5] Tempo de duração de uma aula das primeiras turmas utilizadas nos experimentos, servindo assim como referência de tempo máximo para as coletas.

Antes de iniciar as coletas foi esclarecido aos alunos o objetivo da pesquisa e que nenhum dado pessoal seria divulgado, assim como, os resultados seriam utilizados sempre de forma coletiva. Também foi informado que a participação era voluntária, sem apresentar prejuízo aos mesmos em caso de não participação. Desta forma, os dados utilizados e aqui apresentados são apenas dos alunos que concordaram em participar desta pesquisa.

## 4.1 EXERCÍCIOS UTILIZADOS

Para as coletas dos dados foi solicitado aos alunos que desenvolvessem um programa para computador solucionando determinados exercícios. O conhecimento necessário para construir a solução do exercício estava em linha com o que havia sido ensinado anteriormente durante as aulas das disciplinas apresentadas no Quadro 12.

Foram utilizados quatro exercícios distintos, aplicados às turmas de acordo com o conteúdo ministrado para cada uma delas até o momento da coleta. Os enunciados dos exercícios estão descritos a seguir.

### 4.1.1 Exercício 1 – Estruturas de repetição

O exercício 1 teve o intuito de verificar o aprendizado relacionado a estruturas de repetição e a substituição de valores entre variáveis. Sendo assim, foi solicitado aos alunos para que desenvolvessem um programa com o seguinte enunciado: “Escreva um programa na linguagem C que liste os 20 primeiros números da sequência de Fibonacci. A sequência deve ser gerada a partir de 2 números solicitados ao usuário. A sequência de Fibonacci é gerada com a soma dos 2 números anteriores. Por exemplo: Entrada: 1 e 2; Saída: 3, 5, 8, 13, 21…”

Este exercício foi utilizado com as turmas dos experimentos 1, 2 e 5 (ver o Quadro 12), sendo utilizado na primeira coleta de dados.

### 4.1.2 Exercício 2 – Vetores

O exercício 2 foi aplicado para verificar o aprendizado relacionado a vetores e à busca sequencial de valores nestes. Sendo assim, foi solicitado aos alunos que desenvolvessem um programa com o seguinte enunciado: “Escreva um programa

em C que solicite 10 números ao usuário. Armazene estes números em um vetor. Após armazenar os números, solicite ao usuário um número para ser encontrado no vetor. Caso encontre o número, o programa deve informar as posições em que o número está e a quantidade de vezes em que foi encontrado. Caso não encontre, o programa deve informar que não encontrou o número. Utilize estruturas de repetição."

Este exercício foi utilizado com as turmas dos experimentos 1, 2 e 5 (ver o Quadro 12), sendo utilizado na segunda coleta de dados.

### 4.1.3 Exercício 3 – Busca binária

O exercício 3 foi aplicado para verificar o aprendizado relacionado à busca binária. Sendo assim, foi solicitado aos alunos que desenvolvessem um programa com o seguinte enunciado: "Escreva um programa que realize uma busca binária de um determinado valor, informado pelo usuário, em um vetor ordenado com valores pré-definidos. O vetor é composto pelos 20 primeiros elementos da sequência de Fibonacci iniciada pelos números 1 e 2, sendo eles: 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6765 e 10946. O programa deve retornar as informações: - o valor informado pelo usuário foi encontrado? - em quantos passos de busca encontrou o valor foi encontrado? - em qual posição o valor está?"

Este exercício foi utilizado com as turmas dos experimentos 3 e 4 (ver o Quadro 12), sendo utilizado na primeira coleta de dados.

### 4.1.4 Exercício 4 – Ordenação

O exercício 4 foi aplicado para verificar o aprendizado relacionado à ordenação. Sendo assim, foi solicitado aos alunos que desenvolvessem um programa com o seguinte enunciado: "Escreva um programa que solicite 5 números ao usuário e armazene estes números em um vetor. Após ter armazenado todos os números o programa deve ordenar, em forma crescente, o vetor através da técnica de seleção. Ao final o programa deve imprimir os números de forma ordenada."

Este exercício foi utilizado com as turmas dos experimentos 3 e 4 (ver o Quadro 12), sendo utilizado na segunda coleta de dados.

## 4.2 CONSOLIDAÇÃO DE DADOS DO EXPERIMENTO

Com base nos dados coletados de cada aluno é possível realizar uma consolidação de dados da turma durante a realização do exercício proposto. Esta consolidação pode ser utilizada como informação auxiliar ao instrutor/docente para mapear as dificuldades da turma como um todo.

Esta consolidação é realizada pela ferramenta de consolidação e análise desenvolvida nesta pesquisa e, a partir desta, é gerado um relatório da turma. Este relatório foi enviado aos docentes participantes dos experimentos e os resultados serão detalhados em tabelas nas próximas subseções. Para uma melhor estruturação do texto, a explicação da estrutura das tabelas será realizada apenas uma vez, ficando para as próximas subseções apenas o relato dos resultados dos experimentos.

a) Tabela de resumo de compilações: nesta tabela são apresentados os dados consolidados das compilações e execuções realizadas no experimento. São considerados os números a respeito de: execuções realizadas; compilações realizadas; compilações com sucesso (ausência de erros de compilação); compilações com falha (presença de erro(s) de compilação); compilações sem efeito (compilação realizada sem necessidade, pois o código não sofreu alterações desde a última compilação); média de erros por compilação; quantidade total de erros de compilação; número de erros repetidos entre as compilações (mesmo erro na mesma linha de código); e quantidade de alunos que tiveram a última compilação com sucesso;

b) Tabela de classificação de erros: nesta tabela são apresentados os dados referente à classificação de erros de compilação encontrados durante a coleta do experimento. A classificação foi realizada manualmente, erro a erro, e revisada após a classificação inicial;

c) Tabela de erros de lógica: nesta tabela são apresentados os dados consolidados da turma para o experimento, tendo como base a correção dos artefatos entregues e a presença de erros de lógica nestes. Na linha de “Erro de lógica” constam todos os artefatos que apresentaram algum tipo de erro de lógica na resolução do problema, nas demais linhas estão os detalhamentos dos erros, quando cabível;

d) Tabela de resumo de notas: nesta tabela são apresentados os dados referente à nota atribuída para cada artefato considerado no experimento. Nela são apresentadas as quantidades de aluno para cada nota e o percentual que esta quantidade representa perante a amostra do experimento.

Para a tabela de resumo de compilações e para a tabela de classificação e erros, na coluna “Alunos com ocorrência” é indicado o número total de alunos que realizaram o que está descrito na linha. A coluna “% Alunos com ocorrência” indica o percentual de alunos que realizaram o que está descrito na linha perante o total de alunos do experimento. Para calcular o que está indicado na coluna “Média da turma” é considerado o total de alunos do experimento e, por sua vez, para calcular o valor da coluna “Média – alunos com ocorrência” é considerado apenas o número de alunos que realizou o que está descrito na linha.

Para a atribuição dos erros de lógica e da nota do exercício foram realizadas duas correções, sendo uma pelo autor deste trabalho de dissertação e a outra por outro docente da área de Ciência da Computação com experiência em ensino da disciplina de Lógica de Programação. As divergências entre as correções foram discutidas até que houvesse consenso por parte dos docentes.

Desta forma, nas próximas subseções serão relatados os experimentos realizados e os resultados oriundos destes.

## 4.3 EXPERIMENTOS 1 E 2

O experimento número 1 e o número 2 foram realizados para duas turmas de um curso técnico em informática de nível médio, com 35 alunos cada, e foram aplicados para alunos matriculados na disciplina de Lógica e Linguagem de Programação, disciplina pertencente à 1ª série do referido curso. O docente desta disciplina é o autor deste trabalho de mestrado. Devido as turmas serem da mesma instituição, mesmo curso, mesmo nível e mesma série, os experimentos serão relatados em conjunto.

Os alunos destas turmas utilizaram, durante as aulas, a linguagem C para aprender programação para computadores e utilizaram a IDE Code::Blocks. Para grande parte dos alunos destas turmas o conhecimento adquirido nesta disciplina foi o primeiro contato destes com programação para computadores. O experimento foi

aplicado no final do ano de 2017, após os alunos terem aprendido sobre estruturas de decisão, estruturas de repetição e vetores.

Para estes experimentos foi utilizado o laboratório da instituição de ensino em que estão matriculados e apenas os computadores do laboratório, sem acesso à Internet. O *plugin* pertinente ao ferramental do método proposto estava previamente instalado e ativado na IDE Code::Blocks. Ao término do exercício o professor coletou o arquivo XML gerado pelo *plugin* e o arquivo com o programa criado pelo aluno.

Conforme proposto no método, foram realizadas duas coletas. Para a primeira coleta foi aplicado o exercício 1 (ver seção 4.1.1), de estruturas de repetição e para a segunda coleta foi aplicado o exercício 2 (ver seção 4.1.2), de vetores. Após a primeira coleta foi realizada a devolutiva aos alunos a respeito do desempenho nesta. Esta devolutiva foi realizada antes da segunda coleta de dados. A seguir serão relatados os resultados destas coletas.

### 4.3.1 Resultados da primeira coleta – experimento 1

Os dados consolidados da primeira coleta para o experimento 1, a respeito das compilações e execuções, estão apresentados na Tabela 1. Com base nos dados apresentados é possível observar que a média de compilações realizadas para esta coleta foi de 37,34, sendo que a média de compilações com falha foi de 14,74 e a média de erros por compilação de 1,74. Também é possível verificar que 85,7% dos alunos do experimento repetiram erros entre as compilações, sendo que para estes, a média de erros repetidos foi de 18,23. Ainda, 30 dos 35 alunos (85,71%) tiveram a última compilação com sucesso, ou seja, o artefato entregue não apresentou erros de compilação.

TABELA 1 – RESUMO DE COMPILAÇÕES – EXPERIMENTO 1 – PRIMEIRA COLETA

| **Descrição** | **Alunos com ocorrência** | **% Alunos com ocorrência** | **Média da turma** | **Média – alunos com ocorrência** |
|---|---|---|---|---|
| Execuções realizadas | 32 | 91,4 | 20,49 | 22,41 |
| Compilações realizadas | 35 | 100,0 | 37,34 | 37,34 |
| Compilações com sucesso | 34 | 97,1 | 19,37 | 19,94 |
| Compilações com falha | 35 | 100,0 | 14,74 | 14,74 |
| Compilações sem efeito | 29 | 82,9 | 3,23 | 3,90 |
| Média de erros | 35 | - | 1,74 | 1,74 |
| Quantidade total de erros | 35 | - | 29,71 | 29,71 |
| Erros repetidos entre compilações | 30 | 85,7 | 15,63 | 18,23 |
| Última compilação com sucesso | 30 | 85,71 | - | - |

FONTE: o autor (2018).

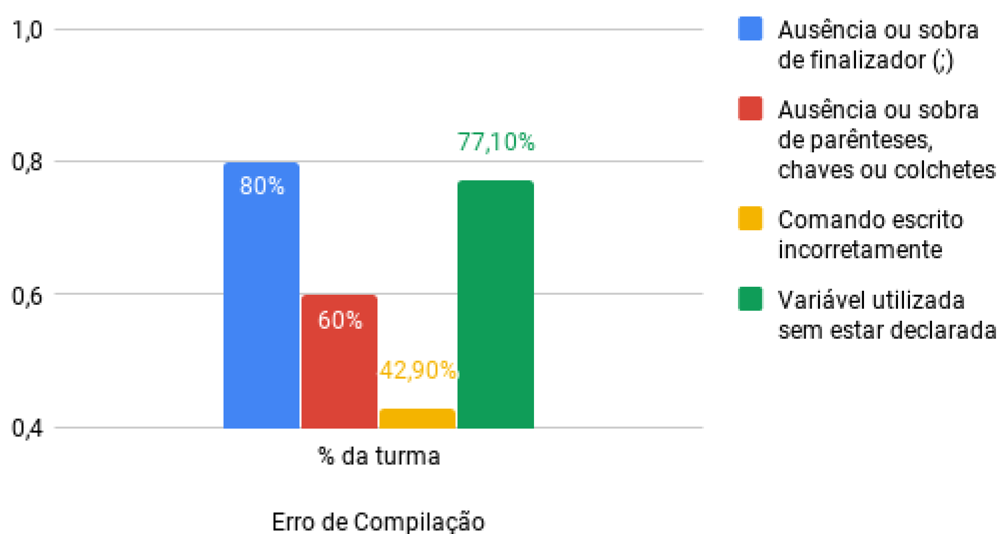
Os dados consolidados a respeito da classificação dos erros de compilação estão apresentados na Tabela 2 e ilustrados no Gráfico 1.

TABELA 2 – CLASSIFICAÇÃO DOS ERROS DE COMPILAÇÃO – EXPERIMENTO 1 – PRIMEIRA COLETA

| Classificação | Alunos com ocorrência | % Alunos com ocorrência | Média da turma | Média – alunos com ocorrência |
|---|---|---|---|---|
| Ausência ou sobra de finalizador (;) | 28 | 80,0 | 6,63 | 8,29 |
| Ausência ou sobra de parênteses, chaves ou colchetes | 21 | 60,0 | 6,17 | 10,29 |
| Comando escrito incorretamente (sintaxe incorreta ou parâmetros incorretos) | 15 | 42,9 | 5,83 | 13,60 |
| Variável utilizada sem estar declarada | 27 | 77,1 | 4,91 | 6,37 |
| Variável declarada incorretamente ou valor inicial atribuído incorretamente | 4 | 11,4 | 0,14 | 1,25 |
| Variável declarada, porém, utilizada com o nome escrito incorretamente | 0 | 0,0 | 0,00 | 0,00 |
| Tipo de dados incorreto ou inexistente | 0 | 0,0 | 0,00 | 0,00 |
| Comando/expressão incompleta (else sem if, while sem do, falta de parâmetros) | 9 | 25,7 | 0,91 | 3,56 |
| Conectivo de comparação incorreto | 2 | 5,7 | 0,20 | 3,50 |
| Atribuição de valores incorreta | 11 | 31,4 | 2,06 | 6,55 |

FONTE: o autor (2018).

GRÁFICO 1 – ERROS DE COMPILAÇÃO – EXPERIMENTO 1 – PRIMEIRA COLETA




FONTE: O autor (2018).

De acordo com os dados apresentados é possível observar que 80% dos alunos cometeram, ao menos uma vez, o erro de ausência ou sobra de finalizador

(;), sendo que a média deste erro para estes alunos foi de 8,29. Outro erro cometido por um grande percentual destes alunos (77,1%) foi o de variável utilizada sem estar declarada, sendo que a média deste erro para estes alunos foi de 6,37.

Os dados consolidados a respeito dos erros de lógica apresentados no artefato entregue estão descritos na Tabela 3. Nesta tabela é possível observar que 94,3% dos alunos entregaram o artefato com erros de lógica, sendo que 100% destes apresentaram erro aritmético, ou seja, problemas para o cálculo necessário para a resolução do exercício proposto.

TABELA 3 – RESUMO DE ERROS DE LÓGICA – EXPERIMENTO 1 – PRIMEIRA COLETA

| Descrição | Alunos com ocorrência | % de alunos da turma |
|---|---|---|
| Erro de Lógica | 33 | 94,3 |
| Erro com operador lógico | 1 | 2,9 |
| Erro com operador relacional | 0 | 0,0 |
| Erro de ordem/sequência de comandos | 9 | 25,7 |
| Erro aritmético | 33 | 94,3 |
| Erro com estruturas de repetição | 19 | 54,3 |
| Erro com estruturas de decisão | 0 | 0,0 |

FONTE: o autor (2018).

O resumo das notas atribuídas está descrito na Tabela 4. Para esta coleta a maioria dos alunos (42,9%) ficou com nota 40 e a média das notas da turma foi de 39,43.

TABELA 4 – RESUMO DE NOTAS – EXPERIMENTO 1 – PRIMEIRA COLETA

| Nota | Quantidade de Alunos | % de alunos da turma |
|---|---|---|
| Nota 100 | 1 | 2,9 |
| Nota 80 | 1 | 2,9 |
| Nota 60 | 7 | 20,0 |
| Nota 40 | 15 | 42,9 |
| Nota 20 | 9 | 25,7 |
| Nota 0 | 2 | 5,7 |

FONTE: o autor (2018).

### 4.3.2 Resultados da segunda coleta – experimento 1

Os dados consolidados da segunda coleta para o experimento 1, a respeito das compilações e execuções, estão apresentados na Tabela 5. Com base nos dados apresentados é possível observar que a média de compilações realizadas para esta coleta foi de 31,31, sendo que a média de compilações com falha foi de

10,63 e a média de erros por compilação de 2,03. Também é possível verificar que 65,7% dos alunos do experimento repetiram erros entre as compilações, sendo que para estes a média de erros repetidos foi de 26,09. Ainda, 33 dos 35 alunos (94,29%) tiveram a última compilação com sucesso, ou seja, o artefato entregue não apresentou erros de compilação.

Em comparação com a primeira coleta para o experimento 1, observa-se que na segunda coleta houve uma menor média de compilações realizadas (-6,03), menor média de compilações com falha (-4,11), porém um aumento na média de erros por compilação (+0,29). Na segunda coleta houve um menor número de alunos que repetiram erros entre as compilações (-20% da turma), porém, para os que repetiram erros, o número de erros repetidos aumentou (+7,86). Finalmente, o percentual de alunos que entregaram o artefato sem erros de compilação foi maior na segunda coleta (+8,58% da turma).

TABELA 5 – RESUMO DE COMPILAÇÕES – EXPERIMENTO 1 – SEGUNDA COLETA

| Descrição | Alunos com ocorrência | % Alunos com ocorrência | Média da turma | Média – alunos com ocorrência |
|---|---|---|---|---|
| Execuções realizadas | 33 | 94,3 | 20,71 | 21,97 |
| Compilações realizadas | 35 | 100,0 | 31,31 | 31,31 |
| Compilações com sucesso | 33 | 94,3 | 16,57 | 17,58 |
| Compilações com falha | 32 | 91,4 | 10,63 | 11,63 |
| Compilações sem efeito | 31 | 88,6 | 4,11 | 4,65 |
| Média de erros | 32 | - | 2,03 | 2,22 |
| Quantidade total de erros | 32 | - | 29,20 | 31,94 |
| Erros repetidos entre compilações | 23 | 65,7 | 17,14 | 26,09 |
| Última compilação com sucesso | 33 | 94,29 | - | - |

FONTE: o autor (2018).

Os dados consolidados a respeito da classificação dos erros de compilação estão apresentados na Tabela 6 e ilustrados no Gráfico 2. De acordo com os dados apresentados é possível observar que 68,6% dos alunos cometeram, ao menos uma vez, o erro de ausência ou sobra de finalizador (;), sendo que a média deste erro para estes alunos foi de 6,46. Outro erro cometido por um grande percentual destes alunos (77,1%) foi o de ausência ou sobra de parênteses, chaves ou colchetes, sendo que a média deste erro para estes alunos foi de 14,44.

Em comparação com a primeira coleta para o experimento 1, é possível observar que houve uma alteração entre os tipos de erro mais cometidos. Na primeira coleta foram os de ausência ou sobra de finalizador (;), cometido por 80%

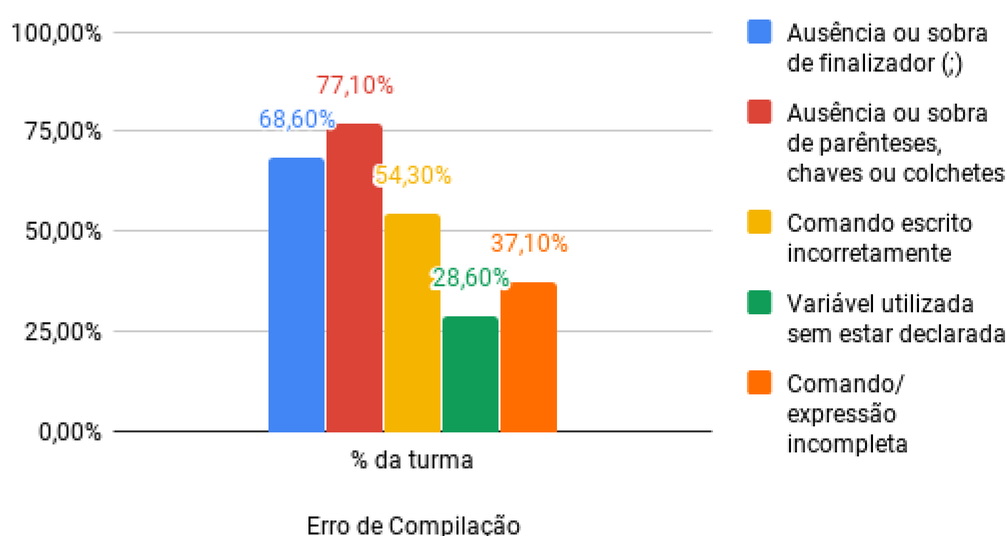
dos alunos, com média de 8,29 vezes e o de variável utilizada sem estar declarada, cometido por 77,1%, com média de 6,37 vezes.

TABELA 6 – CLASSIFICAÇÃO DOS ERROS DE COMPILAÇÃO – EXPERIMENTO 1 – SEGUNDA COLETA

| Classificação | Alunos com ocorrência | % Alunos com ocorrência | Média da turma | Média – alunos com ocorrência |
|---|---|---|---|---|
| Ausência ou sobra de finalizador (;) | 24 | 68,6 | 4,43 | 6,46 |
| Ausência ou sobra de parênteses, chaves ou colchetes | 27 | 77,1 | 11,14 | 14,44 |
| Comando escrito incorretamente (sintaxe incorreta ou parâmetros incorretos) | 19 | 54,3 | 6,80 | 12,53 |
| Variável utilizada sem estar declarada | 10 | 28,6 | 2,11 | 7,40 |
| Variável declarada incorretamente ou valor inicial atribuído incorretamente | 3 | 8,6 | 0,71 | 8,33 |
| Variável declarada, porém, utilizada com o nome escrito incorretamente | 6 | 17,1 | 0,83 | 4,83 |
| Tipo de dados incorreto ou inexistente | 7 | 20,0 | 1,71 | 8,57 |
| Comando/expressão incompleta (else sem if, while sem do, falta de parâmetros) | 13 | 37,1 | 1,31 | 3,54 |
| Conectivo de comparação incorreto | 2 | 5,7 | 0,06 | 1,00 |
| Atribuição de valores incorreta | 3 | 8,6 | 0,09 | 1,00 |

FONTE: o autor (2018).

GRÁFICO 2 – ERROS DE COMPILAÇÃO – EXPERIMENTO 1 – SEGUNDA COLETA




FONTE: O autor (2018).

Os dados consolidados a respeito dos erros de lógica apresentados no artefato entregue estão descritos na Tabela 7. Nesta tabela é possível observar que

82,9% dos alunos entregaram o artefato com erros de lógica, sendo que o problema estava na forma como a solução foi elaborada e não outro erro específico de lógica.

Em comparação com a primeira coleta do experimento 1 é possível observar que houve uma diminuição dos alunos que entregaram os artefatos com erro de lógica (-11,4% da turma).

TABELA 7 – RESUMO DE ERROS DE LÓGICA – EXPERIMENTO 1 – SEGUNDA COLETA

| Descrição | Alunos com ocorrência | % de alunos da turma |
|---|---|---|
| Erro de Lógica | 29 | 82,9 |
| Erro com operador lógico | 0 | 0,0 |
| Erro com operador relacional | 0 | 0,0 |
| Erro de ordem/sequência de comandos | 0 | 0,0 |
| Erro aritmético | 0 | 0,0 |
| Erro com estruturas de repetição | 3 | 8,6 |
| Erro com estruturas de decisão | 3 | 8,6 |

FONTE: o autor (2018).

O resumo das notas atribuídas está descrito na Tabela 8. Para esta coleta a maioria dos alunos (48,6%) ficou com nota 60 e a média das notas da turma foi de 56,00.

Em comparação com a primeira coleta do experimento 1 foi possível observar uma melhora na média da turma (+16,57) e alteração da maioria da turma da nota 40 para a nota 60.

TABELA 8 – RESUMO DE NOTAS – EXPERIMENTO 1 – SEGUNDA COLETA

| Nota | Quantidade de Alunos | % de alunos da turma |
|---|---|---|
| Nota 100 | 6 | 17,1 |
| Nota 80 | 0 | 0,0 |
| Nota 60 | 17 | 48,6 |
| Nota 40 | 7 | 20,0 |
| Nota 20 | 3 | 8,6 |
| Nota 0 | 2 | 5,7 |

FONTE: o autor (2018).

### 4.3.3 Resultados da primeira coleta – experimento 2

Os dados consolidados da primeira coleta para o experimento 2, a respeito das compilações e execuções, estão apresentados na Tabela 9. Com base nos dados apresentados é possível observar que a média de compilações realizadas para esta coleta foi de 42,51, sendo que a média de compilações com falha foi de

20,37 e a média de erros por compilação de 1,79. Também é possível verificar que 77,1% dos alunos do experimento repetiram erros entre as compilações, sendo que para estes a média de erros repetidos foi de 41,26. Ainda, 30 dos 35 alunos (85,71%) tiveram a última compilação com sucesso, ou seja, o artefato entregue não apresentou erros de compilação.

TABELA 9 – RESUMO DE COMPILAÇÕES – EXPERIMENTO 2 – PRIMEIRA COLETA

| Descrição | Alunos com ocorrência | % Alunos com ocorrência | Média da turma | Média – alunos com ocorrência |
|---|---|---|---|---|
| Execuções realizadas | 29 | 82,9 | 21,11 | 25,48 |
| Compilações realizadas | 35 | 100,0 | 42,51 | 42,51 |
| Compilações com sucesso | 31 | 88,6 | 17,54 | 19,81 |
| Compilações com falha | 33 | 94,3 | 20,37 | 21,61 |
| Compilações sem efeito | 23 | 65,7 | 4,60 | 7,00 |
| Média de erros | 33 | - | 1,79 | 1,90 |
| Quantidade total de erros | 33 | - | 50,17 | 53,21 |
| Erros repetidos entre compilações | 27 | 77,1 | 31,83 | 41,26 |
| Última compilação com sucesso | 30 | 85,71 | - | - |

FONTE: o autor (2018).

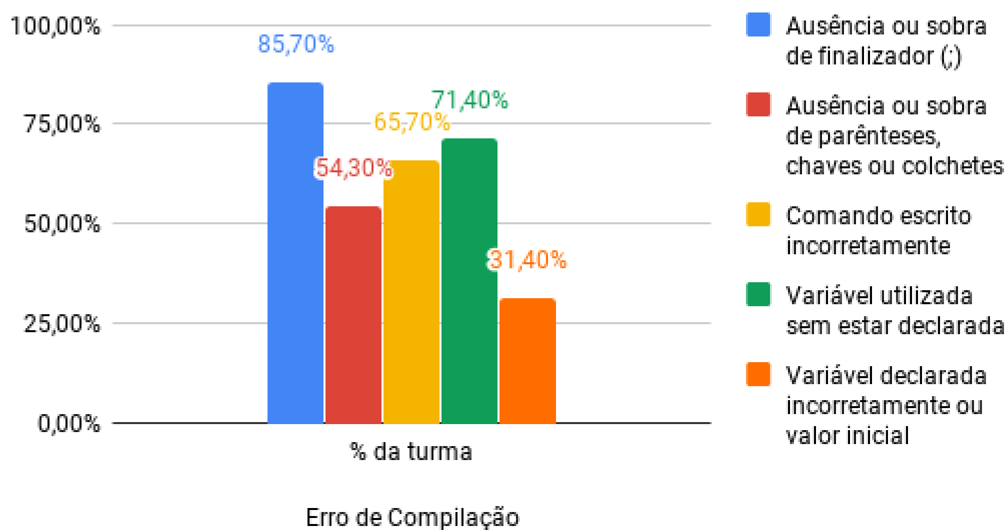
Os dados consolidados a respeito da classificação dos erros de compilação estão apresentados na Tabela 10 e ilustrados no Gráfico 3.

TABELA 10 – CLASSIFICAÇÃO DOS ERROS DE COMPILAÇÃO – EXPERIMENTO 2 – PRIMEIRA COLETA

| Classificação | Alunos com ocorrência | % Alunos com ocorrência | Média da turma | Média – alunos com ocorrência |
|---|---|---|---|---|
| Ausência ou sobra de finalizador (;) | 30 | 85,7 | 18,94 | 22,10 |
| Ausência ou sobra de parênteses, chaves ou colchetes | 19 | 54,3 | 8,83 | 16,26 |
| Comando escrito incorretamente (sintaxe incorreta ou parâmetros incorretos) | 23 | 65,7 | 12,06 | 18,35 |
| Variável utilizada sem estar declarada | 25 | 71,4 | 7,57 | 10,60 |
| Variável declarada incorretamente ou valor inicial atribuído incorretamente | 11 | 31,4 | 1,20 | 3,82 |
| Variável declarada, porém, utilizada com o nome escrito incorretamente | 3 | 8,6 | 0,09 | 1,00 |
| Tipo de dados incorreto ou inexistente | 1 | 2,9 | 0,06 | 2,00 |
| Comando/expressão incompleta (else sem if, while sem do, falta de parâmetros) | 3 | 8,6 | 0,11 | 1,33 |
| Conectivo de comparação incorreto | 0 | 0,0 | 0,00 | 0,00 |
| Atribuição de valores incorreta | 7 | 20,0 | 1,31 | 6,57 |

FONTE: o autor (2018).

GRÁFICO 3 – ERROS DE COMPILAÇÃO – EXPERIMENTO 2 – PRIMEIRA COLETA




FONTE: O autor (2018).

De acordo com os dados apresentados é possível observar que 85,7% dos alunos cometeram, ao menos uma vez, o erro de ausência ou sobra de finalizador (;), sendo que a média deste erro para estes alunos foi de 22,10. Outro erro cometido por um grande percentual destes alunos (71,4%) foi o de variável utilizada sem estar declarada, sendo que a média deste erro para estes alunos foi de 10,60.

Os dados consolidados a respeito dos erros de lógica apresentados no artefato entregue estão descritos na Tabela 11. Nesta tabela é possível observar que 88,6% dos alunos entregaram o artefato com erros de lógica, sendo que 80% destes apresentaram erro aritmético, ou seja, problemas para o cálculo necessário para a resolução do exercício proposto.

TABELA 11 – RESUMO DE ERROS DE LÓGICA – EXPERIMENTO 2 – PRIMEIRA COLETA

| Descrição | Alunos com ocorrência | % de alunos da turma |
|---|---|---|
| Erro de Lógica | 31 | 88,6 |
| Erro com operador lógico | 0 | 0,0 |
| Erro com operador relacional | 0 | 0,0 |
| Erro de ordem/sequência de comandos | 8 | 22,9 |
| Erro aritmético | 28 | 80,0 |
| Erro com estruturas de repetição | 16 | 45,7 |
| Erro com estruturas de decisão | 2 | 5,7 |

FONTE: o autor (2018).

O resumo das notas atribuídas está descrito na Tabela 12. Para esta coleta a maioria dos alunos (28,6%) ficou com nota 20 e a média das notas da turma foi de 36,00.

TABELA 12 – RESUMO DE NOTAS – EXPERIMENTO 2 – PRIMEIRA COLETA

| Nota | Quantidade de Alunos | % de alunos da turma |
|---|---|---|
| Nota 100 | 3 | 8,6 |
| Nota 80 | 1 | 2,9 |
| Nota 60 | 8 | 22,9 |
| Nota 40 | 5 | 14,3 |
| Nota 20 | 10 | 28,6 |
| Nota 0 | 8 | 22,9 |

FONTE: o autor (2018).

#### 4.3.4 Resultados da segunda coleta – experimento 2

Os dados consolidados da segunda coleta para o experimento 2, a respeito das compilações e execuções, estão apresentados na Tabela 13. Com base nos dados apresentados é possível observar que a média de compilações realizadas para esta coleta foi de 29,66, sendo que a média de compilações com falha foi de 9,77 e a média de erros por compilação de 2,05. Também é possível verificar que 68,6% dos alunos do experimento repetiram erros entre as compilações, sendo que para estes a média de erros repetidos foi de 17,04. Ainda, 29 dos 35 alunos (82,86%) tiveram a última compilação com sucesso, ou seja, o artefato entregue não apresentou erros de compilação.

Em comparação com a primeira coleta para o experimento 2, observa-se que na segunda coleta houve uma menor média de compilações realizadas (-12,85), menor média de compilações com falha (-10,6), porém um aumento na média de erros por compilação (+0,26). Na segunda coleta houve um menor número de alunos que repetiram erros entre as compilações (-8,5% da turma), e para os que repetiram erros, o número de erros repetidos diminuiu (-24,22). Finalmente, o percentual de alunos que entregaram o artefato sem erros de compilação foi menor na segunda coleta (-2,85% da turma).

TABELA 13 – RESUMO DE COMPILAÇÕES – EXPERIMENTO 2 – SEGUNDA COLETA

| Descrição | Alunos com ocorrência | % Alunos com ocorrência | Média da turma | Média – alunos com ocorrência |
|---|---|---|---|---|
| Execuções realizadas | 33 | 94,3 | 19,31 | 20,48 |
| Compilações realizadas | 35 | 100,0 | 29,66 | 29,66 |
| Compilações com sucesso | 30 | 85,7 | 16,31 | 19,03 |
| Compilações com falha | 33 | 94,3 | 9,77 | 10,36 |
| Compilações sem efeito | 28 | 80,0 | 3,57 | 4,46 |
| Média de erros | 33 | - | 2,05 | 2,17 |
| Quantidade total de erros | 33 | - | 25,26 | 26,79 |
| Erros repetidos entre compilações | 24 | 68,6 | 11,69 | 17,04 |
| Última compilação com sucesso | 29 | 82,86 | - | - |

FONTE: o autor (2018).

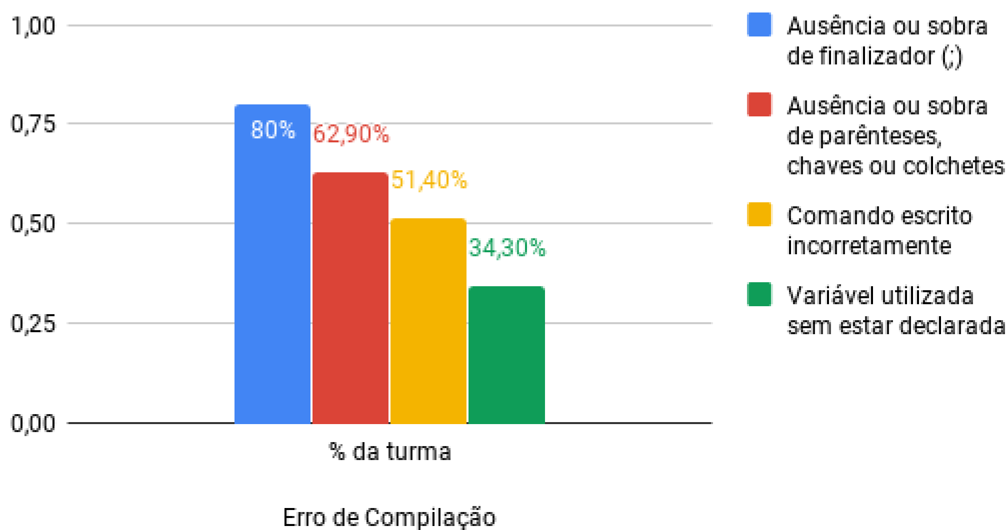
Os dados consolidados a respeito da classificação dos erros de compilação estão apresentados na Tabela 14 e ilustrados no Gráfico 4. De acordo com os dados apresentados é possível observar que 80% dos alunos cometeram, ao menos uma vez, o erro de ausência ou sobra de finalizador (;), sendo que a média deste erro para estes alunos foi de 9,93. Outro erro cometido por um grande percentual destes alunos (62,9%) foi o de ausência ou sobra de parênteses, chaves ou colchetes, sendo que a média deste erro para estes alunos foi de 13,00.

TABELA 14 – CLASSIFICAÇÃO DOS ERROS DE COMPILAÇÃO – EXPERIMENTO 2 – SEGUNDA COLETA

| Classificação | Alunos com ocorrência | % Alunos com ocorrência | Média da turma | Média – alunos com ocorrência |
|---|---|---|---|---|
| Ausência ou sobra de finalizador (;) | 28 | 80,0 | 7,94 | 9,93 |
| Ausência ou sobra de parênteses, chaves ou colchetes | 22 | 62,9 | 8,17 | 13,00 |
| Comando escrito incorretamente (sintaxe incorreta ou parâmetros incorretos) | 18 | 51,4 | 3,43 | 6,67 |
| Variável utilizada sem estar declarada | 12 | 34,3 | 1,14 | 3,33 |
| Variável declarada incorretamente ou valor inicial atribuído incorretamente | 8 | 22,9 | 0,77 | 3,38 |
| Variável declarada, porém, utilizada com o nome escrito incorretamente | 5 | 14,3 | 0,14 | 1,00 |
| Tipo de dados incorreto ou inexistente | 9 | 25,7 | 0,80 | 3,11 |
| Comando/expressão incompleta (else sem if, while sem do, falta de parâmetros) | 8 | 22,9 | 1,11 | 4,88 |
| Conectivo de comparação incorreto | 6 | 17,1 | 1,49 | 8,67 |
| Atribuição de valores incorreta | 3 | 8,6 | 0,26 | 3,00 |

FONTE: o autor (2018).

GRÁFICO 4 – ERROS DE COMPILAÇÃO – EXPERIMENTO 2 – SEGUNDA COLETA




FONTE: O autor (2018).

Em comparação com a primeira coleta para o experimento 2, é possível observar que houve uma alteração entre os tipos de erro mais cometidos. Na primeira coleta foram os de ausência ou sobra de finalizador (;), cometido por 85,7% dos alunos, com média de 22,10 vezes e o de variável utilizada sem estar declarada, cometido por 71,4%, com média de 10,60 vezes.

Os dados consolidados a respeito dos erros de lógica apresentados no artefato entregue estão descritos na Tabela 15. Nesta tabela é possível observar que 82,9% dos alunos entregaram o artefato com erros de lógica, sendo que o problema estava na forma como a solução foi elaborada e não outro erro específico de lógica.

Em comparação com a primeira coleta do experimento 2 é possível observar que houve uma diminuição dos alunos que entregaram os artefatos com erro de lógica (-5,7% da turma).

TABELA 15 – RESUMO DE ERROS DE LÓGICA – EXPERIMENTO 2 – SEGUNDA COLETA

| Descrição | Alunos com ocorrência | % de alunos da turma |
|---|---|---|
| Erro de Lógica | 29 | 82,9 |
| Erro com operador lógico | 3 | 8,6 |
| Erro com operador relacional | 0 | 0,0 |
| Erro de ordem/sequência de comandos | 0 | 0,0 |
| Erro aritmético | 1 | 2,9 |
| Erro com estruturas de repetição | 7 | 20,0 |
| Erro com estruturas de decisão | 10 | 28,6 |

FONTE: o autor (2018).

O resumo das notas atribuídas está descrito na Tabela 16. Para esta coleta a maioria dos alunos (37,1%) ficou com nota 60 e a média das notas da turma foi de 49,14. Em comparação com a primeira coleta do experimento 2 foi possível observar uma melhora na média da turma (+13,14) e alteração da maioria da turma da nota 20 para a nota 60.

TABELA 16 – RESUMO DE NOTAS – EXPERIMENTO 2 – SEGUNDA COLETA

| Nota | Quantidade de Alunos | % de alunos da turma |
|---|---|---|
| Nota 100 | 6 | 17,1 |
| Nota 80 | 0 | 0,0 |
| Nota 60 | 13 | 37,1 |
| Nota 40 | 6 | 17,1 |
| Nota 20 | 5 | 14,3 |
| Nota 0 | 5 | 14,3 |

FONTE: o autor (2018).

## 4.4 EXPERIMENTO 3

O experimento 3 foi realizado com uma turma do curso de Sistemas de Informação de um centro universitário privado e foi aplicado para 5 alunos matriculados na disciplina de Estrutura de Dados, disciplina pertencente ao 2º semestre (1ª série) do referido curso. As coletas foram realizadas sem a participação direta dos autores, sendo conduzidas pelo professor da disciplina.

Os alunos desta turma utilizaram durante as aulas a linguagem C para aprender programação para computadores e utilizaram a IDE Code::Blocks. O experimento foi aplicado no final do ano de 2017.

Para este experimento foi utilizado o laboratório da instituição de ensino em que estão matriculados e apenas os computadores do laboratório, sem acesso à Internet. O *plugin* estava previamente instalado e ativado na IDE Code::Blocks. Ao término do exercício o professor coletou o arquivo XML gerado pelo *plugin* e o arquivo com o programa criado pelo aluno.

Conforme proposto no método, foram realizadas duas coletas. Para a primeira coleta foi aplicado o exercício 3 (ver seção 4.1.3), de busca binária e para a segunda coleta foi aplicado o exercício 4 (ver seção 4.1.4), de ordenação. Após a primeira coleta foi realizada a devolutiva aos alunos a respeito do desempenho

nesta, esta devolutiva foi realizada antes da segunda coleta de dados. A seguir serão relatados os resultados destas coletas.

### 4.4.1 Resultados da primeira coleta – experimento 3

Os dados consolidados da primeira coleta para o experimento 3, a respeito das compilações e execuções, estão apresentados na Tabela 17. Com base nos dados apresentados é possível observar que a média de compilações realizadas para esta coleta foi de 30,80, sendo que a média de compilações com falha foi de 13,00 e a média de erros por compilação de 4,19. Também é possível verificar que 100% dos alunos do experimento repetiram erros entre as compilações, sendo que para estes, a média de erros repetidos foi de 10,20. Ainda, apenas 2 dos 5 alunos (40%) tiveram a última compilação com sucesso, ou seja, o artefato entregue não apresentou erros de compilação.

TABELA 17 – RESUMO DE COMPILAÇÕES – EXPERIMENTO 3 – PRIMEIRA COLETA

| Descrição | Alunos com ocorrência | % Alunos com ocorrência | Média da turma | Média – alunos com ocorrência |
|---|---|---|---|---|
| Execuções realizadas | 3 | 60,0 | 15,20 | 25,33 |
| Compilações realizadas | 5 | 100,0 | 30,80 | 30,80 |
| Compilações com sucesso | 3 | 60,0 | 9,80 | 16,33 |
| Compilações com falha | 5 | 100,0 | 13,00 | 13,00 |
| Compilações sem efeito | 3 | 60,0 | 8,00 | 13,33 |
| Média de erros | 5 | - | 4,19 | 4,19 |
| Quantidade total de erros | 5 | - | 32,40 | 32,40 |
| Erros repetidos entre compilações | 5 | 100,0 | 10,20 | 10,20 |
| Última compilação com sucesso | 2 | 40,00 | - | - |

FONTE: o autor (2018).

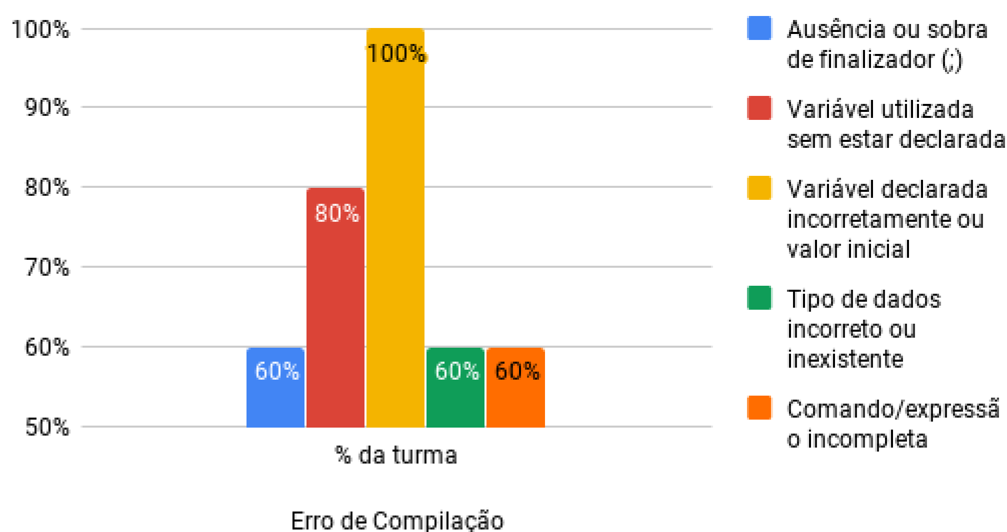
Os dados consolidados a respeito da classificação dos erros de compilação estão apresentados na Tabela 18 e ilustrados no Gráfico 5. De acordo com os dados apresentados é possível observar que 100% dos alunos cometeram, ao menos uma vez, o erro de variável declarada incorretamente ou valor inicial atribuído incorretamente, sendo que a média deste erro para estes alunos foi de 4,80. Outro erro cometido por um grande percentual destes alunos (80%) foi o de variável utilizada sem estar declarada, sendo que a média deste erro para estes alunos foi de 4,75.

TABELA 18 – CLASSIFICAÇÃO DOS ERROS DE COMPILAÇÃO – EXPERIMENTO 3 – PRIMEIRA COLETA

| Classificação | Alunos com ocorrência | % Alunos com ocorrência | Média da turma | Média – alunos com ocorrência |
|---|---|---|---|---|
| Ausência ou sobra de finalizador (;) | 3 | 60,0 | 5,60 | 9,33 |
| Ausência ou sobra de parênteses, chaves ou colchetes | 2 | 40,0 | 3,60 | 9,00 |
| Comando escrito incorretamente (sintaxe incorreta ou parâmetros incorretos) | 2 | 40,0 | 1,20 | 3,00 |
| Variável utilizada sem estar declarada | 4 | 80,0 | 3,80 | 4,75 |
| Variável declarada incorretamente ou valor inicial atribuído incorretamente | 5 | 100,0 | 4,80 | 4,80 |
| Variável declarada, porém, utilizada com o nome escrito incorretamente | 0 | 0,0 | 0,00 | 0,00 |
| Tipo de dados incorreto ou inexistente | 3 | 60,0 | 3,60 | 6,00 |
| Comando/expressão incompleta (else sem if, while sem do, falta de parâmetros) | 3 | 60,0 | 9,60 | 16,00 |
| Conectivo de comparação incorreto | 1 | 20,0 | 0,20 | 1,00 |
| Atribuição de valores incorreta | 0 | 0,0 | 0,00 | 0,00 |

FONTE: O autor (2018).

GRÁFICO 5 – ERROS DE COMPILAÇÃO – EXPERIMENTO 3 – PRIMEIRA COLETA




FONTE: O autor (2018).

Os dados consolidados a respeito dos erros de lógica apresentados no artefato entregue estão descritos na Tabela 19. Nesta tabela é possível observar que todos os alunos entregaram o artefato com erros de lógica.

TABELA 19 – RESUMO DE ERROS DE LÓGICA – EXPERIMENTO 3 – PRIMEIRA COLETA

| Descrição | Alunos com ocorrência | % de alunos da turma |
|---|---|---|
| Erro de Lógica | 5 | 100,0 |
| Erro com operador lógico | 1 | 20,0 |
| Erro com operador relacional | 0 | 0,0 |
| Erro de ordem/sequência de comandos | 3 | 60,0 |
| Erro aritmético | 0 | 0,0 |
| Erro com estruturas de repetição | 3 | 60,0 |
| Erro com estruturas de decisão | 3 | 60,0 |

FONTE: O autor (2018).

O resumo das notas atribuídas está descrito na Tabela 20. Para esta coleta a média das notas da turma foi de 24,00.

TABELA 20 – RESUMO DE NOTAS – EXPERIMENTO 3 – PRIMEIRA COLETA

| Nota | Quantidade de Alunos | % de alunos da turma |
|---|---|---|
| Nota 100 | 0 | 0,0 |
| Nota 80 | 0 | 0,0 |
| Nota 60 | 2 | 40,0 |
| Nota 40 | 0 | 0,0 |
| Nota 20 | 0 | 0,0 |
| Nota 0 | 3 | 60,0 |

FONTE: O autor (2018).

### 4.4.2 Resultados da segunda coleta – experimento 3

Os dados consolidados da segunda coleta para o experimento 3, a respeito das compilações e execuções, estão apresentados na Tabela 21. Com base nos dados apresentados é possível observar que a média de compilações realizadas para esta coleta foi de 20,00, sendo que a média de compilações com falha foi de 6,00 e a média de erros por compilação de 2,62. Também é possível verificar que 60% dos alunos do experimento repetiram erros entre as compilações, sendo que para estes a média de erros repetidos foi de 2,00. Ainda, 3 dos 5 alunos (60%) tiveram a última compilação com sucesso, ou seja, o artefato entregue não apresentou erros de compilação.

Em comparação com a primeira coleta para o experimento 3, observa-se que na segunda coleta houve uma menor média de compilações realizadas (-10,80), menor média de compilações com falha (-7,00) e menor média de erros por compilação (-1,57). Na segunda coleta houve um menor número de alunos que

repetiram erros entre as compilações (-40% da turma), e para os que repetiram erros, o número de erros repetidos diminuiu (-8,20). Finalmente, o percentual de alunos que entregaram o artefato sem erros de compilação foi maior na segunda coleta (+20% da turma).

TABELA 21 – RESUMO DE COMPILAÇÕES – EXPERIMENTO 3 – SEGUNDA COLETA

| Descrição | Alunos com ocorrência | % Alunos com ocorrência | Média da turma | Média – alunos com ocorrência |
|---|---|---|---|---|
| Execuções realizadas | 4 | 80,0 | 12,20 | 15,25 |
| Compilações realizadas | 5 | 100,0 | 20,00 | 20,00 |
| Compilações com sucesso | 4 | 80,0 | 11,60 | 14,50 |
| Compilações com falha | 5 | 100,0 | 6,00 | 6,00 |
| Compilações sem efeito | 3 | 60,0 | 2,40 | 4,00 |
| Média de erros | 5 | - | 2,62 | 2,62 |
| Quantidade total de erros | 5 | - | 12,20 | 12,20 |
| Erros repetidos entre compilações | 3 | 60,0 | 1,20 | 2,00 |
| Última compilação com sucesso | 3 | 60,00 | - | - |

FONTE: O autor (2018).

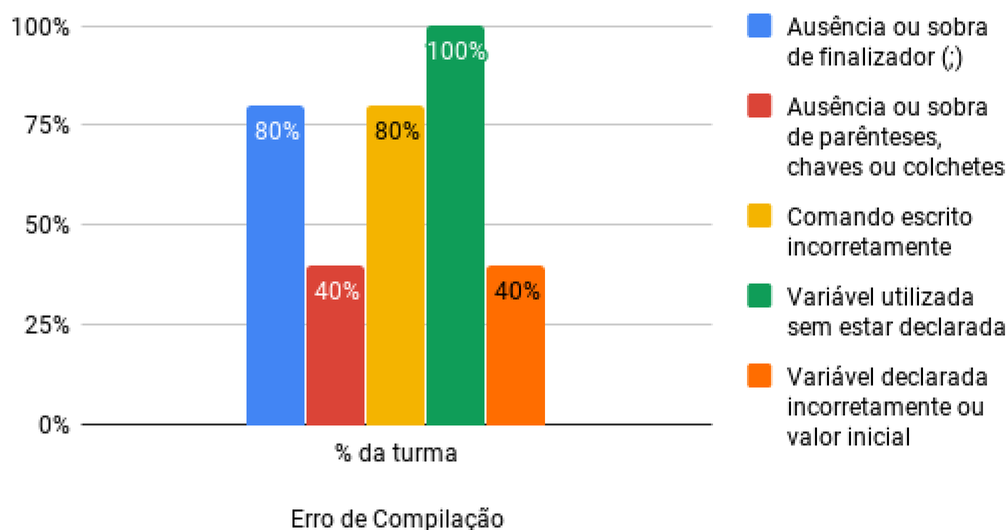
Os dados consolidados a respeito da classificação dos erros de compilação estão apresentados na Tabela 22 e ilustrados no Gráfico 6.

TABELA 22 – CLASSIFICAÇÃO DOS ERROS DE COMPILAÇÃO – EXPERIMENTO 3 – SEGUNDA COLETA

| Classificação | Alunos com ocorrência | % Alunos com ocorrência | Média da turma | Média – alunos com ocorrência |
|---|---|---|---|---|
| Ausência ou sobra de finalizador (;) | 4 | 80,0 | 1,20 | 1,50 |
| Ausência ou sobra de parênteses, chaves ou colchetes | 2 | 40,0 | 1,80 | 4,50 |
| Comando escrito incorretamente (sintaxe incorreta ou parâmetros incorretos) | 4 | 80,0 | 1,80 | 2,25 |
| Variável utilizada sem estar declarada | 5 | 100,0 | 6,00 | 6,00 |
| Variável declarada incorretamente ou valor inicial atribuído incorretamente | 2 | 40,0 | 1,20 | 3,00 |
| Variável declarada, porém, utilizada com o nome escrito incorretamente | 0 | 0,0 | 0,00 | 0,00 |
| Tipo de dados incorreto ou inexistente | 0 | 0,0 | 0,00 | 0,00 |
| Comando/expressão incompleta (else sem if, while sem do, falta de parâmetros) | 1 | 20,0 | 0,20 | 1,00 |
| Conectivo de comparação incorreto | 0 | 0,0 | 0,00 | 0,00 |
| Atribuição de valores incorreta | 0 | 0,0 | 0,00 | 0,00 |

FONTE: O autor (2018).

GRÁFICO 6 – ERROS DE COMPILAÇÃO – EXPERIMENTO 3 – SEGUNDA COLETA




FONTE: O autor (2018).

De acordo com os dados apresentados é possível observar que 100% dos alunos cometeram, ao menos uma vez, o erro de variável utilizada sem estar declarada, sendo que a média deste erro para estes alunos foi de 6,00. Outros erros cometidos por um grande percentual destes alunos (80%) foi o de ausência ou sobra de finalizador (;), sendo que a média deste erro para estes alunos foi de 1,50 e o de comando escrito incorretamente (sintaxe incorreta ou parâmetros incorretos), média de 2,25.

Os dados consolidados a respeito dos erros de lógica apresentados no artefato entregue estão descritos na Tabela 23. Nesta tabela é possível observar que, assim como na primeira coleta, todos os alunos entregaram o artefato com erros de lógica.

TABELA 23 – RESUMO DE ERROS DE LÓGICA – EXPERIMENTO 3 – SEGUNDA COLETA

| Descrição | Alunos com ocorrência | % de alunos da turma |
|---|---|---|
| Erro de Lógica | 5 | 100,0 |
| Erro com operador lógico | 0 | 0,0 |
| Erro com operador relacional | 0 | 0,0 |
| Erro de ordem/sequência de comandos | 1 | 20,0 |
| Erro aritmético | 0 | 0,0 |
| Erro com estruturas de repetição | 1 | 20,0 |
| Erro com estruturas de decisão | 1 | 20,0 |

FONTE: O autor (2018).

O resumo das notas atribuídas está descrito na Tabela 24. Para esta coleta a média das notas da turma foi de 32,00. Em comparação com a primeira coleta do experimento 3 foi possível observar uma melhora na média da turma (+8,00).

TABELA 24 – RESUMO DE NOTAS – EXPERIMENTO 3 – SEGUNDA COLETA

| Nota | Quantidade de Alunos | % de alunos da turma |
|---|---|---|
| Nota 100 | 0 | 0,0 |
| Nota 80 | 0 | 0,0 |
| Nota 60 | 2 | 40,0 |
| Nota 40 | 0 | 0,0 |
| Nota 20 | 2 | 40,0 |
| Nota 0 | 1 | 20,0 |

FONTE: O autor (2018).

## 4.5 EXPERIMENTO 4

O experimento 4 foi realizado com uma turma de uma universidade pública, sendo aplicado a uma turma com 14 alunos dos cursos de Sistemas de Informação, Engenharia de Computação e Engenharia Eletrônica, matriculados na disciplina de Fundamentos de Programação 1, disciplina pertencente a 1ª série dos referidos cursos. A primeira coleta foi realizada com participação do autor principal deste trabalho, a segunda coleta foi realizada diretamente pelo professor da disciplina.

Os alunos desta turma utilizaram durante as aulas a linguagem C para aprender programação para computadores e utilizaram a IDE Code::Blocks. O experimento foi aplicado no primeiro semestre do ano de 2018.

Para este experimento foi utilizado o laboratório da instituição de ensino em que os alunos estão matriculados, sendo que os alunos puderam utilizar os computadores do laboratório e também seus próprios notebooks, orientados à não utilizar a Internet. O *plugin* foi instalado na IDE Code::Blocks pelos alunos antes de iniciar o exercício. Ao término do exercício cada aluno enviou o programa e o arquivo XML gerado durante a resolução do exercício.

Conforme proposto no método, foram realizadas duas coletas. Para a primeira coleta foi aplicado o exercício 3 (ver seção 4.1.3), de busca binária e para a segunda coleta foi aplicado o exercício 4 (ver seção 4.1.4), de ordenação. Após a primeira coleta foi realizada a devolutiva aos alunos a respeito do desempenho

nesta, esta devolutiva foi realizada antes da segunda coleta de dados. A seguir serão relatados os resultados destas coletas.

### 4.5.1 Resultados da primeira coleta – experimento 4

Os dados consolidados da primeira coleta para o experimento 4, a respeito das compilações e execuções, estão apresentados na Tabela 25. Com base nos dados apresentados é possível observar que a média de compilações realizadas para esta coleta foi de 19,50, sendo que a média de compilações com falha foi de 4,79 e a média de erros por compilação de 2,04. Também é possível verificar que 42,9% dos alunos do experimento repetiram erros entre as compilações, sendo que para estes, a média de erros repetidos foi de 13,33. Ainda, 10 dos 14 alunos (71,43%) tiveram a última compilação com sucesso, ou seja, o artefato entregue não apresentou erros de compilação.

TABELA 25 – RESUMO DE COMPILAÇÕES – EXPERIMENTO 4 – PRIMEIRA COLETA

| Descrição | Alunos com ocorrência | % Alunos com ocorrência | Média da turma | Média – alunos com ocorrência |
|---|---|---|---|---|
| Execuções realizadas | 10 | 71,4 | 14,71 | 20,60 |
| Compilações realizadas | 12 | 85,7 | 19,50 | 22,75 |
| Compilações com sucesso | 11 | 78,6 | 8,64 | 11,00 |
| Compilações com falha | 11 | 78,6 | 4,79 | 6,09 |
| Compilações sem efeito | 10 | 71,4 | 6,07 | 8,50 |
| Média de erros | 11 | - | 2,04 | 2,60 |
| Quantidade total de erros | 11 | - | 18,14 | 23,09 |
| Erros repetidos entre compilações | 6 | 42,9 | 5,71 | 13,33 |
| Última compilação com sucesso | 10 | 71,43 | - | - |

FONTE: O autor (2018).

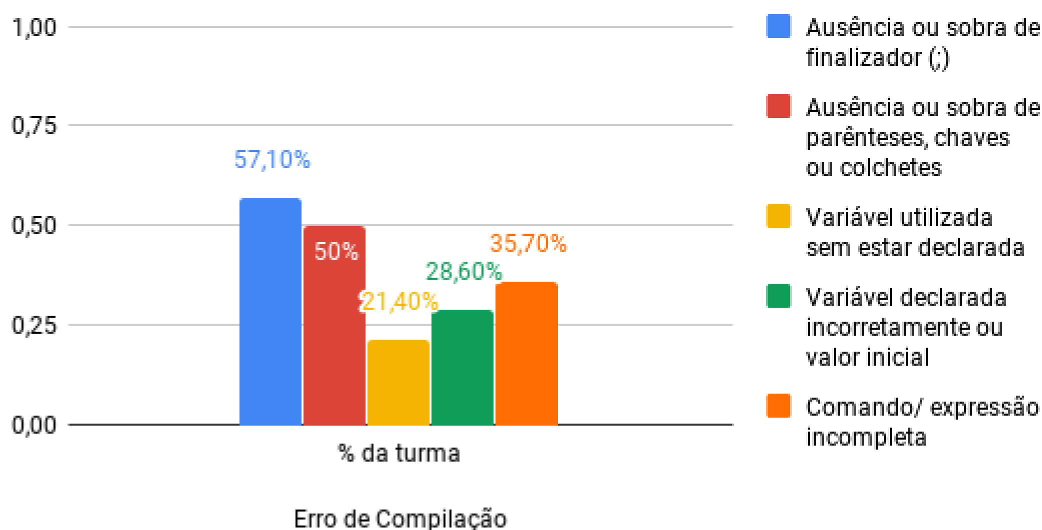
Os dados consolidados a respeito da classificação dos erros de compilação estão apresentados na Tabela 26 e ilustrados no Gráfico 7. De acordo com os dados apresentados é possível observar que 57,1% dos alunos cometeram, ao menos uma vez, o erro de ausência ou sobra de finalizador (;), sendo que a média deste erro para estes alunos foi de 1,50. Outro erro cometido por um grande percentual destes alunos (50%) foi o de ausência ou sobra de parênteses, chaves ou colchetes, sendo que a média deste erro para estes alunos foi de 26,57.

TABELA 26 – CLASSIFICAÇÃO DOS ERROS DE COMPILAÇÃO – EXPERIMENTO 4 – PRIMEIRA COLETA

| Classificação | Alunos com ocorrência | % Alunos com ocorrência | Média da turma | Média – alunos com ocorrência |
|---|---|---|---|---|
| Ausência ou sobra de finalizador (;) | 8 | 57,1 | 0,86 | 1,50 |
| Ausência ou sobra de parênteses, chaves ou colchetes | 7 | 50,0 | 13,29 | 26,57 |
| Comando escrito incorretamente (sintaxe incorreta ou parâmetros incorretos) | 1 | 7,1 | 0,07 | 1,00 |
| Variável utilizada sem estar declarada | 3 | 21,4 | 0,64 | 3,00 |
| Variável declarada incorretamente ou valor inicial atribuído incorretamente | 4 | 28,6 | 1,50 | 5,25 |
| Variável declarada, porém, utilizada com o nome escrito incorretamente | 0 | 0,0 | 0,00 | 0,00 |
| Tipo de dados incorreto ou inexistente | 1 | 7,1 | 0,36 | 5,00 |
| Comando/expressão incompleta (else sem if, while sem do, falta de parâmetros) | 5 | 35,7 | 1,36 | 3,80 |
| Conectivo de comparação incorreto | 0 | 0,0 | 0,00 | 0,00 |
| Atribuição de valores incorreta | 1 | 7,1 | 0,07 | 1,00 |

FONTE: O autor (2018).

GRÁFICO 7 – ERROS DE COMPILAÇÃO – EXPERIMENTO 4 – PRIMEIRA COLETA




FONTE: O autor (2018).

Os dados consolidados a respeito dos erros de lógica apresentados no artefato entregue estão descritos na Tabela 27. Nesta tabela é possível observar que todos os alunos entregaram o artefato com erros de lógica.

TABELA 27 – RESUMO DE ERROS DE LÓGICA – EXPERIMENTO 4 – PRIMEIRA COLETA

| Descrição | Alunos com ocorrência | % de alunos da turma |
|---|---|---|
| Erro de Lógica | 14 | 100,0 |
| Erro com operador lógico | 0 | 0,0 |
| Erro com operador relacional | 1 | 7,1 |
| Erro de ordem/sequência de comandos | 0 | 0,0 |
| Erro aritmético | 0 | 0,0 |
| Erro com estruturas de repetição | 14 | 100,0 |
| Erro com estruturas de decisão | 9 | 64,3 |

FONTE: O autor (2018).

O resumo das notas atribuídas está descrito na Tabela 28. Para esta coleta a média das notas da turma foi de 22,86.

TABELA 28 – RESUMO DE NOTAS – EXPERIMENTO 4 – PRIMEIRA COLETA

| Nota | Quantidade de Alunos | % de alunos da turma |
|---|---|---|
| Nota 100 | 0 | 0,0 |
| Nota 80 | 0 | 0,0 |
| Nota 60 | 1 | 7,1 |
| Nota 40 | 1 | 7,1 |
| Nota 20 | 11 | 78,6 |
| Nota 0 | 1 | 7,1 |

FONTE: O autor (2018).

### 4.5.2 Resultados da segunda coleta – experimento 4

Os dados consolidados da segunda coleta para o experimento 4, a respeito das compilações e execuções, estão apresentados na Tabela 29. Com base nos dados apresentados é possível observar que a média de compilações realizadas para esta coleta foi de 12,29, sendo que a média de compilações com falha foi de 2,00 e a média de erros por compilação de 1,71. Também é possível verificar que 28,6% dos alunos do experimento repetiram erros entre as compilações, sendo que para estes a média de erros repetidos foi de 2,00. Ainda, 13 dos 14 alunos (92,86%) tiveram a última compilação com sucesso, ou seja, o artefato entregue não apresentou erros de compilação.

Em comparação com a primeira coleta para o experimento 4, observa-se que na segunda coleta houve uma menor média de compilações realizadas (-7,21), menor média de compilações com falha (-2,79) e menor média de erros por

compilação (-0,27). Na segunda coleta houve um menor número de alunos que repetiram erros entre as compilações (-14,3% da turma), e para os que repetiram erros o número de erros repetidos diminuiu (-11,33). Finalmente, o percentual de alunos que entregaram o artefato sem erros de compilação foi maior na segunda coleta (+21,43% da turma).

TABELA 29 – RESUMO DE COMPILAÇÕES – EXPERIMENTO 4 – SEGUNDA COLETA

| Descrição | Alunos com ocorrência | % Alunos com ocorrência | Média da turma | Média – alunos com ocorrência |
|---|---|---|---|---|
| Execuções realizadas | 13 | 92,9 | 9,00 | 9,69 |
| Compilações realizadas | 14 | 100,0 | 12,29 | 12,29 |
| Compilações com sucesso | 13 | 92,9 | 7,79 | 8,38 |
| Compilações com falha | 10 | 71,4 | 2,00 | 2,80 |
| Compilações sem efeito | 9 | 64,3 | 2,50 | 3,89 |
| Média de erros | 10 | - | 1,77 | 2,48 |
| Quantidade total de erros | 10 | - | 4,21 | 5,90 |
| Erros repetidos entre compilações | 4 | 28,6 | 0,57 | 2,00 |
| Última compilação com sucesso | 13 | 92,86 | - | - |

FONTE: O autor (2018).

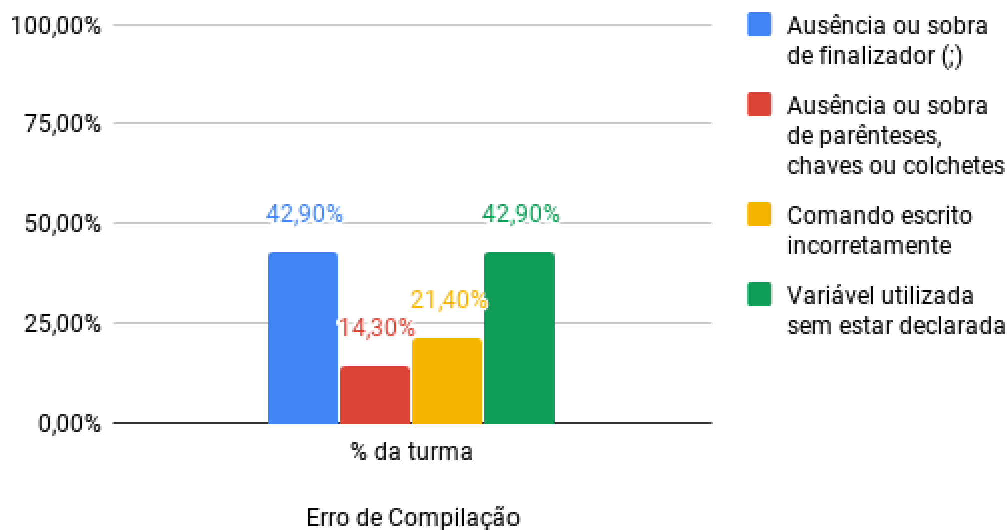
Os dados consolidados a respeito da classificação dos erros de compilação estão apresentados na Tabela 30 e ilustrados no Gráfico 8.

TABELA 30 – CLASSIFICAÇÃO DOS ERROS DE COMPILAÇÃO – EXPERIMENTO 4 – SEGUNDA COLETA

| Classificação | Alunos com ocorrência | % Alunos com ocorrência | Média da turma | Média – alunos com ocorrência |
|---|---|---|---|---|
| Ausência ou sobra de finalizador (;) | 6 | 42,9 | 0,86 | 2,00 |
| Ausência ou sobra de parênteses, chaves ou colchetes | 2 | 14,3 | 0,14 | 1,00 |
| Comando escrito incorretamente (sintaxe incorreta ou parâmetros incorretos) | 3 | 21,4 | 0,71 | 3,33 |
| Variável utilizada sem estar declarada | 6 | 42,9 | 1,57 | 3,67 |
| Variável declarada incorretamente ou valor inicial atribuído incorretamente | 1 | 7,1 | 0,07 | 1,00 |
| Variável declarada, porém, utilizada com o nome escrito incorretamente | 2 | 14,3 | 0,29 | 2,00 |
| Tipo de dados incorreto ou inexistente | 1 | 7,1 | 0,57 | 8,00 |
| Comando/expressão incompleta (else sem if, while sem do, falta de parâmetros) | 0 | 0,0 | 0,00 | 0,00 |
| Conectivo de comparação incorreto | 0 | 0,0 | 0,00 | 0,00 |
| Atribuição de valores incorreta | 0 | 0,0 | 0,00 | 0,00 |

FONTE: O autor (2018).

GRÁFICO 8 – ERROS DE COMPILAÇÃO – EXPERIMENTO 4 – SEGUNDA COLETA




FONTE: O autor (2018).

De acordo com os dados apresentados é possível observar que 42,9% dos alunos cometeram, ao menos uma vez, o erro de ausência ou sobra de finalizador (;), sendo que a média deste erro para estes alunos foi de 2,00. Outro erro cometido por também 42,9% dos alunos foi o de variável utilizada sem estar declarada, sendo que a média deste erro para estes alunos foi de 3,67.

Os dados consolidados a respeito dos erros de lógica apresentados no artefato entregue estão descritos na Tabela 31. Nesta tabela é possível observar que 35,7% dos alunos entregaram o artefato com erros de lógica, sendo que o problema estava no uso de estruturas de repetição. Em comparação com a primeira coleta do experimento 4 é possível observar que houve uma diminuição dos alunos que entregaram os artefatos com erro de lógica (-64,3% da turma).

TABELA 31 – RESUMO DE ERROS DE LÓGICA – EXPERIMENTO 4 – SEGUNDA COLETA

| Descrição | Alunos com ocorrência | % de alunos da turma |
|---|---|---|
| Erro de Lógica | 5 | 35,7 |
| Erro com operador lógico | 0 | 0,0 |
| Erro com operador relacional | 0 | 0,0 |
| Erro de ordem/sequência de comandos | 0 | 0,0 |
| Erro aritmético | 0 | 0,0 |
| Erro com estruturas de repetição | 4 | 28,6 |
| Erro com estruturas de decisão | 1 | 7,1 |

FONTE: O autor (2018).

O resumo das notas atribuídas está descrito na Tabela 32. Para esta coleta a média das notas da turma foi de 75,71. Em comparação com a primeira coleta do experimento 4 foi possível observar uma melhora considerável na média da turma (+52,85).

TABELA 32 – RESUMO DE NOTAS – EXPERIMENTO 4 – SEGUNDA COLETA

| Nota | Quantidade de Alunos | % de alunos da turma |
|---|---|---|
| Nota 100 | 8 | 57,1 |
| Nota 80 | 1 | 7,1 |
| Nota 60 | 2 | 14,3 |
| Nota 40 | 1 | 7,1 |
| Nota 20 | 1 | 7,1 |
| Nota 0 | 1 | 7,1 |

FONTE: O autor (2018).

## 4.6 EXPERIMENTO 5

O experimento 5 foi realizado com uma turma de uma universidade pública, sendo aplicado a uma turma com 8 alunos do curso de Engenharia Mecânica, matriculados na disciplina de Computação 2, disciplina pertencente a 2ª série do referido curso. Ambas as coletas foram realizadas diretamente pelo professor da disciplina, sem a participação direta dos pesquisadores.

Os alunos desta turma utilizaram durante as aulas a linguagem C para aprender programação para computadores e utilizaram a IDE Code::Blocks. O experimento foi aplicado no primeiro semestre do ano de 2018.

Para este experimento foi utilizado o laboratório da instituição de ensino em que estão matriculados, sendo que os alunos puderam utilizar os computadores do laboratório e também seus próprios notebooks, orientados à não utilizar a Internet. O *plugin* foi instalado na IDE Code::Blocks pelos alunos antes de iniciar o exercício. Ao término do exercício cada aluno enviou o programa e o arquivo XML gerado durante a resolução do exercício.

Conforme proposto no método, foram realizadas duas coletas. Para a primeira coleta foi aplicado o exercício 1 (ver seção 4.1.1), de estruturas de repetição e para a segunda coleta foi aplicado o exercício 2 (ver seção 4.1.2), de vetores. Após a primeira coleta foi realizada a devolutiva aos alunos a respeito do

desempenho nesta, esta devolutiva foi realizada antes da segunda coleta de dados. A seguir serão relatados os resultados destas coletas.

### 4.6.1 Resultados da primeira coleta – experimento 5

Os dados consolidados da primeira coleta para o experimento 5, a respeito das compilações e execuções, estão apresentados na Tabela 33. Com base nos dados apresentados é possível observar que a média de compilações realizadas para esta coleta foi de 14,50, sendo que a média de compilações com falha foi de 1,88 e a média de erros por compilação de 1,01. Também é possível verificar que 37,5% dos alunos do experimento repetiram erros entre as compilações, sendo que para estes, a média de erros repetidos foi de 3,00. Ainda, 100% dos alunos tiveram a última compilação com sucesso, ou seja, o artefato entregue não apresentou erros de compilação.

TABELA 33 – RESUMO DE COMPILAÇÕES – EXPERIMENTO 5 – PRIMEIRA COLETA

| Descrição | Alunos com ocorrência | % Alunos com ocorrência | Média da turma | Média – alunos com ocorrência |
|---|---|---|---|---|
| Execuções realizadas | 8 | 100,0 | 12,50 | 12,50 |
| Compilações realizadas | 8 | 100,0 | 14,50 | 14,50 |
| Compilações com sucesso | 8 | 100,0 | 11,13 | 11,13 |
| Compilações com falha | 5 | 62,5 | 1,88 | 3,00 |
| Compilações sem efeito | 4 | 50,0 | 1,50 | 3,00 |
| Média de erros | 5 | - | 1,01 | 1,61 |
| Quantidade total de erros | 4 | - | 3,13 | 6,25 |
| Erros repetidos entre compilações | 3 | 37,5 | 1,13 | 3,00 |
| Última compilação com sucesso | 8 | 100,00 | - | - |

FONTE: O autor (2018).

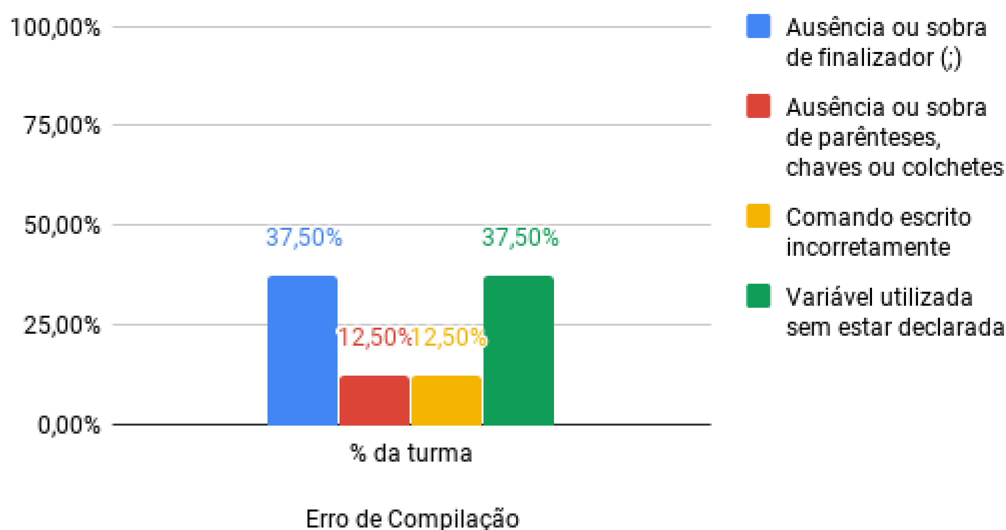
Os dados consolidados a respeito da classificação dos erros de compilação estão apresentados na Tabela 34 e ilustrados no Gráfico 9. De acordo com os dados apresentados é possível observar que 37,5% dos alunos cometeram, ao menos uma vez, o erro de ausência ou sobra de finalizador (;), sendo que a média deste erro para estes alunos foi de 5,00. Outro erro cometido por 37,5% destes alunos foi o de variável utilizada sem estar declarada, sendo que a média deste erro para estes alunos foi de 1,33.

TABELA 34 – CLASSIFICAÇÃO DOS ERROS DE COMPILAÇÃO – EXPERIMENTO 5 – PRIMEIRA COLETA

| Classificação | Alunos com ocorrência | % Alunos com ocorrência | Média da turma | Média – alunos com ocorrência |
|---|---|---|---|---|
| Ausência ou sobra de finalizador (;) | 3 | 37,5 | 1,88 | 5,00 |
| Ausência ou sobra de parênteses, chaves ou colchetes | 1 | 12,5 | 0,38 | 3,00 |
| Comando escrito incorretamente (sintaxe incorreta ou parâmetros incorretos) | 1 | 12,5 | 0,25 | 2,00 |
| Variável utilizada sem estar declarada | 3 | 37,5 | 0,50 | 1,33 |
| Variável declarada incorretamente ou valor inicial atribuído incorretamente | 1 | 12,5 | 0,13 | 1,00 |
| Variável declarada, porém, utilizada com o nome escrito incorretamente | 0 | 0,0 | 0,00 | 0,00 |
| Tipo de dados incorreto ou inexistente | 0 | 0,0 | 0,00 | 0,00 |
| Comando/expressão incompleta (else sem if, while sem do, falta de parâmetros) | 0 | 0,0 | 0,00 | 0,00 |
| Conectivo de comparação incorreto | 0 | 0,0 | 0,00 | 0,00 |
| Atribuição de valores incorreta | 0 | 0,0 | 0,00 | 0,00 |

FONTE: O autor (2018).

GRÁFICO 9 – ERROS DE COMPILAÇÃO – EXPERIMENTO 5 – PRIMEIRA COLETA




FONTE: O autor (2018).

Os dados consolidados a respeito dos erros de lógica apresentados no artefato entregue estão descritos na Tabela 35. Nesta tabela é possível observar que apenas um aluno entregou o artefato com erro de lógica.

TABELA 35 – RESUMO DE ERROS DE LÓGICA – EXPERIMENTO 5 – PRIMEIRA COLETA

| Descrição | Alunos com ocorrência | % de alunos da turma |
|---|---|---|
| Erro de Lógica | 1 | 12,5 |
| Erro com operador lógico | 0 | 0,0 |
| Erro com operador relacional | 0 | 0,0 |
| Erro de ordem/sequência de comandos | 0 | 0,0 |
| Erro aritmético | 1 | 12,5 |
| Erro com estruturas de repetição | 0 | 0,0 |
| Erro com estruturas de decisão | 0 | 0,0 |

FONTE: O autor (2018).

O resumo das notas atribuídas está descrito na Tabela 36. Para esta coleta a média das notas da turma foi de 95,00. Grande parte dos alunos (87,5%) obtiveram 100 pontos.

TABELA 36 – RESUMO DE NOTAS – EXPERIMENTO 5 – PRIMEIRA COLETA

| Nota | Quantidade de Alunos | % de alunos da turma |
|---|---|---|
| Nota 100 | 7 | 87,5 |
| Nota 80 | 0 | 0,0 |
| Nota 60 | 1 | 12,5 |
| Nota 40 | 0 | 0,0 |
| Nota 20 | 0 | 0,0 |
| Nota 0 | 0 | 0,0 |

FONTE: O autor (2018).

### 4.6.2 Resultados da segunda coleta – experimento 5

Os dados consolidados da segunda coleta para o experimento 5, a respeito das compilações e execuções, estão apresentados na Tabela 37. Com base nos dados apresentados é possível observar que a média de compilações realizadas para esta coleta foi de 25,13, sendo que a média de compilações com falha foi de 1,75 e a média de erros por compilação de 1,10. Também é possível verificar que 37,5% dos alunos do experimento repetiram erros entre as compilações, sendo que para estes a média de erros repetidos foi de 1,67. Ainda, 100% dos alunos tiveram a última compilação com sucesso, ou seja, o artefato entregue não apresentou erros de compilação.

Em comparação com a primeira coleta para o experimento 5, observa-se que na segunda coleta houve uma maior média de compilações realizadas (+10,63), menor média de compilações com falha (-0,13) e menor média de erros por

compilação (-0,14). Na segunda coleta o número de alunos que repetiram erros entre as compilações foi o mesmo em comparação com a primeira e, para os que repetiram erros, o número de erros repetidos diminuiu (-1,33).

TABELA 37 – RESUMO DE COMPILAÇÕES – EXPERIMENTO 5 – SEGUNDA COLETA

| Descrição | Alunos com ocorrência | % Alunos com ocorrência | Média da turma | Média – alunos com ocorrência |
|---|---|---|---|---|
| Execuções realizadas | 8 | 100,0 | 22,00 | 22,00 |
| Compilações realizadas | 8 | 100,0 | 25,13 | 25,13 |
| Compilações com sucesso | 8 | 100,0 | 18,25 | 18,25 |
| Compilações com falha | 6 | 75,0 | 1,75 | 2,33 |
| Compilações sem efeito | 7 | 87,5 | 5,13 | 5,86 |
| Média de erros | 6 | - | 1,10 | 1,47 |
| Quantidade total de erros | 6 | - | 2,50 | 3,33 |
| Erros repetidos entre compilações | 3 | 37,5 | 0,63 | 1,67 |
| Última compilação com sucesso | 8 | 100,0 | - | - |

FONTE: O autor (2018).

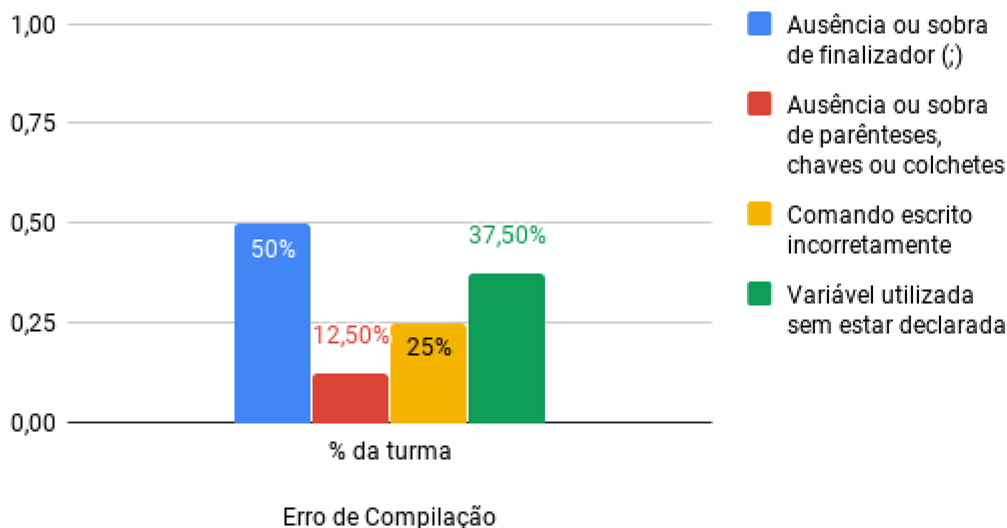
Os dados consolidados a respeito da classificação dos erros de compilação estão apresentados na Tabela 38 e ilustrados no Gráfico 10.

TABELA 38 – CLASSIFICAÇÃO DOS ERROS DE COMPILAÇÃO – EXPERIMENTO 5 – SEGUNDA COLETA

| Classificação | Alunos com ocorrência | % Alunos com ocorrência | Média da turma | Média – alunos com ocorrência |
|---|---|---|---|---|
| Ausência ou sobra de finalizador (;) | 4 | 50,0 | 0,63 | 1,25 |
| Ausência ou sobra de parênteses, chaves ou colchetes | 1 | 12,5 | 0,50 | 4,00 |
| Comando escrito incorretamente (sintaxe incorreta ou parâmetros incorretos) | 2 | 25,0 | 0,63 | 2,50 |
| Variável utilizada sem estar declarada | 3 | 37,5 | 0,63 | 1,67 |
| Variável declarada incorretamente ou valor inicial atribuído incorretamente | 0 | 0,0 | 0,00 | 0,00 |
| Variável declarada, porém, utilizada com o nome escrito incorretamente | 0 | 0,0 | 0,00 | 0,00 |
| Tipo de dados incorreto ou inexistente | 0 | 0,0 | 0,00 | 0,00 |
| Comando/expressão incompleta (else sem if, while sem do, falta de parâmetros) | 0 | 0,0 | 0,00 | 0,00 |
| Conectivo de comparação incorreto | 1 | 12,5 | 0,13 | 1,00 |
| Atribuição de valores incorreta | 0 | 0,0 | 0,00 | 0,00 |

FONTE: O autor (2018).

GRÁFICO 10 – ERROS DE COMPILAÇÃO – EXPERIMENTO 5 – SEGUNDA COLETA




FONTE: O autor (2018).

De acordo com os dados apresentados é possível observar que 50% dos alunos cometeram, ao menos uma vez, o erro de ausência ou sobra de finalizador (;), sendo que a média deste erro para estes alunos foi de 1,25. Outro erro cometido por 37,5% dos alunos foi o de variável utilizada sem estar declarada, sendo que a média deste erro para estes alunos foi de 1,67.

Os dados consolidados a respeito dos erros de lógica apresentados no artefato entregue estão descritos na Tabela 39. Nesta tabela é possível observar que nenhum aluno entregou o artefato com erros de lógica.

TABELA 39 – RESUMO DE ERROS DE LÓGICA – EXPERIMENTO 5 – SEGUNDA COLETA

| Descrição | Alunos com ocorrência | % de alunos da turma |
|---|---|---|
| Erro de Lógica | 0 | 0,0 |
| Erro com operador lógico | 0 | 0,0 |
| Erro com operador relacional | 0 | 0,0 |
| Erro de ordem/sequência de comandos | 0 | 0,0 |
| Erro aritmético | 0 | 0,0 |
| Erro com estruturas de repetição | 0 | 0,0 |
| Erro com estruturas de decisão | 0 | 0,0 |

FONTE: O autor (2018).

O resumo das notas atribuídas está descrito na Tabela 40. Para esta coleta a média das notas da turma foi de 100,00. Em comparação com a primeira coleta do experimento 5 foi possível observar uma melhora considerável na média da turma (+5,00).

TABELA 40 – RESUMO DE NOTAS – EXPERIMENTO 5 – SEGUNDA COLETA

| Nota | Quantidade de Alunos | % de alunos da turma |
|---|---|---|
| Nota 100 | 8 | 100,0 |
| Nota 80 | 0 | 0,0 |
| Nota 60 | 0 | 0,0 |
| Nota 40 | 0 | 0,0 |
| Nota 20 | 0 | 0,0 |
| Nota 0 | 0 | 0,0 |

FONTE: O autor (2018).

## 4.7 CORRELAÇÃO DADOS COLETADOS X DESEMPENHO ACADÊMICO

Conforme previso no método proposto, para os experimentos realizados neste trabalho foi investigada a correlação entre alguns dos dados coletados em plano de fundo durante a solução dos exercícios, como variável independente, e o desempenho acadêmico atribuído ao exercício, como variável dependente. Os dados escolhidos para investigar a correlação são: a) número de compilações com falha; b) número de compilações com sucesso; c) média de erros por compilação); e d) número total de erros.

A investigação da correlação foi realizada através do cálculo do coeficiente de correlação e do mapa de dispersão. Os resultados dos testes para os experimentos realizados estão relatados nas próximas subseções.

### 4.7.1 Investigação de correlação – experimento 1

Os dados coletados no experimento 1 e a nota atribuída aos exercícios foram utilizados para calcular o coeficiente de correlação amostral. Os resultados para a primeira e para a segunda coleta do experimento 1 estão listados no Quadro 13.

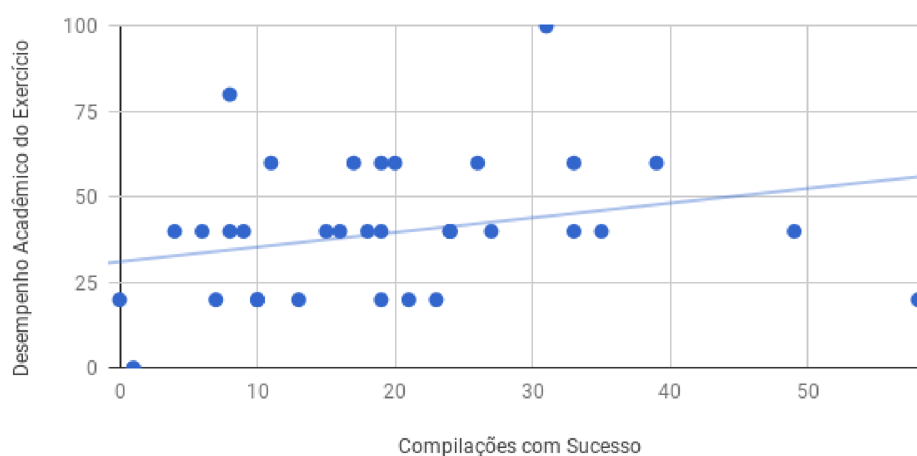
QUADRO 13 – COEFICIENTE DE CORRELAÇÃO – EXPERIMENTO 1

| Variável independente | Coeficiente de Correlação – Primeira Coleta | Coeficiente de Correlação – Segunda Coleta |
|---|---|---|
| Quantidade de compilações com sucesso | 0,27 | 0,12 |
| Quantidade de compilações com falha | -0,27 | -0,42 |
| Média de erros por compilação | -0,19 | -0,15 |
| Número total de erros de compilação | -0,33 | -0,36 |

FONTE: O autor (2018).

Conforme é possível observar no Quadro 13, existe uma correlação linear positiva entre a quantidade de compilações com sucesso e o desempenho acadêmico atribuído ao exercício. Para a primeira coleta ($r$ = 0,27) a correlação é considerada fraca e para a segunda coleta ($r$ = 0,12) é considerada muito fraca. O mapa de dispersão da correlação entre a quantidade de compilações com sucesso e o desempenho acadêmico está ilustrado no Gráfico 11 para a primeira coleta e no Gráfico 12 para a segunda coleta.

GRÁFICO 11 – CORRELAÇÃO ENTRE COMPILAÇÕES COM SUCESSO E DESEMPENHO ACADÊMICO DO EXERCÍCIO – EXPERIMENTO 1 – PRIMEIRA COLETA



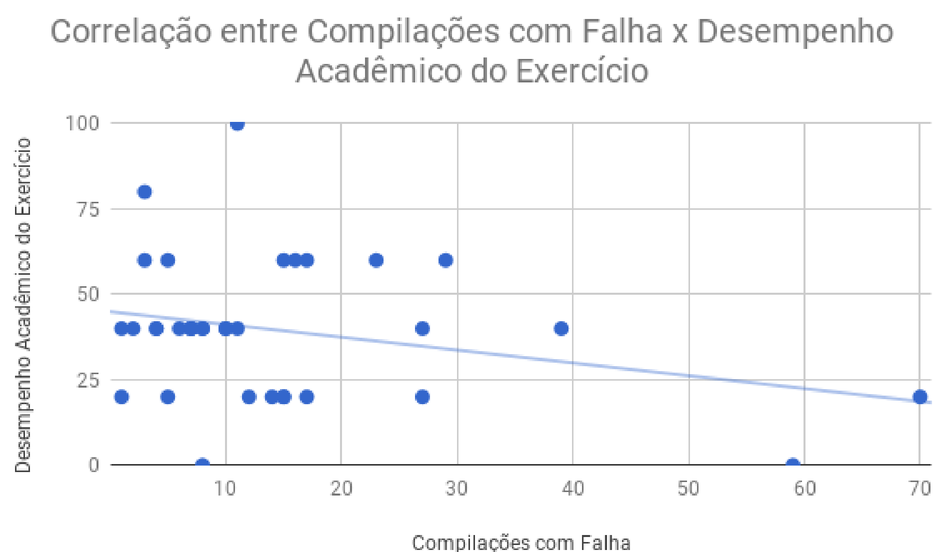

FONTE: O autor (2018).

GRÁFICO 12 – CORRELAÇÃO ENTRE COMPILAÇÕES COM SUCESSO E DESEMPENHO ACADÊMICO DO EXERCÍCIO – EXPERIMENTO 1 – SEGUNDA COLETA

FONTE: O autor (2018).

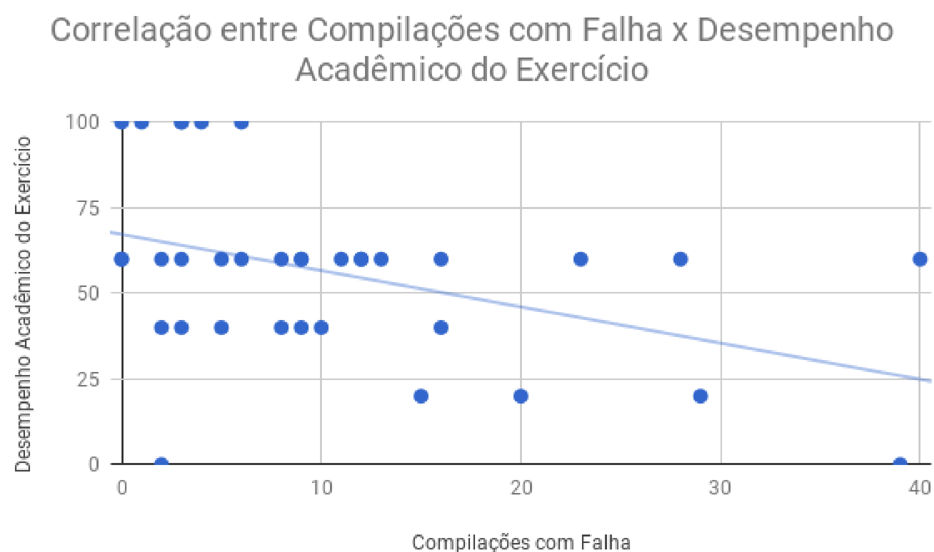
Também é possível observar que existe uma correlação linear negativa entre a quantidade de compilações com falha e o desempenho acadêmico atribuído ao exercício, sendo fraca para a primeira coleta ($r$ = -0,27) e moderada para a segunda coleta ($r$ = -0,42). O mapa de dispersão da correlação entre a quantidade de compilações com falha e o desempenho acadêmico está ilustrado no Gráfico 13 para a primeira coleta e no Gráfico 14 para a segunda coleta.

GRÁFICO 13 – CORRELAÇÃO ENTRE COMPILAÇÕES COM FALHA E DESEMPENHO ACADÊMICO DO EXERCÍCIO – EXPERIMENTO 1 – PRIMEIRA COLETA




FONTE: O autor (2018).

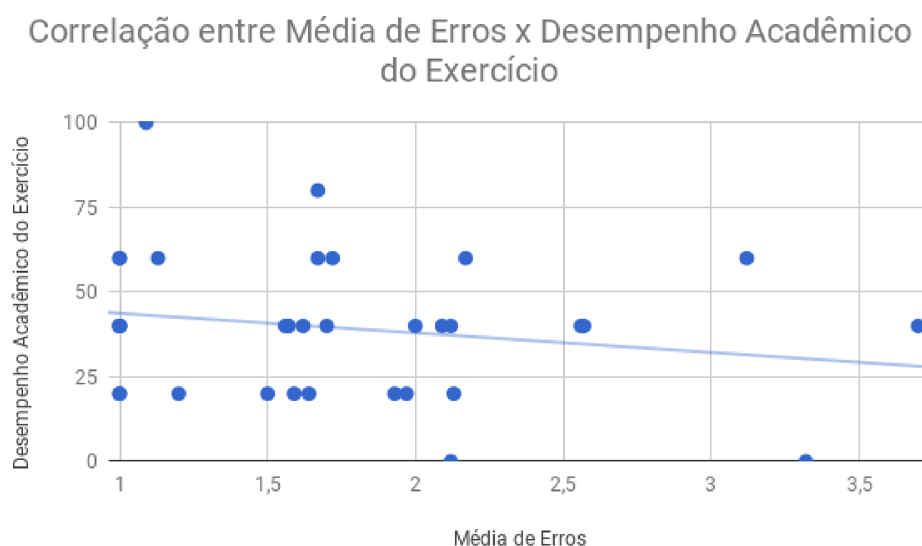
GRÁFICO 14 – CORRELAÇÃO ENTRE COMPILAÇÕES COM FALHA E DESEMPENHO ACADÊMICO DO EXERCÍCIO – EXPERIMENTO 1 – SEGUNDA COLETA




FONTE: O autor (2018).

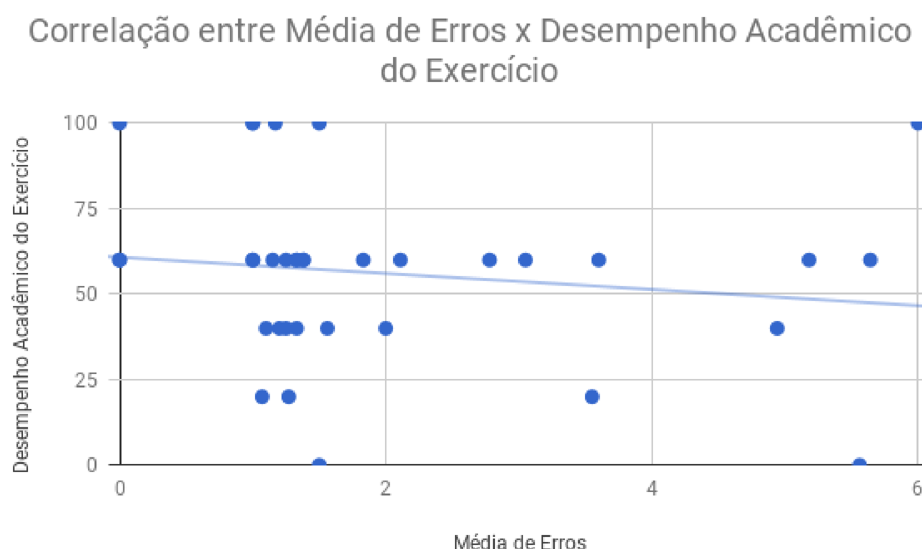
Ainda, é possível observar que existe uma correlação linear negativa muito fraca entre a quantidade média de erros de compilação e o desempenho acadêmico atribuído ao exercício, tanto para a primeira coleta (*r* = -0,19) quanto para a segunda coleta (*r* = -0,15). O mapa de dispersão da correlação entre a média de erros e o desempenho acadêmico está ilustrado no Gráfico 15 para a primeira coleta e no Gráfico 16 para a segunda coleta.

GRÁFICO 15 – CORRELAÇÃO ENTRE A MÉDIA DE ERROS DE COMPILAÇÕES E DESEMPENHO ACADÊMICO DO EXERCÍCIO – EXPERIMENTO 1 – PRIMEIRA COLETA




FONTE: O autor (2018).

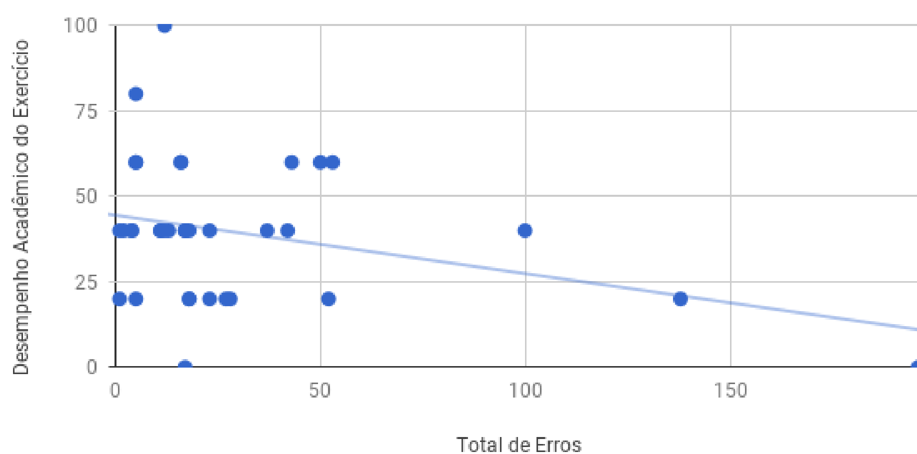
GRÁFICO 16 – CORRELAÇÃO ENTRE A MÉDIA DE ERROS DE COMPILAÇÕES E DESEMPENHO ACADÊMICO DO EXERCÍCIO – EXPERIMENTO 1 – SEGUNDA COLETA




FONTE: O autor (2018).

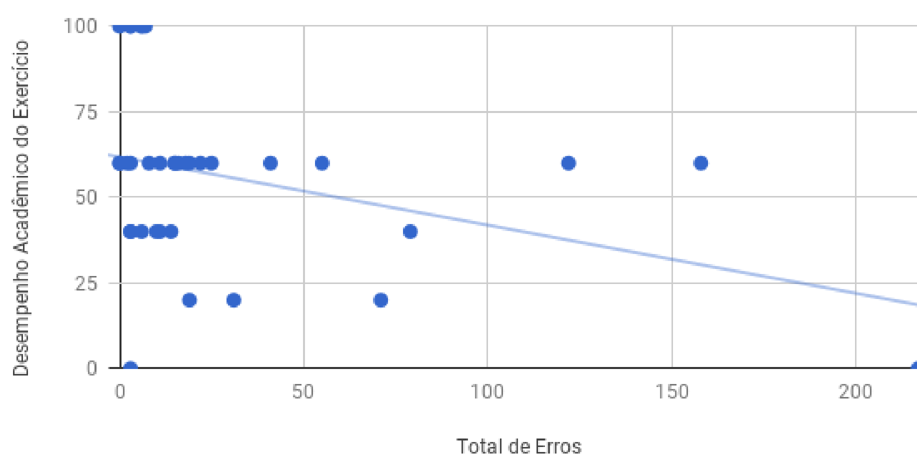
Finalmente, é possível observar que existe uma correlação linear negativa fraca entre a quantidade total de erros de compilação e o desempenho acadêmico atribuído ao exercício, tanto para a primeira coleta (*r* = -0,33) quanto para a segunda coleta (*r* = -0,36). O mapa de dispersão da correlação entre o total de erros e o desempenho acadêmico está ilustrado no Gráfico 17 para a primeira coleta e no Gráfico 18 para a segunda coleta.

GRÁFICO 17 – CORRELAÇÃO ENTRE O TOTAL DE ERROS DE COMPILAÇÕES E DESEMPENHO ACADÊMICO DO EXERCÍCIO – EXPERIMENTO 1 – PRIMEIRA COLETA




FONTE: O autor (2018).

GRÁFICO 18 – CORRELAÇÃO ENTRE O TOTAL DE ERROS DE COMPILAÇÕES E DESEMPENHO ACADÊMICO DO EXERCÍCIO – EXPERIMENTO 1 – SEGUNDA COLETA




FONTE: O autor (2018).

### 4.7.2 Investigação de correlação – experimento 2

Os dados coletados no experimento 2 e a nota atribuída aos exercícios foram utilizados para calcular o coeficiente de correlação amostral. Os resultados para a primeira e para a segunda coleta do experimento 2 estão listados no Quadro 14.

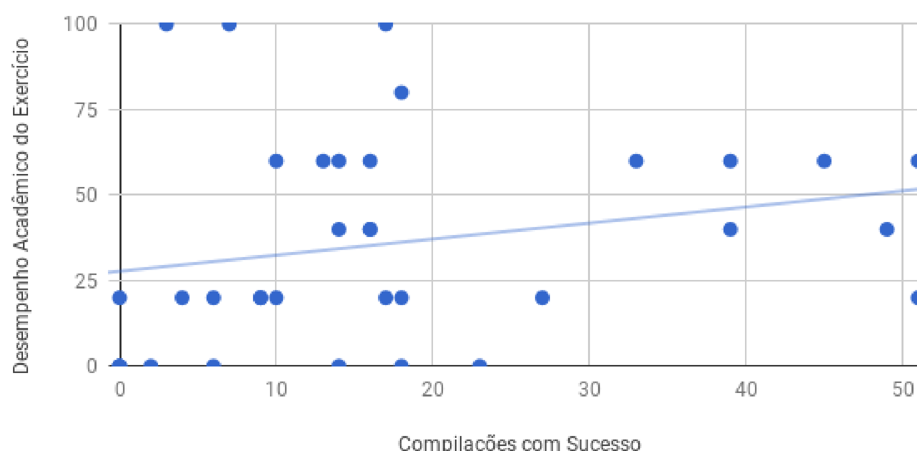
QUADRO 14 – COEFICIENTE DE CORRELAÇÃO – EXPERIMENTO 2

| Variável independente | Coeficiente de Correlação – Primeira Coleta | Coeficiente de Correlação – Segunda Coleta |
|---|---|---|
| Quantidade de compilações com sucesso | 0,22 | -0,19 |
| Quantidade de compilações com falha | -0,37 | -0,41 |
| Média de erros por compilação | -0,31 | -0,16 |
| Número total de erros de compilação | -0,37 | -0,19 |

FONTE: O autor (2018).

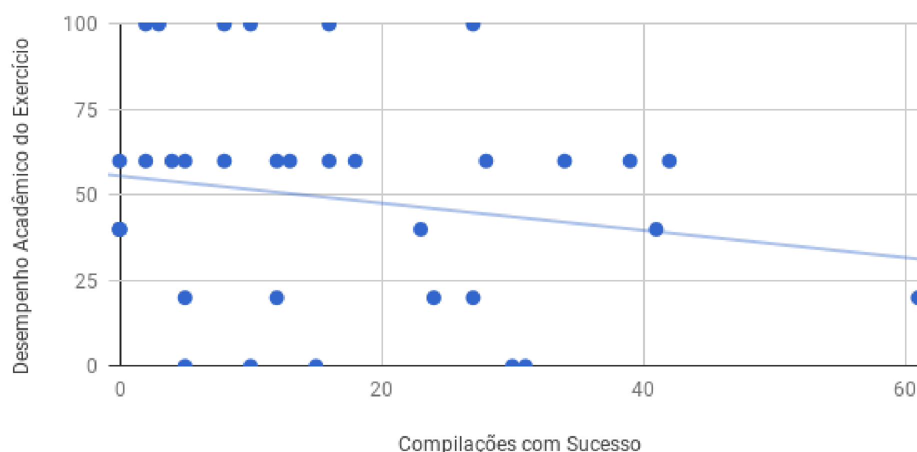
Conforme é possível observar no Quadro 14, existe uma correlação linear positiva fraca entre a quantidade de compilações com sucesso e o desempenho acadêmico atribuído ao exercício para a primeira coleta (*r* = 0,22) e uma correlação linear negativa muito fraca para a segunda coleta (*r* = -0,19). O mapa de dispersão da correlação entre a quantidade de compilações com sucesso e o desempenho acadêmico está ilustrado no Gráfico 19 para a primeira coleta e no Gráfico 20 para a segunda coleta.

GRÁFICO 19 – CORRELAÇÃO ENTRE COMPILAÇÕES COM SUCESSO E DESEMPENHO ACADÊMICO DO EXERCÍCIO – EXPERIMENTO 2 – PRIMEIRA COLETA




FONTE: O autor (2018).

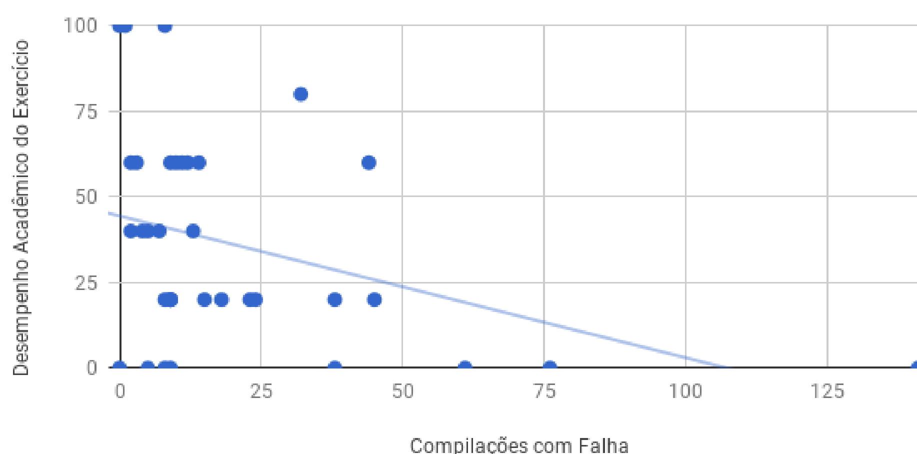
GRÁFICO 20 – CORRELAÇÃO ENTRE COMPILAÇÕES COM SUCESSO E DESEMPENHO ACADÊMICO DO EXERCÍCIO – EXPERIMENTO 2 – SEGUNDA COLETA




FONTE: O autor (2018).

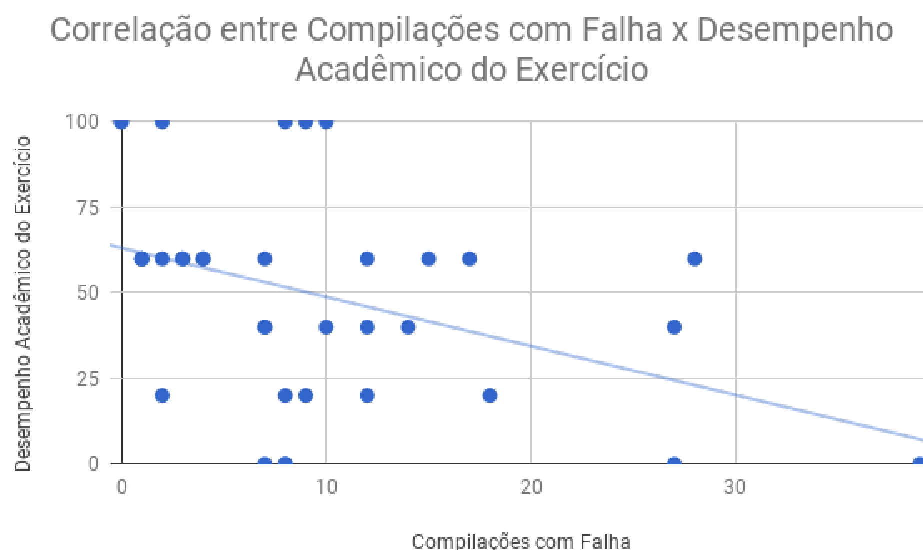
Também é possível observar que existe uma correlação linear negativa entre a quantidade de compilações com falha e o desempenho acadêmico atribuído ao exercício, considerada fraca para a primeira coleta ($r$ = -0,37) e moderada para a segunda coleta ($r$ = -0,41). O mapa de dispersão da correlação entre a quantidade de compilações com falha e o desempenho acadêmico está ilustrado no Gráfico 21 para a primeira coleta e no Gráfico 22 para a segunda coleta.

GRÁFICO 21 – CORRELAÇÃO ENTRE COMPILAÇÕES COM FALHA E DESEMPENHO ACADÊMICO DO EXERCÍCIO – EXPERIMENTO 2 – PRIMEIRA COLETA




FONTE: O autor (2018).

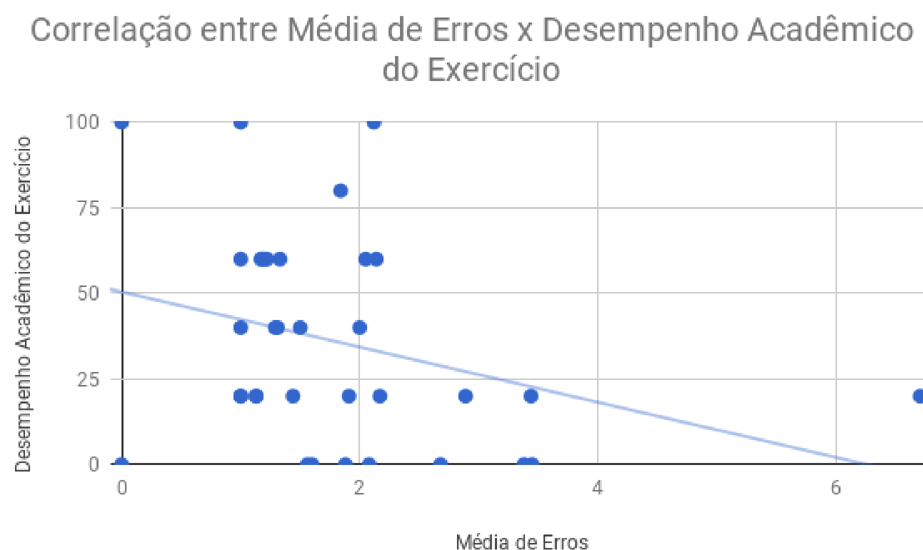
GRÁFICO 22 – CORRELAÇÃO ENTRE COMPILAÇÕES COM FALHA E DESEMPENHO ACADÊMICO DO EXERCÍCIO – EXPERIMENTO 2 – SEGUNDA COLETA




FONTE: O autor (2018).

Ainda, é possível observar que existe uma correlação linear negativa entre a quantidade média de erros de compilação e o desempenho acadêmico atribuído ao exercício, sendo fraca para a primeira coleta ($r$ = -0,31) e muito fraca para a segunda coleta ($r$ = -0,16). O mapa de dispersão da correlação entre a média de erros e o desempenho acadêmico está ilustrado no Gráfico 23 para a primeira coleta e no Gráfico 24 para a segunda coleta.

GRÁFICO 23 – CORRELAÇÃO ENTRE A MÉDIA DE ERROS DE COMPILAÇÕES E DESEMPENHO ACADÊMICO DO EXERCÍCIO – EXPERIMENTO 2 – PRIMEIRA COLETA




FONTE: O autor (2018).

GRÁFICO 24 – CORRELAÇÃO ENTRE A MÉDIA DE ERROS DE COMPILAÇÕES E DESEMPENHO ACADÊMICO DO EXERCÍCIO – EXPERIMENTO 2 – SEGUNDA COLETA

Correlação entre Média de Erros x Desempenho Acadêmico do Exercício

Desempenho Acadêmico do Exercício

100
75
50
25
0

0 2 4 6 8

Média de Erros

FONTE: O autor (2018).

Finalmente, é possível observar que existe uma correlação linear negativa entre a quantidade total de erros de compilação e o desempenho acadêmico atribuído ao exercício, sendo fraca para a primeira coleta (*r* = -0,37) e muito fraca para a segunda coleta (*r* = -0,19). O mapa de dispersão da correlação entre o total de erros e o desempenho acadêmico está ilustrado no Gráfico 25 para a primeira coleta e no Gráfico 26 para a segunda coleta.

GRÁFICO 25 – CORRELAÇÃO ENTRE O TOTAL DE ERROS DE COMPILAÇÕES E DESEMPENHO ACADÊMICO DO EXERCÍCIO – EXPERIMENTO 2 – PRIMEIRA COLETA

Correlação entre Total de Erros x Desempenho Acadêmico do Exercício

Desempenho Acadêmico do Exercício

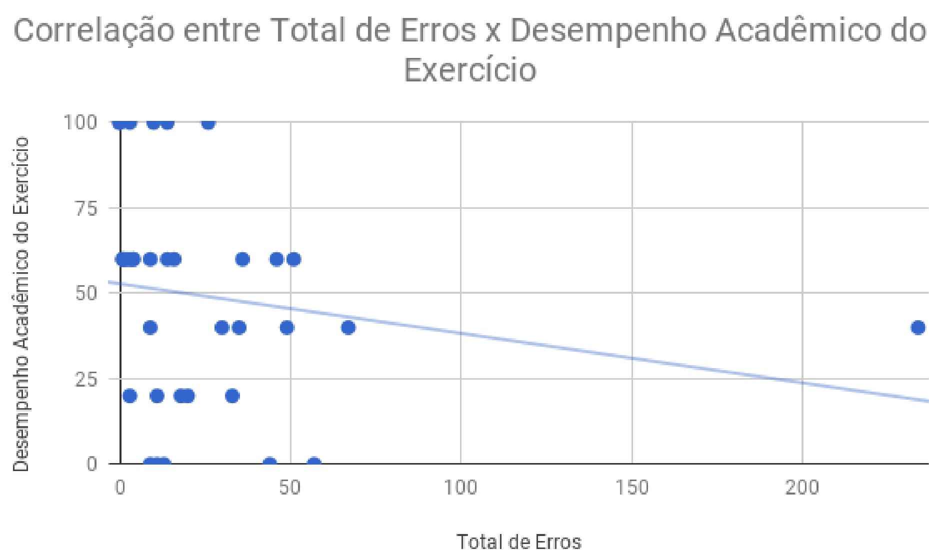
100
75
50
25
0

0 100 200 300

Total de Erros

FONTE: O autor (2018).

GRÁFICO 26 – CORRELAÇÃO ENTRE O TOTAL DE ERROS DE COMPILAÇÕES E DESEMPENHO ACADÊMICO DO EXERCÍCIO – EXPERIMENTO 2 – SEGUNDA COLETA




FONTE: O autor (2018).

### 4.7.3 Investigação de correlação – experimento 3

Os dados coletados no experimento 3 e a nota atribuída aos exercícios foram utilizados para calcular o coeficiente de correlação amostral. Os resultados para a primeira e para a segunda coleta do experimento 3 estão listados no Quadro 15. Devido ao baixo número de alunos participantes do experimento 3 (5 alunos) entende-se que não é necessário ilustrar o mapa de dispersão para este experimento.

QUADRO 15 – COEFICIENTE DE CORRELAÇÃO – EXPERIMENTO 3

| Variável independente | Coeficiente de Correlação – Primeira Coleta | Coeficiente de Correlação – Segunda Coleta |
|---|---|---|
| Quantidade de compilações com sucesso | 0,50 | -0,07 |
| Quantidade de compilações com falha | -0,06 | 0,27 |
| Média de erros por compilação | -0,49 | 0,43 |
| Número total de erros de compilação | -0,43 | 0,47 |

FONTE: O autor (2018).

Conforme é possível observar no Quadro 15, existe uma correlação linear positiva moderada entre a quantidade de compilações com sucesso e o desempenho acadêmico atribuído ao exercício para a primeira coleta ($r$ = 0,50) e não existe correlação para a segunda coleta ($r$ = -0,07).

Também é possível observar que não existe uma correlação linear entre a quantidade de compilações com falha e o desempenho acadêmico atribuído ao exercício para a primeira coleta (*r* = -0,06) e existe uma correlação linear positiva fraca para a segunda coleta (*r* = 0,27).

Ainda, é possível observar que existe uma correlação linear negativa moderada entre a quantidade média de erros de compilação e o desempenho acadêmico atribuído ao exercício para a primeira coleta (*r* = -0,49) e uma correlação linear positiva moderada para a segunda coleta (*r* = 0,43).

Finalmente, é possível observar que existe uma correlação linear negativa moderada entre a quantidade total de erros de compilação e o desempenho acadêmico atribuído ao exercício para a primeira coleta (*r* = -0,43) e uma correlação linear positiva moderada para a segunda coleta (*r* = 0,47).

### 4.7.4 Investigação de correlação – experimento 4

Os dados coletados no experimento 4 e a nota atribuída aos exercícios foram utilizados para calcular o coeficiente de correlação amostral. Os resultados para a primeira e para a segunda coleta do experimento 4 estão listados no Quadro 16.

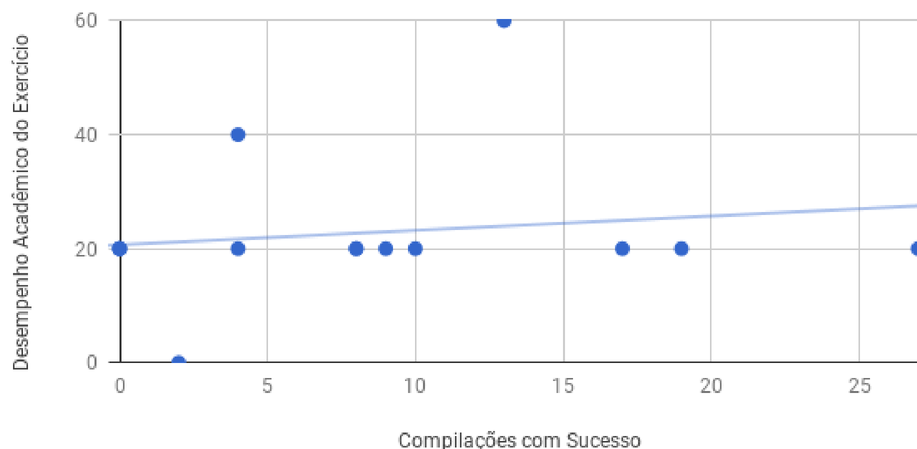
QUADRO 16 – COEFICIENTE DE CORRELAÇÃO – EXPERIMENTO 4

| Variável independente | Coeficiente de Correlação – Primeira Coleta | Coeficiente de Correlação – Segunda Coleta |
|---|---:|---:|
| Quantidade de compilações com sucesso | 0,15 | -0,12 |
| Quantidade de compilações com falha | -0,14 | -0,49 |
| Média de erros por compilação | -0,25 | -0,64 |
| Número total de erros de compilação | -0,10 | -0,83 |

FONTE: O autor (2018).

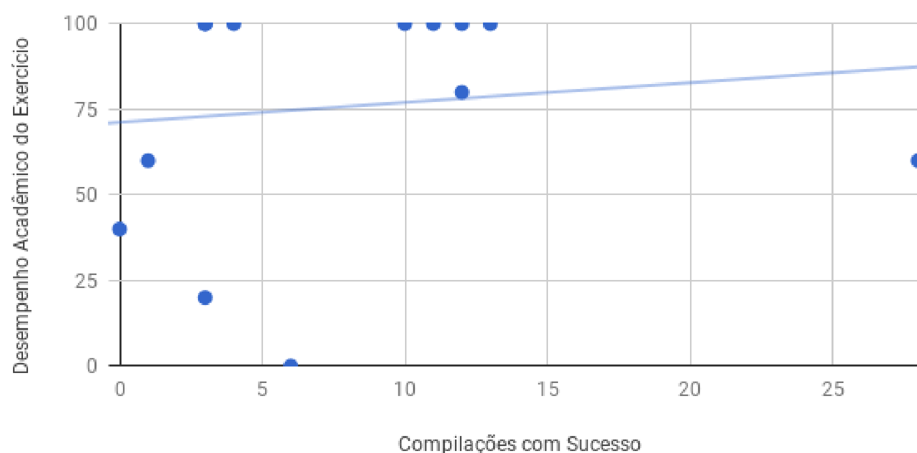
Conforme é possível observar no Quadro 16, existe uma correlação linear positiva muito fraca entre a quantidade de compilações com sucesso e o desempenho acadêmico atribuído ao exercício para a primeira coleta (*r* = 0,15) e uma correlação linear negativa muito fraca para a segunda coleta (*r* = -0,12). O mapa de dispersão da correlação entre a quantidade de compilações com sucesso e o desempenho acadêmico está ilustrado no Gráfico 27 para a primeira coleta e no Gráfico 28 para a segunda coleta.

GRÁFICO 27 – CORRELAÇÃO ENTRE COMPILAÇÕES COM SUCESSO E DESEMPENHO ACADÊMICO DO EXERCÍCIO – EXPERIMENTO 4 – PRIMEIRA COLETA




FONTE: O autor (2018).

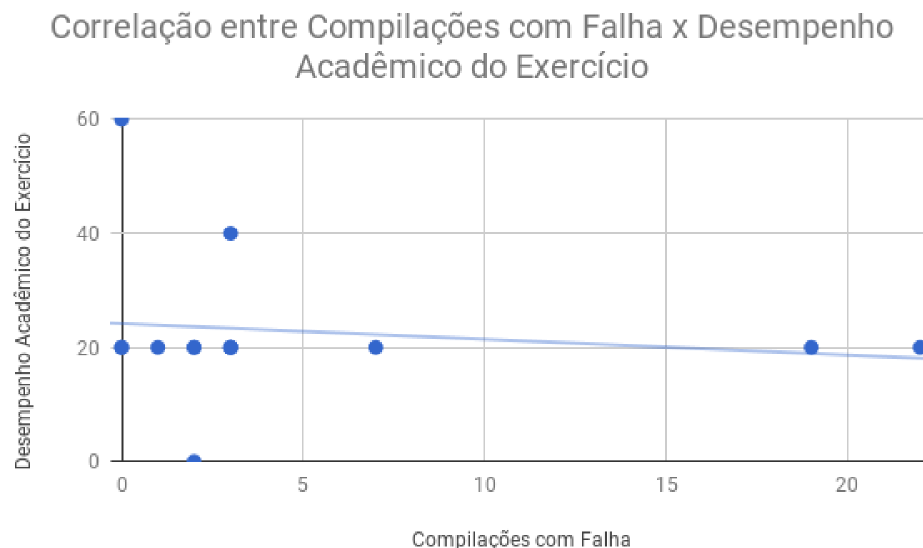
GRÁFICO 28 – CORRELAÇÃO ENTRE COMPILAÇÕES COM SUCESSO E DESEMPENHO ACADÊMICO DO EXERCÍCIO – EXPERIMENTO 4 – SEGUNDA COLETA




FONTE: O autor (2018).

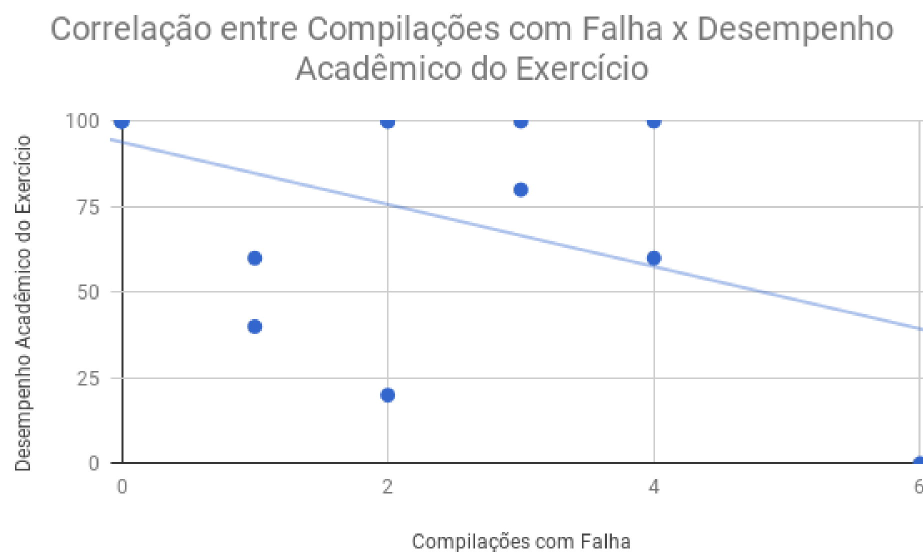
Também é possível observar que existe uma correlação linear negativa entre a quantidade de compilações com falha e o desempenho acadêmico atribuído ao exercício, sendo muito fraca para a primeira coleta (*r* = -0,14) e moderada para a segunda coleta (*r* = -0,49). O mapa de dispersão da correlação entre a quantidade de compilações com falha e o desempenho acadêmico está ilustrado no Gráfico 29 para a primeira coleta e no Gráfico 30 para a segunda coleta.

GRÁFICO 29 – CORRELAÇÃO ENTRE COMPILAÇÕES COM FALHA E DESEMPENHO ACADÊMICO DO EXERCÍCIO – EXPERIMENTO 4 – PRIMEIRA COLETA




FONTE: O autor (2018).

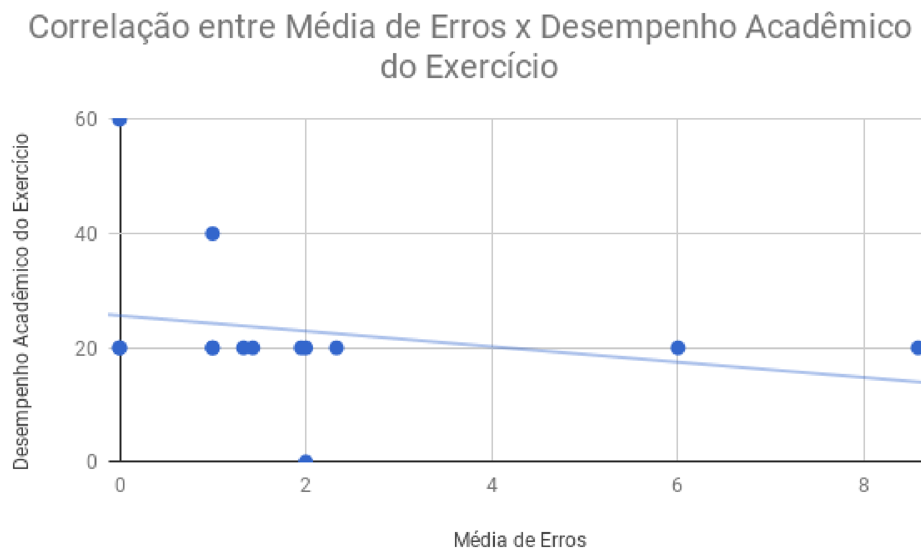
GRÁFICO 30 – CORRELAÇÃO ENTRE COMPILAÇÕES COM FALHA E DESEMPENHO ACADÊMICO DO EXERCÍCIO – EXPERIMENTO 4 – SEGUNDA COLETA




FONTE: O autor (2018).

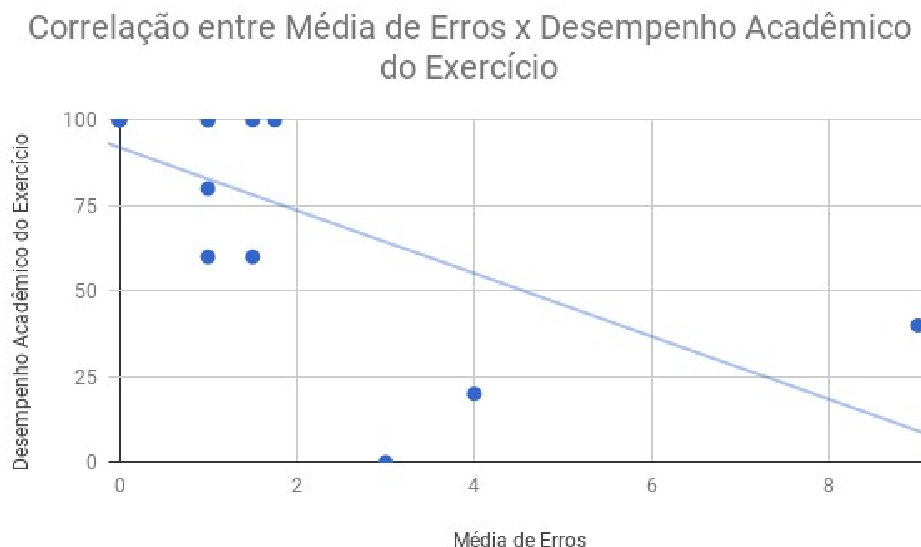
Ainda, é possível observar que existe uma correlação linear negativa entre a quantidade média de erros de compilação e o desempenho acadêmico atribuído ao exercício, sendo fraca para a primeira coleta ($r$ = -0,25) e forte para a segunda coleta ($r$ = -0,64). O mapa de dispersão da correlação entre a média de erros e o desempenho acadêmico está ilustrado no Gráfico 31 para a primeira coleta e no Gráfico 32 para a segunda coleta.

GRÁFICO 31 – CORRELAÇÃO ENTRE A MÉDIA DE ERROS DE COMPILAÇÕES E DESEMPENHO ACADÊMICO DO EXERCÍCIO – EXPERIMENTO 4 – PRIMEIRA COLETA




FONTE: O autor (2018).

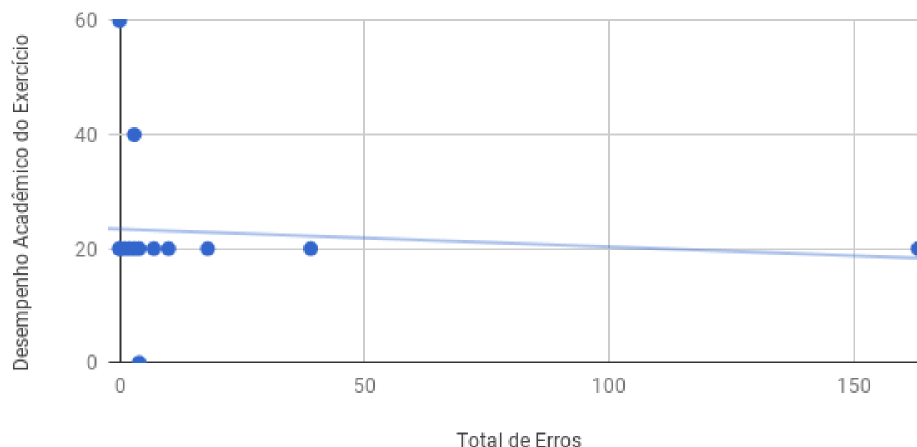
GRÁFICO 32 – CORRELAÇÃO ENTRE A MÉDIA DE ERROS DE COMPILAÇÕES E DESEMPENHO ACADÊMICO DO EXERCÍCIO – EXPERIMENTO 4 – SEGUNDA COLETA




FONTE: O autor (2018).

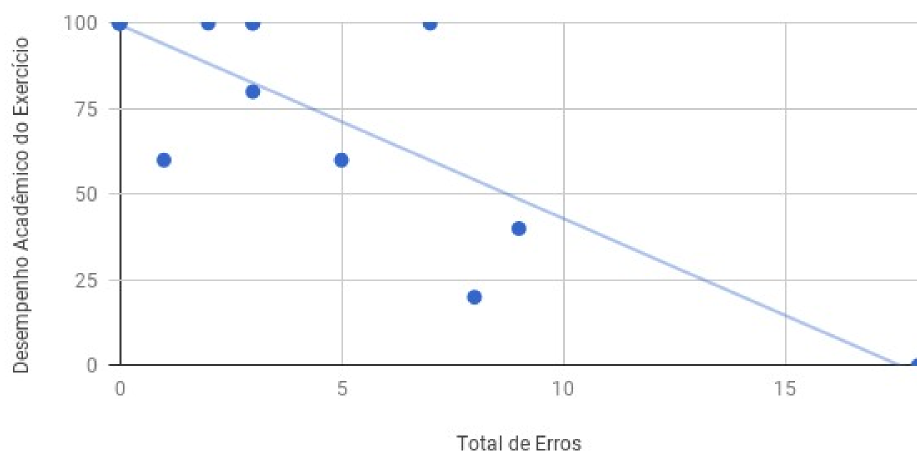
Finalmente, é possível observar que existe uma correlação linear negativa entre a quantidade total de erros de compilação e o desempenho acadêmico atribuído ao exercício, sendo muito fraca para a primeira coleta ($r$ = -0,10) e muito forte para a segunda coleta ($r$ = -0,83). O mapa de dispersão da correlação entre o total de erros e o desempenho acadêmico está ilustrado no Gráfico 33 para a primeira coleta e no Gráfico 34 para a segunda coleta.

GRÁFICO 33 – CORRELAÇÃO ENTRE O TOTAL DE ERROS DE COMPILAÇÕES E DESEMPENHO ACADÊMICO DO EXERCÍCIO – EXPERIMENTO 4 – PRIMEIRA COLETA




FONTE: O autor (2018).

GRÁFICO 34 – CORRELAÇÃO ENTRE O TOTAL DE ERROS DE COMPILAÇÕES E DESEMPENHO ACADÊMICO DO EXERCÍCIO – EXPERIMENTO 4 – SEGUNDA COLETA




FONTE: O autor (2018).

### 4.7.5 Investigação de correlação – experimento 5

Os dados coletados no experimento 5 e a nota atribuída aos exercícios foram utilizados para calcular o coeficiente de correlação amostral. Os resultados para a primeira coleta do experimento 5 estão listados no Quadro 17. Devido ao baixo número de alunos participantes do experimento 3 (8 alunos) entende-se que

não é necessário ilustrar o mapa de dispersão para este experimento. Ainda, não foi possível calcular o coeficiente de correlação para a segunda coleta, pois todos os alunos obtiveram a nota 100 como desempenho acadêmico do exercício.

QUADRO 17 – COEFICIENTE DE CORRELAÇÃO – EXPERIMENTO 5

| Variável independente | Coeficiente de Correlação – Primeira Coleta |
|---|---|
| Quantidade de compilações com sucesso | -0,14 |
| Quantidade de compilações com falha | -0,88 |
| Média de erros por compilação | -0,22 |
| Número total de erros de compilação | -0,72 |

FONTE: O autor (2018).

Conforme é possível observar no Quadro 17, existe uma correlação linear negativa muito fraca entre a quantidade de compilações com sucesso e o desempenho acadêmico atribuído ao exercício para a primeira coleta ($r$ = -0,14).

Também é possível observar que existe uma correlação linear negativa muito forte entre a quantidade de compilações com falha e o desempenho acadêmico atribuído ao exercício para a primeira coleta ($r$ = -0,88).

Ainda, é possível observar que existe uma correlação linear negativa fraca entre a quantidade média de erros de compilação e o desempenho acadêmico atribuído ao exercício para a primeira coleta ($r$ = -0,22).

Finalmente, é possível observar que existe uma correlação linear negativa forte entre a quantidade total de erros de compilação e o desempenho acadêmico atribuído ao exercício para a primeira coleta ($r$ = -0,72).

## 4.8 PERCEPÇÃO DOS ALUNOS

Com o intuito de validar a percepção dos alunos quanto aos dados coletados e ao retorno do experimento, após a realização da segunda coleta foi submetido um breve questionário a eles.

A primeira questão foi “Você tinha a percepção de seu comportamento de compilação durante o desenvolvimento de programas?”. As respostas possíveis foram sim e não, sendo que 54,3% responderam sim e 45,7% responderam não.

A segunda questão foi “Você tinha a percepção dos erros gerados no processo de compilação durante o desenvolvimento de programas?”. As respostas possíveis foram sim e não, sendo que 55,2% responderam sim e 44,8% responderam não.

Para as últimas questões foi utilizada a escala de Likert com 5 pontos, variando de -2 até 2, sendo que para calcular a nota final foi somado o número de opiniões de cada resposta e multiplicado pelo fator da resposta. Ao final o resultado das 5 opiniões foram somados.

A terceira questão foi “Classifique a relevância das informações geradas sobre compilações e erros de compilação”, sendo que foi permitido escolher a resposta em uma escala de 1 a 5, sendo 1 – irrelevante (-2); 2 – pouco relevante (-1); 3 – indiferente (0); 4 – relevante (1); e 5 – muito relevante (2). Para esta questão o resultado obtido foi de 0,9, ficando próximo da escala de relevante.

A quarta e última questão foi “Classifique qual foi o impacto das informações geradas sobre compilações e erros de compilação no desenvolvimento da solução do problema de hoje”, sendo que foi permitido escolher a resposta em uma escala de 1 a 5, sendo 1 – nenhum impacto (-2); 2 – pouco impacto (-1); 3 – indiferente (0); 4 – algum impacto (1); e 5 – muito impacto (2). Para esta questão o resultado foi de 0,65, ficando entre as escalas de indiferente e algum impacto.

## 4.9 REFLEXÃO SOBRE OS RESULTADOS

A partir da análise dos dados coletados nos experimentos algumas reflexões podem ser realizadas sobre resultado dos experimentos quanto ao impacto do retorno aos alunos e quanto à correlação dos dados com o desempenho acadêmico atribuído ao exercício.

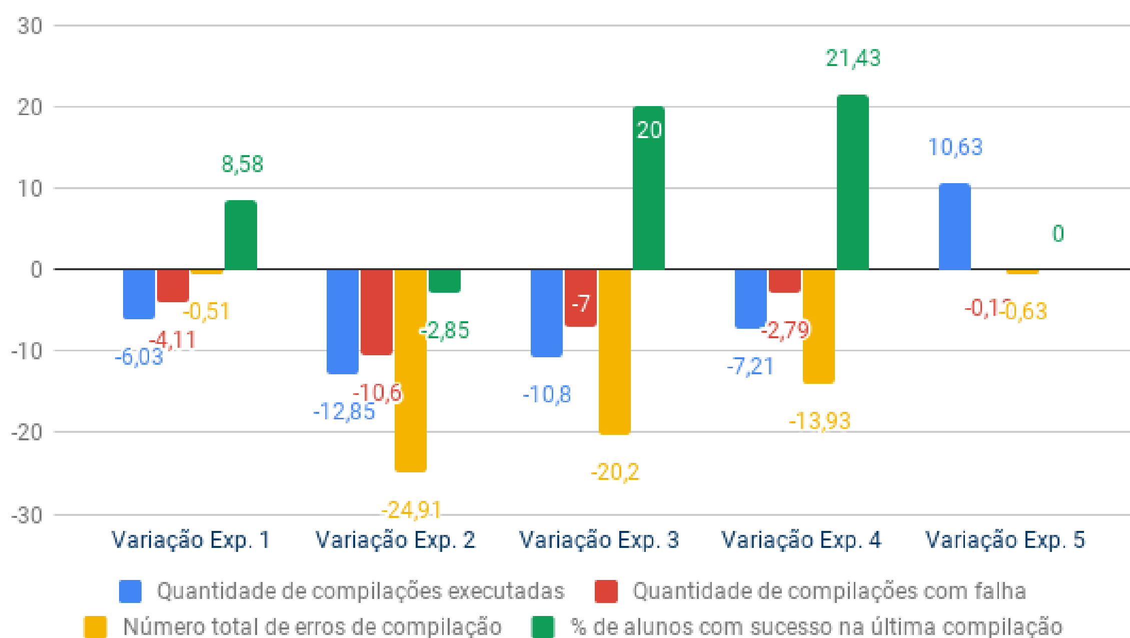
Considerando os dados coletados em cada experimento foi possível notar uma variação entre a primeira e a segunda coleta quanto ao comportamento de compilação dos alunos, conforme comparativo demonstrado no Quadro 18 e ilustrado no Gráfico 35.

QUADRO 18 – VARIAÇÃO ENTRE COLETAS – COMPILAÇÕES E ERROS

| Descrição | Variação Exp. 1 | Variação Exp. 2 | Variação Exp. 3 | Variação Exp. 4 | Variação Exp. 5 |
|---|---|---|---|---|---|
| Quantidade de compilações executadas | -6,03 | -12,85 | -10,80 | -7,21 | +10,63 |
| Quantidade de compilações com sucesso | -2,80 | -1,23 | +1,80 | -0,85 | +7,12 |
| Quantidade de compilações com falha | -4,11 | -10,60 | -7,00 | -2,79 | -0,13 |
| Média de erros por compilação | +0,29 | +0,26 | -1,57 | -0,27 | +0,09 |
| Número total de erros de compilação | -0,51 | -24,91 | -20,2 | -13,93 | -0,63 |
| % de alunos com sucesso na última compilação | +8,58% | -2,85% | +20% | +21,43% | 0 |

FONTE: O autor (2018).

GRÁFICO 35 – VARIAÇÃO ENTRE COLETAS




FONTE: O autor (2018).

Conforme é possível observar no Quadro 18, para a segunda coleta dos experimentos 1, 2, 3 e 4 houve um menor número de compilações realizadas. Já para o experimento 5 houve um maior número de compilações. Quanto ao resultado das compilações, para as compilações com sucesso houve um decréscimo entre a primeira e a segunda coleta para os experimentos 1, 2 e 3. Já para os experimentos 3 e 5 houve um aumento do número das compilações com sucesso.

Quanto as compilações com falha, em todos os experimentos houve um menor número na segunda coleta em comparação com a primeira. Na média de erros por compilação houve um menor número para os experimentos 3 e 4 e um maior número para os experimentos 1, 2 e 5. Já o para o número total de erros de compilação houve um menor número de erros para todos os experimentos.

Finalmente, quanto ao número de alunos que tiveram a última compilação com sucesso, houve um aumento do percentual destes para os experimentos 1, 2, 3 e 4. Para o experimento 5 o percentual foi mantido em 100%.

Sendo assim, pode-se observar uma diferença de comportamento entre a primeira e a segunda coleta dos experimentos, podendo esta diferença ser causada pela ciência do aluno a respeito do seu comportamento na primeira coleta.

Quanto ao tipo de erro de compilação cometido pelos alunos, pode-se observar que o tipo de erro mais cometido foi o de ausência ou sobra de finalizador (;), sendo o tipo de erro mais cometido na primeira coleta do experimento 1 (80% dos alunos), na primeira e na segunda coleta do experimento 2 (85,7% e 80% dos alunos), na primeira e na segunda coleta do experimento 4 (57,1% e 42,9% dos alunos) e na primeira e na segunda coleta do experimento 5 (37,5% e 50% dos alunos). Outro tipo de erro de compilação que também foi cometido por um grande percentual de alunos foi o de variável utilizada sem estar declarada, sendo o segundo erro mais cometido na primeira coleta do experimento 1 (77,1% dos alunos), também o segundo erro mais cometido na primeira coleta do experimento 2 (71,4% dos alunos), o segundo erro mais cometido na primeira coleta e o primeiro erro mais cometido na segunda coleta do experimento 3 (80% e 100% dos alunos), o primeiro erro mais cometido na segunda coleta do experimento 4 (42,9% dos alunos), o primeiro erro mais cometido na primeira coleta e o segundo erro mais cometido na segunda coleta do experimento 5 (37,5% dos alunos em ambas).

Desta forma é possível observar um padrão de recorrência dos tipos de erros mais cometidos no coletivo, não obstante, observa-se também uma diferença entre o tipo de erro de compilação com maior frequência de acordo com problema a ser resolvido.

A respeito da correlação entre os dados coletados em plano de fundo durante o desenvolvimento dos programas e ao desempenho acadêmico atribuído ao exercício, levando em conta apenas os coeficientes de correlação considerados como moderado, forte e muito forte, observa-se que em apenas uma coleta houve uma correlação moderada entre o número de compilações com sucesso e o rendimento acadêmico atribuído ao exercício, sendo a primeira coleta do experimento 3 ($r$ = 0,50), sendo uma correlação linear positiva, ou seja, quanto maior o número de compilações com sucesso maior a nota do exercício. Como esta correlação apareceu em apenas uma coleta, ela está descartada.

Já quanto a correlação entre o número de compilações com falha e o desempenho acadêmico atribuído ao exercício, houve uma correlação linear negativa moderada para a segunda coleta do experimento 1 ($r$ = -0,42), do experimento 2 ($r$ = -0,41) e do experimento 4 ($r$ = -0,49), e uma correlação linear negativa muito forte para a primeira coleta do experimento 5 ($r$ = -0,88). Sendo

assim, é possível observar um indício de que quanto maior o número de compilações com falha menor o desempenho acadêmico no exercício.

Quanto a correlação entre a média de erros de compilações e o desempenho acadêmico atribuído ao exercício, houve uma correlação linear negativa moderada para a primeira coleta do experimento 3 ($r$ = -0,49) e uma correlação linear positiva moderada para a segunda coleta do mesmo experimento ($r$ = 0,43). Também se observa uma correlação linear negativa forte para a segunda coleta do experimento 4 ($r$ = -0,64). Visto que para o experimento 3 os resultados foram conflitantes entre as coletas e em apenas uma coleta de um experimento houve uma correlação considerada forte, esta correlação está descartada.

Finalmente, quanto a correlação entre o número total de erros de compilações e o desempenho acadêmico atribuído ao exercício, houve uma correlação linear negativa moderada para a primeira coleta do experimento 3 ($r$ = -0,43) e uma correlação linear positiva moderada para a segunda coleta do mesmo experimento ($r$ = 0,47), repetindo a tendência da correlação relatada no parágrafo anterior para este experimento. Já para a segunda coleta do experimento 4 observa-se uma correlação linear negativa muito forte ($r$ = -0,83) e forte para a primeira coleta do experimento 5 ($r$ = -0,72). Sendo assim, nota-se uma tendência de que quanto maior o número total de erros de compilação, menor o desempenho acadêmico obtido no exercício.

Em suma, a partir dos resultados obtidos, consolidados neste capítulo e individualmente fornecidos aos docentes das turmas que participaram do experimento, é possível notar uma alteração no comportamento da turma entre as coletas, podendo esta alteração ser oriunda das informações fornecidas no relatório individual de desempenho, visto que entre uma e outra coleta não foi trabalhado especificamente com os alunos a respeito do que foi observado. Ainda, observa-se um padrão entre os erros cometidos pelos alunos, ficando eles concentrados na ausência ou sobra de finalizador (;) e na utilização de variável sem estar declarada. Finalmente, quanto a correlação dos dados com o desempenho acadêmico atribuído ao exercício, nota-se uma tendência de que quanto maior o número de erros de compilação, observados pela quantidade de compilações com falha, pela média de erros de compilação e pelo número total de erros de compilação, menor é o desempenho acadêmico atribuído ao exercício.

# 5 CONCLUSÃO E TRABALHOS FUTUROS

## 5.1 CONCLUSÃO

Este trabalho apresenta um Método e Ferramental para Mapeamento da Evolução de Programadores Durante o Desenvolvimento de Programas. Tal método e ferramental foram testados, através de experimentos, aplicados com alunos de cursos pertinentes à área de Ciência da Computação e Engenharias, matriculados em disciplinas introdutórias de programação. Os experimentos foram aplicados em turmas de 3 instituições diferentes, em cursos de nível técnico e superior.

O método foi proposto visando preencher a lacuna encontrada durante revisão bibliográfica da literatura pertinente, que é a ausência de trabalhos que relatem a coleta de dados oriundos de eventos gerados durante a programação para computadores, classificando também os erros de lógica no artefato final e correlacionado os dados do processo de compilação com uma nota de desempenho acadêmico atribuída ao exercício. Ressalta-se também, como lacuna encontrada, a ausência de trabalhos que relatem a realização de mais de uma coleta para um mesmo grupo de indivíduos, objetivando acompanhar e comparar a evolução entre estas, o que este trabalho fez em 5 experimentos.

Por sua vez, o ferramental proposto possibilita a coleta e análise de dados oriundos de eventos gerados durante a programação para computadores, suportando o método proposto. Tais eventos gerados possibilitaram identificar padrões de comportamento de programação. A separação do ferramental em duas ferramentas visa a posterior utilização destas com outras IDEs ou em outros tipos de análises. Desta forma, o *plugin* de coleta pode ser desenvolvido para outros ambientes e linguagens de programação e a ferramenta de análise ser desenvolvida considerando outros aspectos de análise.

O método e ferramental proposto podem ser utilizados em diversas situações relacionadas ao desenvolvimento de software, seja no processo de ensino-aprendizagem, visando auxiliar no entendimento de como o indivíduo (ou grupo de indivíduos) escreve seus programas para computadores, mapeando possíveis dificuldades durante o processo de desenvolvimento ou, ainda, em ambientes corporativos, para treinamento ou visando o entendimento de como o colaborador desempenha suas atividades, possibilitando mapear padrões e

características deste durante o desenvolvimento de programas para computador. Ressalta-se que, para aplicação dos experimentos relatados neste trabalho, foi optado por realizar as coletas em ambientes de ensino-aprendizagem, devido à atividade profissional do autor e ao acesso dos ambientes de ensino por este.

Embora os dados coletados não sejam estatisticamente significantes, vale destacar os resultados obtidos através das análises realizadas. Sendo que, a partir do cálculo do coeficiente de correlação entre os dados de compilação e a nota atribuída ao exercício foi possível observar um indício de que: a) quanto maior o número de compilações com falha, menor o desempenho acadêmico no exercício; e b) quanto maior o número total de erros de compilação, menor o desempenho acadêmico obtido no exercício.

Ainda, referente aos dados coletados nos experimentos, é possível realizar um agrupamento dos erros cometidos durante o desenvolvimento dos programas pelos alunos, fornecendo ao docente um panorama da turma quanto às dificuldades desta. Destaca-se também que, foi observado um impacto positivo quanto ao retorno detalhado aos alunos sobre os dados coletados durante o desenvolvimento de programas pois, na segunda coleta de dados, em comparação com a primeira, é possível observar uma menor incidência de erros de compilação cometidos durante o processo de desenvolvimento do programa.

## 5.2 AMEAÇAS À VALIDADE DESTE TRABALHO

Apesar da utilização de método científico para elaboração do método e ferramental deste trabalho, é possível observar como ameaça à validade deste o fato da classificação de erros de compilação e erros lógicos serem realizadas de forma manual, contando com a experiência do pesquisador. Também é possível apontar como ameaça à validade deste trabalho a atribuição da nota ao artefato final, realizada através de correção não automática.

Para mitigar a ameaça de classificação dos erros, foi realizada uma dupla verificação do erro pelo pesquisador, classificando os erros de forma inicial e validando a classificação posteriormente. Quanto à atribuição de nota e classificação dos erros lógicos, esta foi realizada pelo pesquisador e por outro docente com experiência no ensino de Lógica de Programação, para as divergências encontradas foi buscado um consenso.

Pode-se ainda apontar que a quantidade de dados coletados não seja estatisticamente significante, porém ressalta-se que o propósito do trabalho é a validação do método e do ferramental, não sendo necessário maior quantidade de dados para tal.

## 5.3 TRABALHOS FUTUROS

Como trabalhos futuros, é possível realizar o acompanhamento dos indivíduos por um maior intervalo de tempo, mapeando a evolução dos mesmos e gerando uma base significativa para análise dos dados, utilizando inclusive técnicas de *datamining*. Ainda, a análise de dados não precisa ficar restrita as realizadas neste trabalho, podendo o leque de análises ser ampliado, utilizando inclusive a base coletada nos experimentos relatados por este trabalho.

Não obstante, também é possível ampliar as coletas para outros ambientes de desenvolvimento e linguagens de programação, propiciando uma análise comparativa entre linguagens e, quiçá, paradigmas de programação.

Outras formas de aplicação da coleta dos dados apresentada neste trabalho também podem ser realizadas. Um exemplo é a utilização em seções de *Coding Dojo* e maratonas de programação.

Ainda, é possível ampliar a forma de coleta e análise, permitindo que o próprio aluno/programador submeta os dados coletados para um sistema e este consiga acompanhar a sua evolução no decorrer do tempo.

Finalmente, a partir do mapeamento dos erros e das dificuldades, é possível gerar um retorno mais específico ao programador, evitando que o mesmo incorra de forma repetitiva nos erros mais comuns de programação.

**REFERÊNCIAS**

ABES. Mercado Brasileiro de Software: panorama e tendências, 2016, Brazilian Software Market: scenario and trends, 2016 [versão para o inglês: Anselmo Gentile] - 1ª. ed. - São Paulo: ABES - Associação Brasileira das Empresas de Software, 2016.

ABES. Mercado Brasileiro de Software: panorama e tendências, 2017, Brazilian Software Market: scenario and trends, 2017 [versão para o inglês: Anselmo Gentile] - 1ª. ed. - São Paulo: ABES - Associação Brasileira das Empresas de Software, 2017.

ACM/IEEE-CS Joint Task Force on Computing Curricula. Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science, ACM, New York, NY, USA, 2013.

AHMADZADEH, M., ELLIMAN, D., HIGGINS, C. An analysis of patterns of debugging among novice computer science students. In Proceedings of the 10th annual SIGCSE conference on Innovation and technology in computer science education (ITiCSE '05). ACM, New York, NY, USA, 84-88. 2005.

BANASZEWSKI, R. F. Paradigma Orientado a Notificações: Avanços e Comparações. Dissertação de Mestrado, CPGEI/UTFPR. Curitiba, 2009.

BENNEDSEN, J., CASPERSEN, M. E. Failure rates in introductory programming. SIGCSE Bulletin, 39(2):32-36, 2007.

BINI, E. M. Ensino De Programação Com Ênfase Na Solução De Problemas. Dissertação de Mestrado, Universidade Tecnológica Federal do Paraná – UTFPR. Curso de Pós-Graduação em Ensino de Ciência e Tecnologia. Ponta Grossa, 2010.

BLIKSTEIN, P. Using learning analytics to assess students' behavior in open-ended programming tasks. In Proceedings of the 1st International Conference on Learning Analytics and Knowledge (LAK '11). ACM, New York, NY, USA, 110-116. 2011.

BLIKSTEIN, P., WORSLEY, M., PIECH, C., SAHAMI, M., COOPER, S., KOLLER, D. Programming Pluralism: Using Learning Analytics to Detect Patterns in the Learning of Computer Programming, Journal of the Learning Sciences. 2014.

BORGES, M. A. F. Avaliação de uma Metodologia Alternativa para a Aprendizagem de Programação. VIII Workshop de Educação em Computação – WEI 2000. Curitiba, PR, Brasil. 2000.

BOURQUE, P., FAIRLEY, R. E. SWEBOK: Guide to the Software Engineering Body of Knowledge, version 3.0 Edition, IEEE Computer Society, Los Alamitos, CA, 2014.

BROOKSHEAR, J. G. Computer Science: An Overview (9 ed.). Addison Wesley, 2006.

CONSELHO NACIONAL DE DESENVOLVIMENTO CIENTÍFICO E TECNOLÓGICO. Tabela de áreas do conhecimento. 2012. Disponível em: <http://www.cnpq.br/documents/10157/186158/TabeladeAreasdoConhecimento.pdf>. Acesso em: 22 nov. 2016.

FERREIRA, C. A. Linguagem e Compilador para o Paradigma Orientado a Notificações (PON): Avanços e Comparações. Dissertação de Mestrado, PPGCA/UTFPR. Curitiba, 2015.

GRANDELL, L., PELTOMÄKI, M., BACK, R. B., SALAKOSKI, T. Why complicate things?: introducing programming in high school using Python. In Proceedings of the 8th Australasian Conference on Computing Education - Volume 52 (ACE '06), Denise Tolhurst and Samuel Mann (Eds.), Vol. 52. Australian Computer Society, Inc., Darlinghurst, Australia, Australia, 71-80. 2006.

HOLMBOE, C. A cognitive framework for knowledge in informatics: the case of object-orientation. In Proceedings of the 4th annual SIGCSE/SIGCUE ITiCSE conference on Innovation and technology in computer science education (ITiCSE '99), Bill Manaris (Ed.). ACM, New York, NY, USA, 17-20, 1999.

IEPSEN, E. F., BERCHT, M., REATEGUI, E. Detecção e Tratamento do Estado Afetivo Frustração do Aluno na Disciplina de Algoritmos. Simpósio Brasileiro de Informática na Educação - SBIE, 80-89, 2011.

IHANTOLA, P., VIHAVAINEN, A., AHADI, A., BUTLER, M., BÖRSTLER, J., EDWARDS, S. H., ISOHANNI, E., KORHONEN, A., PETERSEN, A., RIVERS, K., RUBIO, M. A., SHEARD, J., SKUPAS, B., SPACCO, J., SZABO, C., TOLL, D. Educational Data Mining and Learning Analytics in Programming: Literature Review and Case Studies. In Proceedings of the 2015 ITiCSE on Working Group Reports (ITICSE-WGR '15). ACM, New York, NY, USA, 41-63. 2015.

INSTITUTO FEDERAL DO PARANÁ. Campus União da Vitória. Projeto Pedagógico do Curso Técnico em Informática Integrado ao Ensino Médio. Paraná, 2014.

JADUD, M. C. Methods and tools for exploring novice compilation behaviour. In Proceedings of the second international workshop on Computing education research (ICER '06). ACM, New York, NY, USA, 73-84. 2006.

JADUD, M. C., DORN, B. Aggregate Compilation Behavior: Findings and Implications from 27,698 Users. In Proceedings of the eleventh annual International Conference on International Computing Education Research (ICER '15). ACM, New York, NY, USA, 131-139. 2015.

KUNKLE, W. M. ALLEN, R. B. 2016. The impact of different teaching approaches and languages on student learning of introductory programming concepts. ACM Trans. Comput. Educ. 16, 1, Article 3 (January 2016).

LARSON, R., FARBER, B. Estatística aplicada. 2. ed. São Paulo: Prentice Hall, 2004.

Lawrence Livermore National Laboratory. The Basis Development Team. The Basis System. 2007. Disponível em: <https://wci.llnl.gov/codes/basis/manual/index.html>. Acesso em: 28 jan. 2018.

MASON, R. Designing introductory programming courses: the role of cognitive load. PhD thesis, Southern Cross University, Lismore, NSW, 2012.

MCCALL, D. Novice Programmer Errors - Analysis and Diagnostics. Doctor of Philosophy (PhD) thesis, University of Kent. 2016.

MCCALL, D., KÖLLING, M. 2014. Meaningful Categorisation of Novice Programmer Errors. Frontiers In Education Conference, 2589-2596. 2014.

MCKEOGH, J., EXTON, C. Eclipse plug-in to monitor the programmer behaviour. In Proceedings of the 2004 OOPSLA workshop on eclipse technology eXchange (eclipse '04). ACM, New York, NY, USA, 93-97. 2004.

MEIRELLES, F. S. Informática: novas aplicações com microcomputadores. 2. ed., atual. e ampl. São Paulo: Pearson Education do Brasil, 1994.

MINISTÉRIO DA EDUCAÇÃO. Secretaria de Educação Profissional e Tecnológica. Catálogo Nacional de Cursos Técnicos. Brasília, 2012.

MUNSON, J. P., SCHILLING, E. A. Analyzing novice programmers' response to compiler error messages. J. Comput. Sci. Coll. 31, 3 (January 2016), 53-61. 2016.

MURPHY, C., KAISER, G., LOVELAND, K., HASAN, S. Retina: helping students and instructors based on observed programming activities. SIGCSE Bull. 41, 1 (March 2009), 178-182. 2009.

NOBRE, I. A. M. N., MENEZES, C. S. Suporte à Cooperação em um Ambiente de Aprendizagem para Programação (SAmbA). XIII Simpósio Brasileiro de Informática na Educação – SBIE 2002. São Leopoldo, RS, Brasil. 2002.

NORRIS, C., BARRY, F., FENWICK Jr. J. B., REID, K., ROUNTREE, J. ClockIt: collecting quantitative data on how beginning software developers really work. SIGCSE Bull. 40, 3 (June 2008), 37-41. 2008.

NORTON, P. Introdução à informática. São Paulo: MaKron Books do Brasil, 1997.

PORDEUS. L. F. Simulação de uma arquitetura de computação própria ao paradigma orientado a notificações. Dissertação de Mestrado, CPGEI/UTFPR. Curitiba, 2017.

PRESSMAN, R. S., MAXIM, B. R. Engenharia de software: uma abordagem profissional. 8. ed. Porto Alegre: AMGH, 2016.

ROBINS, A., ROUNTREE, J., ROUNTREE, N. Learning and teaching programming: A review and discussion. Computer Science Education, 13:2, 137-172, 2003.

RODRIGO, M. M. T., BAKER, R. S. J. D. Coarse-grained detection of student frustration in an introductory programming course. In Proceedings of the fifth international workshop on Computing education research workshop (ICER '09). ACM, New York, NY, USA, 75-80. 2009.

RONSZCKA, A. F. Contribuições para a concepção de aplicações no Paradigma Orientado a Notificações (PON) sob o viés de padrões. Dissertação de Mestrado, CPGEI/UTFPR. Curitiba, 2012.

RONSZCKA, A. F., FERREIRA, C. A., STADZISZ, P. C., FABRO, J. A., SIMÃO, J. M. Notification Oriented Programming Language and Compiler, 2017. VII SBESC - Brazilian Symposium on Computing Systems Engineering - November 07 - 10, 2017 - Curitiba - Paraná – Brazil.

SANTOS, L. A. Linguagem e Compilador para o Paradigma Orientado A Notificações: Avanços para Facilitar a Codificação e sua Validação em uma Aplicação de Controle de Futebol de Robôs. Dissertação de Mestrado, CPGEI/UTFPR. Curitiba, 2017.

SANTOS, L. A., SIMÃO, J. M., FABRO, J. A. Linguagem e Compilador para o Paradigma Orientado a Notificações Avanços para a Redução de Complexidade de Código, 2017. VII SBESC - Brazilian Symposium on Computing Systems Engineering - November 07 - 10, 2017 - Curitiba - Paraná – Brazil.

SILVA, M. T., COSTA, E. B., BARBOSA, P. H., CAVALCANTE, J. C. Um Arcabouço para Construção de Mecanismos de Análise de Códigos de Programação Introdutória. 1. Simpósio Brasileiro de Informática na Educação - SBIE, 1083-1092. 2014.

SIMÃO, J. M., STADZISZ, P. C. Paradigma Orientado a Notificações (PON) – Uma Técnica de Composição e Execução de Software Orientado a Notificações. Pedido de Patente submetida ao INPI/Brazil (Instituto Nacional de Propriedade Industrial) em 2008 e a Agência de Inovação/UTFPR em 2007. No. INPI Provisório 015080004262. Patente submetida ao INPI. Brasil, 2008.

SOCIEDADE BRASILEIRA DE COMPUTAÇÃO. Currículo de referência da SBC para cursos de graduação em bacharelado em ciência da computação e engenharia de computação. 2005.

THOMAS, R. C., KARAHASANOVIC, A., KENNEDY, G. E. An investigation into keystroke latency metrics as an indicator of programming performance. In Proceedings of the 7th Australasian conference on Computing education - Volume 42 (ACE '05), Alison Young and Denise Tolhurst (Eds.), Vol. 42. Australian Computer Society, Inc., Darlinghurst, Australia, Australia, 127-134. 2005.

VUJOŠEVIĆ-JANIČIĆ, M., TOŠIĆ, D. The Role of Programming Paradigms In The First Programming Courses, The Teaching of Mathematics, 2008, Vol. XI, 2, pp. 63–83.

XAVIER, R. D. Paradigmas de desenvolvimento de software: comparação entre abordagens orientada a eventos e orientada a notificações. Dissertação de Mestrado, PPGCA/UTFPR. Curitiba, 2014.

WATSON, C., LI, F. W. B. Failure rates in introductory programming revisited, in Proceedings of the 2014 conference on Innovation technology in computer science education (ITiCSE '14). New York: Association for Computing Machinery (ACM), pp. 39-44. 2014.

YOON, Y., MYERS, B. A. Capturing and analyzing low-level events from the code editor. In Proceedings of the 3rd ACM SIGPLAN workshop on Evaluation and usability of programming languages and tools (PLATEAU '11). ACM, New York, NY, USA, 25-30. 2011.

# APÊNDICE A – EXPERIMENTO COMPLEMENTAR

Experimento complementar realizado após a entrega do documento para análise da banca. Optou-se por incluir os resultados em forma de apêndice para não alterar o texto analisado pela banca.

## EXPERIMENTO COMPLEMENTAR

O experimento complementar foi realizado para uma turma de um curso técnico em informática de nível médio, com 39 alunos, e foram aplicados para alunos matriculados na disciplina de Lógica e Linguagem de Programação, disciplina pertencente à 1ª série do referido curso. O docente desta disciplina é o autor deste trabalho de mestrado.

Os alunos destas turmas utilizaram, durante as aulas, a linguagem C para aprender programação para computadores e utilizaram a IDE Code::Blocks. Para grande parte dos alunos desta turma o conhecimento adquirido nesta disciplina foi o primeiro contato destes com programação para computadores. O experimento foi aplicado no primeiro semestre do ano de 2018, após os alunos terem aprendido sobre estruturas de decisão e estruturas de repetição.

Para este experimento foi utilizado o laboratório da instituição de ensino em que os alunos estão matriculados e apenas os computadores do laboratório, sem acesso à Internet. O *plugin* pertinente ao ferramental do método proposto estava previamente instalado e ativado na IDE Code::Blocks. Ao término do exercício o professor coletou o arquivo XML gerado pelo *plugin* e o arquivo com o programa criado pelo aluno.

Conforme proposto no método, foram realizadas duas coletas. Para a primeira coleta foi aplicado um exercício com estrutura de decisão e para a segunda coleta um exercício com estrutura de decisão e estrutura de repetição. O enunciado dos exercícios está descrito a seguir.

<u>Primeira coleta</u>: “Escreva um programa em C que solicite a renda de 3 pessoas de uma mesma família e retorne o valor da renda per capita (soma das rendas / pelo número de pessoas), indicando a faixa de rendimentos conforme a seguir: Faixa 1 – Renda per capita até 900.00; Faixa 2 – Renda per capita entre 900.01 e 1500.00; Faixa 3 – Renda per capita entre 1500.01 e 2500.00; e Faixa 4 – Renda per capita acima de 2500.01".

Segunda coleta: “Escreva um programa em C que calcule a média de peso dos bois de um rebanho. O número de bois deverá ser solicitado ao usuário e, de acordo com o número de bois, o programa deve solicitar o peso de cada boi. Ao final o programa deve informar a média de peso do rebanho e se esta média está boa (peso igual ou superior a 600 kg) ou ruim (abaixo de 600 kg).”.

Após a primeira coleta foi realizada a devolutiva aos alunos a respeito do desempenho nesta. Esta devolutiva foi realizada antes da segunda coleta de dados. A seguir serão relatados os resultados destas coletas.

## RESULTADOS DA PRIMEIRA COLETA – EXPERIMENTO COMPLEMENTAR

Os dados consolidados da primeira coleta para o experimento complementar, a respeito das compilações e execuções, estão apresentados na Tabela 1. Com base nos dados apresentados é possível observar que a média de compilações realizadas para esta coleta foi de 26,59, sendo que a média de compilações com falha foi de 14,67 e a média de erros por compilação de 2,15. Também é possível verificar que 71,8% dos alunos do experimento repetiram erros entre as compilações, sendo que para estes, a média de erros repetidos foi de 41,25. Ainda, 30 dos 39 alunos (76,92%) tiveram a última compilação com sucesso, ou seja, o artefato entregue não apresentou erros de compilação.

TABELA 1 – RESUMO DE COMPILAÇÕES – EXPERIMENTO COMPLEMENTAR – PRIMEIRA COLETA

| Descrição | Alunos com ocorrência | % Alunos com ocorrência | Média da turma | Média – alunos com ocorrência |
|---|---|---|---|---|
| Execuções realizadas | 36 | 92,3 | 12,51 | 13,56 |
| Compilações realizadas | 39 | 100,0 | 26,59 | 26,59 |
| Compilações com sucesso | 35 | 89,7 | 8,82 | 9,83 |
| Compilações com falha | 33 | 84,6 | 14,67 | 17,33 |
| Compilações sem efeito | 22 | 56,4 | 3,10 | 5,50 |
| Média de erros | 33 | - | 2,15 | 2,54 |
| Quantidade total de erros | 33 | - | 51,44 | 60,79 |
| Erros repetidos entre compilações | 28 | 71,8 | 29,62 | 41,25 |
| Última compilação com sucesso | 30 | 76,92 | - | - |

FONTE: O autor (2018).

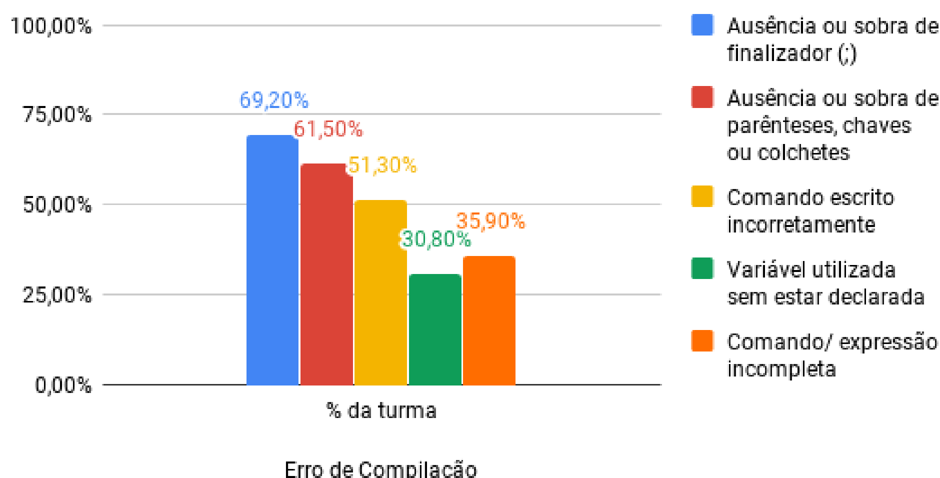
Os dados consolidados a respeito da classificação dos erros de compilação estão apresentados na Tabela 2 e ilustrados no Gráfico 1.

TABELA 2 – CLASSIFICAÇÃO DOS ERROS DE COMPILAÇÃO – EXPERIMENTO COMPLEMENTAR – PRIMEIRA COLETA

| Classificação | Alunos com ocorrência | % Alunos com ocorrência | Média da turma | Média – alunos com ocorrência |
|---|---|---|---|---|
| Ausência ou sobra de finalizador (;) | 27 | 69,2 | 6,03 | 8,70 |
| Ausência ou sobra de parênteses, chaves ou colchetes | 24 | 61,5 | 13,90 | 22,58 |
| Comando escrito incorretamente (sintaxe incorreta ou parâmetros incorretos) | 20 | 51,3 | 14,72 | 28,70 |
| Variável utilizada sem estar declarada | 12 | 30,8 | 2,95 | 9,58 |
| Variável declarada incorretamente ou valor inicial atribuído incorretamente | 7 | 17,9 | 9,41 | 52,43 |
| Variável declarada, porém, utilizada com o nome escrito incorretamente | 1 | 2,6 | 0,05 | 2,00 |
| Tipo de dados incorreto ou inexistente | 2 | 5,1 | 0,18 | 3,50 |
| Comando/expressão incompleta (else sem if, while sem do, falta de parâmetros) | 14 | 35,9 | 3,13 | 8,71 |
| Conectivo de comparação incorreto | 6 | 15,4 | 0,82 | 5,33 |
| Atribuição de valores incorreta | 3 | 7,7 | 0,26 | 3,33 |

FONTE: O autor (2018).

GRÁFICO 1 – ERROS DE COMPILAÇÃO – EXPERIMENTO COMPLEMENTAR – PRIMEIRA COLETA




FONTE: O autor (2018).

De acordo com os dados apresentados é possível observar que 69,2% dos alunos cometeram, ao menos uma vez, o erro de ausência ou sobra de finalizador (;), sendo que a média deste erro para estes alunos foi de 8,70. Outro erro cometido por um grande percentual destes alunos (61,5%) foi o de ausência ou sobra de

parênteses, chaves ou colchetes, sendo que a média deste erro para estes alunos foi de 22,58.

Os dados consolidados a respeito dos erros de lógica apresentados no artefato entregue estão descritos na Tabela 3. Nesta tabela é possível observar que 61,5% dos alunos entregaram o artefato com erros de lógica, sendo que 30,8% destes apresentaram erro com operador relacional e erro com estrutura de decisão.

TABELA 3 – RESUMO DE ERROS DE LÓGICA – EXPERIMENTO COMPLEMENTAR – PRIMEIRA COLETA

| Descrição | Alunos com ocorrência | % de alunos da turma |
|---|---|---|
| Erro de Lógica | 24 | 61,5 |
| Erro com operador lógico | 7 | 17,9 |
| Erro com operador relacional | 12 | 30,8 |
| Erro de ordem/sequência de comandos | 0 | 0,0 |
| Erro aritmético | 6 | 15,4 |
| Erro com estruturas de repetição | 0 | 0,0 |
| Erro com estruturas de decisão | 12 | 30,8 |

FONTE: O autor (2018).

O resumo das notas atribuídas está descrito na Tabela 4. Para esta coleta a maioria dos alunos (38,5%) ficou com nota 100 e a média das notas da turma foi de 60,51.

TABELA 4 – RESUMO DE NOTAS – EXPERIMENTO COMPLEMENTAR – PRIMEIRA COLETA

| Nota | Quantidade de Alunos | % de alunos da turma |
|---|---|---|
| Nota 100 | 15 | 38,5 |
| Nota 80 | 0 | 0,0 |
| Nota 60 | 6 | 15,4 |
| Nota 40 | 9 | 23,1 |
| Nota 20 | 7 | 17,9 |
| Nota 0 | 2 | 5,1 |

FONTE: O autor (2018).

## RESULTADOS DA SEGUNDA COLETA – EXPERIMENTO COMPLEMENTAR

Os dados consolidados da segunda coleta para o experimento complementar, a respeito das compilações e execuções, estão apresentados na Tabela 5. Com base nos dados apresentados é possível observar que a média de compilações realizadas para esta coleta foi de 17,90, sendo que a média de compilações com falha foi de 7,74 e a média de erros por compilação de 1,35. Também é possível verificar que 51,3% dos alunos do experimento repetiram erros

entre as compilações, sendo que para estes a média de erros repetidos foi de 8,45. Ainda, 33 dos 39 alunos (84,62%) tiveram a última compilação com sucesso, ou seja, o artefato entregue não apresentou erros de compilação.

Em comparação com a primeira coleta para o experimento complementar, observa-se que na segunda coleta houve uma menor média de compilações realizadas (-8,69), menor média de compilações com falha (-6,93) e um menor número de média de erros por compilação (-0,80). Na segunda coleta houve um menor número de alunos que repetiram erros entre as compilações (-20,5% da turma), e para os que repetiram erros, o número de erros repetidos diminuiu (-32,80). Finalmente, o percentual de alunos que entregaram o artefato sem erros de compilação foi maior na segunda coleta (7,70% da turma).

TABELA 5 – RESUMO DE COMPILAÇÕES – EXPERIMENTO COMPLEMENTAR – SEGUNDA COLETA

| Descrição | Alunos com ocorrência | % Alunos com ocorrência | Média da turma | Média – alunos com ocorrência |
|---|---|---|---|---|
| Execuções realizadas | 39 | 100,0 | 10,33 | 10,33 |
| Compilações realizadas | 39 | 100,0 | 17,90 | 17,90 |
| Compilações com sucesso | 38 | 97,4 | 7,67 | 7,87 |
| Compilações com falha | 31 | 79,5 | 7,74 | 9,74 |
| Compilações sem efeito | 22 | 56,4 | 2,49 | 4,41 |
| Média de erros | 31 | - | 1,35 | 1,69 |
| Quantidade total de erros | 31 | - | 13,28 | 16,71 |
| Erros repetidos entre compilações | 20 | 51,3 | 4,33 | 8,45 |
| Última compilação com sucesso | 33 | 84,62 | - | - |

FONTE: O autor (2018).

Os dados consolidados a respeito da classificação dos erros de compilação estão apresentados na Tabela 6 e ilustrados no Gráfico 2. De acordo com os dados apresentados é possível observar que 56,4% dos alunos cometeram, ao menos uma vez, o erro de ausência ou sobra de finalizador (;), sendo que a média deste erro para estes alunos foi de 6,23. Outro erro cometido por um grande percentual destes alunos (48,7%) foi o de ausência ou sobra de parênteses, chaves ou colchetes, sendo que a média deste erro para estes alunos foi de 5,26.

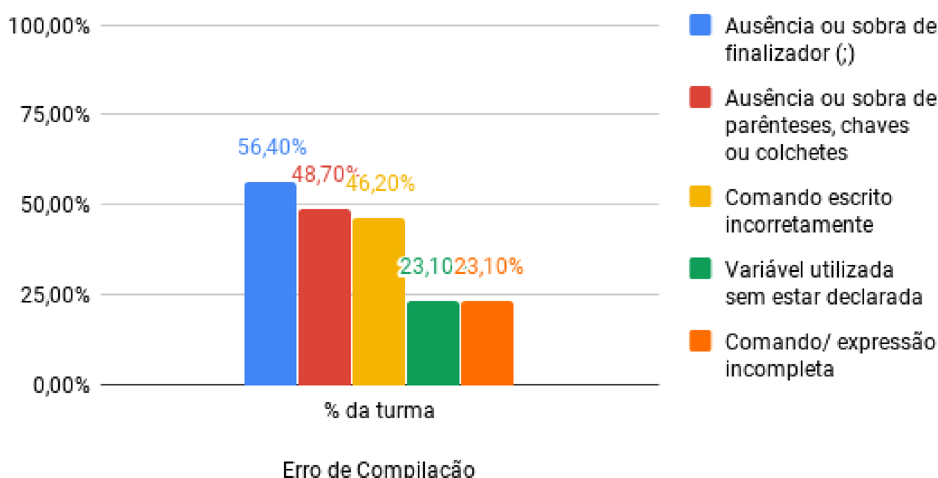
Em comparação com a primeira coleta para o experimento complementar, é possível observar que houve semelhança entre os tipos de erro mais cometidos.

TABELA 6 – CLASSIFICAÇÃO DOS ERROS DE COMPILAÇÃO – EXPERIMENTO COMPLEMENTAR – SEGUNDA COLETA

| Classificação | Alunos com ocorrência | % Alunos com ocorrência | Média da turma | Média – alunos com ocorrência |
|---|---|---|---|---|
| Ausência ou sobra de finalizador (;) | 22 | 56,4 | 3,51 | 6,23 |
| Ausência ou sobra de parênteses, chaves ou colchetes | 19 | 48,7 | 2,56 | 5,26 |
| Comando escrito incorretamente (sintaxe incorreta ou parâmetros incorretos) | 18 | 46,2 | 4,18 | 9,06 |
| Variável utilizada sem estar declarada | 9 | 23,1 | 1,31 | 5,67 |
| Variável declarada incorretamente ou valor inicial atribuído incorretamente | 2 | 5,1 | 0,23 | 4,50 |
| Variável declarada, porém, utilizada com o nome escrito incorretamente | 1 | 2,6 | 0,03 | 1,00 |
| Tipo de dados incorreto ou inexistente | 1 | 2,6 | 0,03 | 1,00 |
| Comando/expressão incompleta (else sem if, while sem do, falta de parâmetros) | 9 | 23,1 | 0,97 | 4,22 |
| Conectivo de comparação incorreto | 5 | 12,8 | 0,33 | 2,60 |
| Atribuição de valores incorreta | 1 | 2,6 | 0,10 | 4,00 |

FONTE: O autor (2018).

GRÁFICO 2 – ERROS DE COMPILAÇÃO – EXPERIMENTO COMPLEMENTAR – SEGUNDA COLETA




FONTE: O autor (2018).

Os dados consolidados a respeito dos erros de lógica apresentados no artefato entregue estão descritos na Tabela 7. Nesta tabela é possível observar que 59% dos alunos entregaram o artefato com erros de lógica, sendo que 38,5% destes apresentaram erro com ordem ou sequência de comandos.

Em comparação com a primeira coleta do experimento complementar é possível observar que houve uma diminuição dos alunos que entregaram os artefatos com erro de lógica (-2,5% da turma).

TABELA 7 – RESUMO DE ERROS DE LÓGICA – EXPERIMENTO COMPLEMENTAR – SEGUNDA COLETA

| Descrição | Alunos com ocorrência | % de alunos da turma |
|---|---|---|
| Erro de Lógica | 23 | 59,0 |
| Erro com operador lógico | 0 | 0,0 |
| Erro com operador relacional | 5 | 12,8 |
| Erro de ordem/sequência de comandos | 15 | 38,5 |
| Erro aritmético | 12 | 30,8 |
| Erro com estruturas de repetição | 6 | 15,4 |
| Erro com estruturas de decisão | 8 | 20,5 |

FONTE: O autor (2018).

O resumo das notas atribuídas está descrito na Tabela 8. Para esta coleta a maioria dos alunos (41%) ficou com nota 100 e a média das notas da turma foi de 57,95. Em comparação com a primeira coleta do experimento complementar foi possível observar uma queda na média da turma (-2,56).

TABELA 8 – RESUMO DE NOTAS – EXPERIMENTO COMPLEMENTAR – SEGUNDA COLETA

| Nota | Quantidade de Alunos | % de alunos da turma |
|---|---|---|
| Nota 100 | 16 | 41,0 |
| Nota 80 | 0 | 0,0 |
| Nota 60 | 7 | 17,9 |
| Nota 40 | 5 | 12,8 |
| Nota 20 | 2 | 5,1 |
| Nota 0 | 9 | 12,1 |

FONTE: O autor (2018).

## CORRELAÇÃO DADOS COLETADOS x DESEMPENHO ACADÊMICO

Conforme previso no método proposto, para o experimento complementar foi investigada a correlação entre alguns dos dados coletados em plano de fundo durante a solução dos exercícios, como variável independente, e o desempenho acadêmico atribuído ao exercício, como variável dependente. Os dados escolhidos para investigar a correlação são: a) número de compilações com falha; b) número de compilações com sucesso; c) média de erros por compilação); e d) número total de erros.

A investigação da correlação foi realizada através do cálculo do coeficiente de correlação e do mapa de dispersão. Os resultados dos testes para o experimento complementar estão relatados s seguir.

Os dados coletados no experimento complementar e a nota atribuída aos exercícios foram utilizados para calcular o coeficiente de correlação amostral. Os resultados para a primeira e para a segunda coleta do experimento complementar estão listados no Quadro 1.

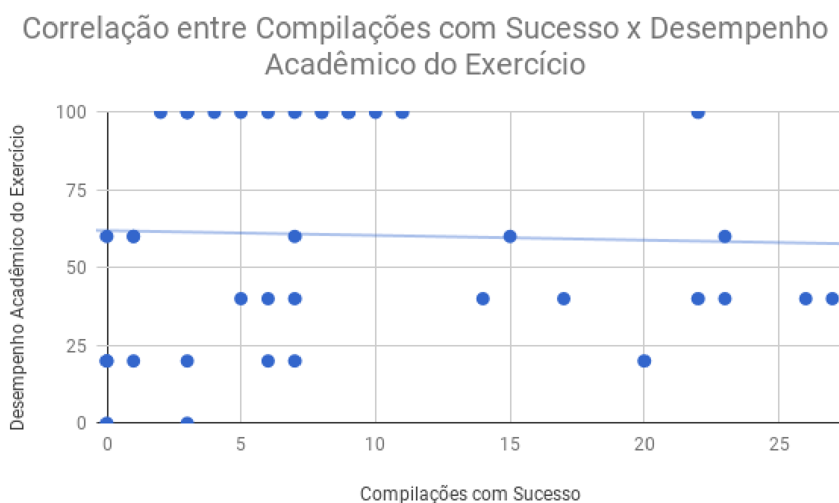
QUADRO 1 – COEFICIENTE DE CORRELAÇÃO – EXPERIMENTO COMPLEMENTAR

| **Variável independente** | **Coeficiente de Correlação – Primeira Coleta** | **Coeficiente de Correlação – Segunda Coleta** |
|---|---|---|
| Quantidade de compilações com sucesso | -0,04 | -0,14 |
| Quantidade de compilações com falha | -0,63 | -0,64 |
| Média de erros por compilação | -0,53 | -0,37 |
| Número total de erros de compilação | -0,54 | -0,61 |

FONTE: O autor (2018).

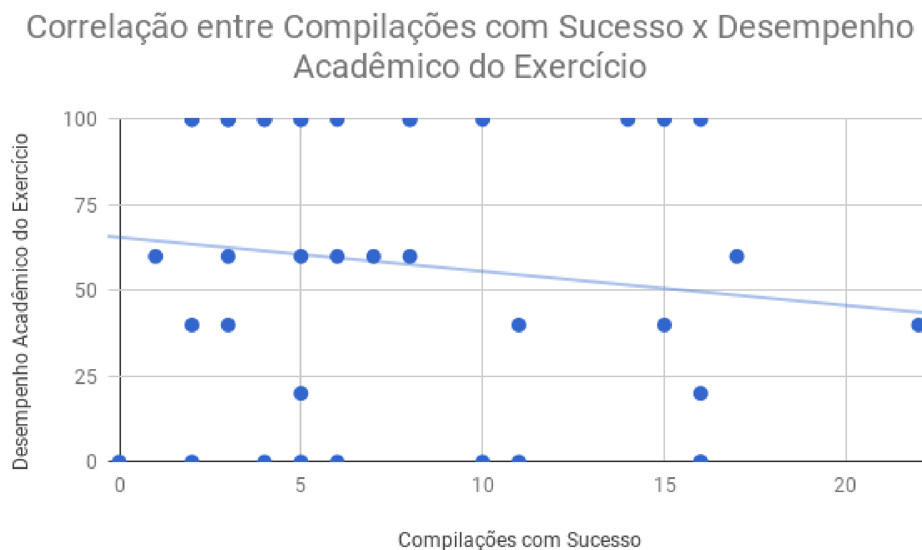
Conforme é possível observar no Quadro 1, existe uma correlação linear negativa entre a quantidade de compilações com sucesso e o desempenho acadêmico atribuído ao exercício. Para a primeira coleta ($r$ = -0,04) e para a segunda coleta ($r$ = -0,14), sendo consideradas como muito fraca. O mapa de dispersão da correlação entre a quantidade de compilações com sucesso e o desempenho acadêmico está ilustrado no Gráfico 3 para a primeira coleta e no Gráfico 4 para a segunda coleta.

GRÁFICO 3 – CORRELAÇÃO ENTRE COMPILAÇÕES COM SUCESSO E DESEMPENHO ACADÊMICO DO EXERCÍCIO – EXPERIMENTO COMPLEMENTAR – PRIMEIRA COLETA




FONTE: O autor (2018).

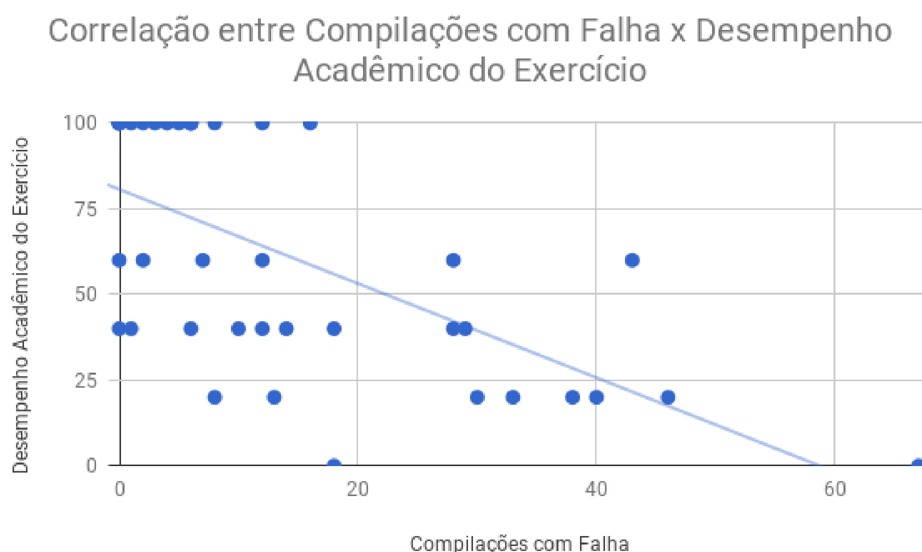
GRÁFICO 4 – CORRELAÇÃO ENTRE COMPILAÇÕES COM SUCESSO E DESEMPENHO ACADÊMICO DO EXERCÍCIO – EXPERIMENTO COMPLEMENTAR – SEGUNDA COLETA




FONTE: O autor (2018).

Também é possível observar que existe uma correlação linear negativa entre a quantidade de compilações com falha e o desempenho acadêmico atribuído ao exercício, sendo forte para a primeira coleta ($r$ = -0,63) e também forte para a segunda coleta ($r$ = -0,64). O mapa de dispersão da correlação entre a quantidade de compilações com falha e o desempenho acadêmico está ilustrado no Gráfico 5 para a primeira coleta e no Gráfico 6 para a segunda coleta.

GRÁFICO 5 – CORRELAÇÃO ENTRE COMPILAÇÕES COM FALHA E DESEMPENHO ACADÊMICO DO EXERCÍCIO – EXPERIMENTO COMPLEMENTAR – PRIMEIRA COLETA




FONTE: O autor (2018).

GRÁFICO 6 – CORRELAÇÃO ENTRE COMPILAÇÕES COM FALHA E DESEMPENHO ACADÊMICO DO EXERCÍCIO – EXPERIMENTO COMPLEMENTAR – SEGUNDA COLETA



FONTE: O autor (2018).

Ainda, é possível observar que existe uma correlação linear negativa entre a quantidade média de erros de compilação e o desempenho acadêmico atribuído ao exercício, sendo moderada para a primeira coleta (*r* = -0,53) e fraca para a segunda coleta (*r* = -0,37). O mapa de dispersão da correlação entre a média de erros e o desempenho acadêmico está ilustrado no Gráfico 7 para a primeira coleta e no Gráfico 8 para a segunda coleta.

GRÁFICO 7 – CORRELAÇÃO ENTRE A MÉDIA DE ERROS DE COMPILAÇÕES E DESEMPENHO ACADÊMICO DO EXERCÍCIO – EXPERIMENTO COMPLEMENTAR – PRIMEIRA COLETA



FONTE: O autor (2018).

GRÁFICO 8 – CORRELAÇÃO ENTRE A MÉDIA DE ERROS DE COMPILAÇÕES E DESEMPENHO ACADÊMICO DO EXERCÍCIO – EXPERIMENTO COMPLEMENTAR – SEGUNDA COLETA

Correlação entre Média de Erros x Desempenho Acadêmico do Exercício

Desempenho Acadêmico do Exercício

Média de Erros

FONTE: O autor (2018).

Finalmente, é possível observar que existe uma correlação linear negativa entre a quantidade total de erros de compilação e o desempenho acadêmico atribuído ao exercício, sendo moderada para a primeira coleta ($r$ = -0,54) e forte para a segunda coleta ($r$ = -0,61). O mapa de dispersão da correlação entre o total de erros e o desempenho acadêmico está ilustrado no Gráfico 9 para a primeira coleta e no Gráfico 10 para a segunda coleta.

GRÁFICO 9 – CORRELAÇÃO ENTRE O TOTAL DE ERROS DE COMPILAÇÕES E DESEMPENHO ACADÊMICO DO EXERCÍCIO – EXPERIMENTO COMPLEMENTAR – PRIMEIRA COLETA

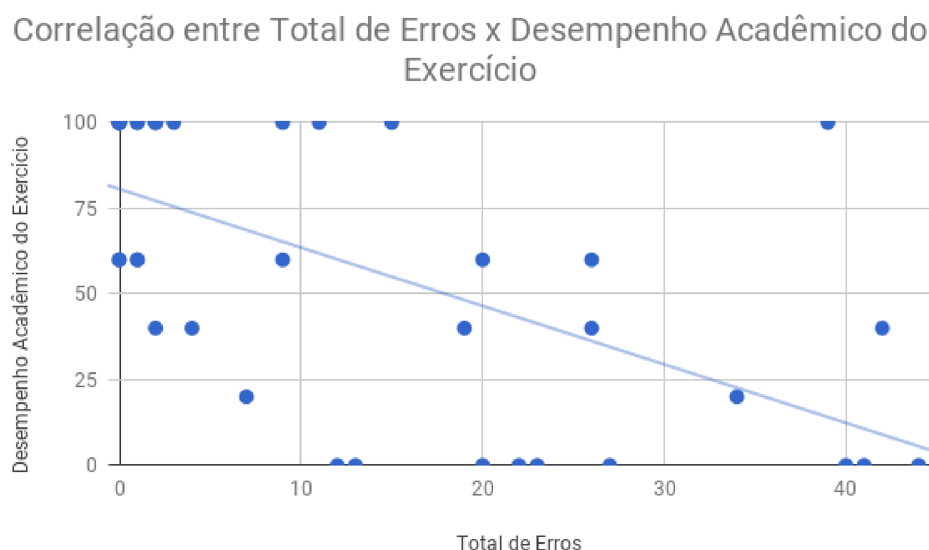
Correlação entre Total de Erros x Desempenho Acadêmico do Exercício

Desempenho Acadêmico do Exercício

Total de Erros

FONTE: O autor (2018).

GRÁFICO 10 – CORRELAÇÃO ENTRE O TOTAL DE ERROS DE COMPILAÇÕES E DESEMPENHO ACADÊMICO DO EXERCÍCIO – EXPERIMENTO COMPLEMENTAR – SEGUNDA COLETA




FONTE: O autor (2018).

REFLEXÃO SOBRE OS RESULTADOS

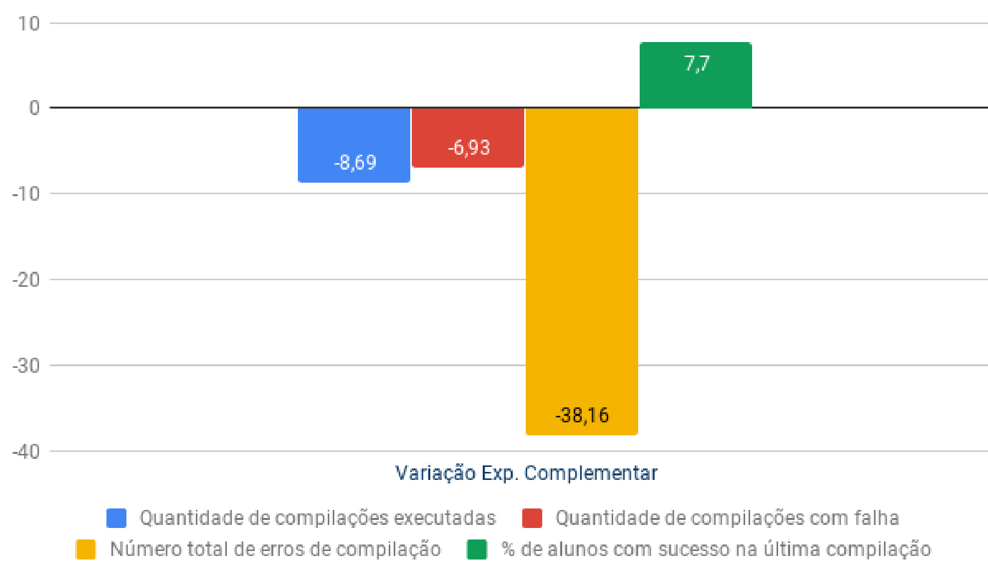
O experimento complementar, realizado após a entrega da dissertação para análise da banca, apresentou-se como oportunidade para corroborar os resultados quanto ao indício de melhora do comportamento da turma entre uma coleta e outra e também quanto aos coeficientes de correlação testados. Também observou-se que o padrão dos erros de compilação repete-se.

Quanto à diferença entre as coletas, é possível observar uma melhora em alguns aspectos, principalmente quanto ao menor número de erros de compilação cometidos pelos alunos. A diferença está ilustrado no Gráfico 11.

Quanto ao coeficiente de correlação, o experimento complementar serviu para corroborar que existe indício de correlação linear negativa entre o desempenho acadêmico e o número de compilações com falha e também entre o desempenho acadêmico e o número total de erros de compilação.

Finalmente, o experimento complementar corrobora o uso do método e do ferramental como auxílio nas atividades profissionais do autor do trabalho, auxiliando no processo de ensino-aprendizagem da programação de computadores.

GRÁFICO 11 – VARIAÇÃO ENTRE COLETAS




FONTE: O autor (2018).

# APÊNDICE B – PROPOSTA DE MÉTODO PARA COMPARATIVO ENTRE PARADIGMAS DE PROGRAMAÇÃO QUANTO À APRENDIZAGEM

Documento apresentado no Seminário de Qualificação II para o programa de Pós-Graduação em Computação Aplicada do DAINF/UTFPR. Defesa realizada em 29/06/2017. Banca composta por: Profa. Dra. Maria Cláudia Figueiredo Pereira Emer, Prof. Dr. João Alberto Fabro, Prof. Dr. Jean Marcelo Simão e Prof. Dr. Laudelino Cordeiro Bastos.

**UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ**

DEPARTAMENTO ACADÊMICO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO APLICADA

**SEMINÁRIO DE ACOMPANHAMENTO II**

**SEMINÁRIO DE ACOMPANHAMENTO II**

# PROPOSTA DE MÉTODO PARA COMPARATIVO ENTRE PARADIGMAS DE PROGRAMAÇÃO QUANTO À APRENDIZAGEM

**Nome: Douglas Lusa Krug**

**Orientador: Prof. Dr. Laudelino Cordeiro Bastos**

**Co-orientador: Prof. Dr. Jean Marcelo Simão**

**CURITIBA**

**2017**

**SUMÁRIO**

1. INTRODUÇÃO....................9
1.1. OBJETIVOS....................13
1.1.1. Objetivo geral....................13
1.1.2. Objetivos específicos....................13
1.2. ESTRUTURA DO TRABALHO....................14
2. REVISÃO BIBLIOGRÁFICA....................15
2.1. PARADIGMA....................15
2.2. PARADIGMAS DE PROGRAMAÇÃO....................17
2.3. PARADIGMA IMPERATIVO....................21
2.4. PARADIGMA DECLARATIVO....................26
2.5. SISTEMAS BASEADOS EM REGRAS....................31
2.5.1. Linguagens e ferramentas para SBR....................35
2.5.2. Aplicações com SBR....................37
2.6. ENSINO DE PROGRAMAÇÃO....................39
2.7. COMPARATIVO ENTRE PARADIGMAS....................41
2.7.1. Características dos paradigmas quanto à aprendizagem....................41
2.7.2. Comparativo quanto ao ensino de programação....................43
2.7.3. Modelo de compreensão de programas....................46
2.7.4. Estudos comparativos....................47
2.7.5. Reflexão sobre Estudos Comparativos....................51
3. MÉTODO PARA COMPARAÇÃO ENTRE PARADIGMAS....................52
3.1. MÉTODO PROPOSTO....................52
3.1.1. Coleta de Dados....................53
3.1.2. Classificação dos Artefatos....................53
3.1.3. Processamento das Atividades Geradas....................54
3.1.4. Análise e Comparação....................55
3.1.5. Próximas Etapas....................55
4. EXPERIMENTO....................56
4.1. ESCOLHA DOS PARADIGMAS....................56
4.2. POPULAÇÃO....................57
4.2.1. Termo de Consentimento Livre e Esclarecido....................57

4.3. ETAPAS DA COLETA DE DADOS....58
4.4. PROBLEMAS PROPOSTOS....61
4.5. QUESTIONÁRIO....62
5. RESULTADOS....63
5.1. CLASSIFICAÇÃO DOS ARTEFATOS....63
5.2. CATEGORIZAÇÃO DE ERROS....64
5.3. QUESTIONÁRIOS APLICADOS....66
5.3.1. Informações Pessoais....66
5.3.2. Avaliação Relacionada ao Paradigma....67
6. DISCUSSÃO....71
7. CONCLUSÃO....74
REFERÊNCIAS....76
SEÇÃO COMPLEMENTAR A – REVISÃO SISTEMÁTICA DA LITERATURA....81
SEÇÃO COMPLEMENTAR B – TORRE DE HANÓI COM PON....99
SEÇÃO COMPLEMENTAR C – TERMO DE CONSENTIMENTO LIVRE E ESCLARECIDO....108
SEÇÃO COMPLEMENTAR D – APOSTILA DE PARADIGMA PROCEDIMENTAL....110
SEÇÃO COMPLEMENTAR E – APOSTILA DE SISTEMAS BASEADOS EM REGRAS....126
SEÇÃO COMPLEMENTAR F – QUESTIONÁRIOS APLICADOS AOS ALUNOS....150

## LISTA DE ILUSTRAÇÕES

Figura 1 – Paradigmas dominantes – Adaptado de BANASZEWSKI, 2009......19
Figura 2 – Taxonomia de paradigmas de programação - Adaptado de VAN ROY, 2009..........................................................................................................20
Figura 3 – Ranking de linguagens de programação (IEEE, 2016)....................21
Figura 4 – Comparação entre C e C++.............................................................24
Figura 5 – Exemplo de redundância temporal e estrutural – Adaptado de SIMÃO et al., 2012..............................................................................26
Figura 6 – Programa em LISP..........................................................................30
Figura 7 – Programa em CLIPS........................................................................30
Figura 8 – Arquitetura de um Sistema Baseado em Regras............................31
Figura 9 – Exemplo de programa em CLIPS – Adaptado de GIARRATANO, 2015..........................................................................................................33
Figura 10 – Etapas - alunos da 1a série...........................................................60
Figura 11 – Etapas - alunos da 2a série...........................................................61
Figura 12 – Gráfico com a classificação dos artefatos.....................................64
Figura 13 – Gráfico com a classificação dos artefatos.....................................65
Figura 14 – Gráfico com a classificação dos artefatos.....................................65

## LISTA DE TABELAS E QUADROS

Quadro 1 – Participantes do experimento........................................................63
Tabela 1 - Percentual das classificações para cada turma e cada paradigma.. 63
Tabela 2 - Percentual das classificações para cada turma e cada paradigma.. 64
Tabela 3 - Percentual dos participantes por idade............................................66
Tabela 4 - Percentual dos participantes por gênero..........................................66
Tabela 5 - Percentual dos participantes por conhecimento anterior em programação..........................................................................................66
Tabela 6 - Percentual dos participantes que dedicou tempo de estudo............67
Tabela 7 – Respostas quanto à motivação.......................................................67
Tabela 8 – Respostas quanto à facilidade da ferramenta.................................68
Tabela 9 – Respostas quanto à clareza das explicações.................................68
Tabela 10 – Respostas quanto à clareza da apostila........................................68
Tabela 11 – Respostas quanto à utilidade dos exemplos.................................68
Tabela 12 – Respostas quanto à facilidade dos exercícios..............................68
Tabela 13 – Respostas quanto à facilidade do exercício final..........................69
Tabela 14 – Respostas quanto ao entendimento do código.............................69
Tabela 15 – Respostas quanto à facilidade de programação...........................69
Tabela 16 – Respostas quanto à experiência geral do experimento................69

## LISTA DE ABREVIATURAS E SIGLAS

| **Sigla** | **Significado** | **Sigla (EN)** | **Significado (EN)** |
|---|---|---|---|
| MI | Máquina de Inferência | IE | *Inference Engine* |
| PD | Paradigma Declarativo | DP | *Declarative Programming* |
| PF | Paradigma Funcional | FP | *Functional Programming* |
| PI | Paradigma Imperativo | IP | *Imperative Programming* |
| PL | Paradigma Lógico | LP | *Logic Programming* |
| PON | Paradigma Orientado a Notificações | NOP | *Notification Oriented Paradigm* |
| PP | Paradigma Procedimental | PP | *Procedural Programming* |
| SBR | Sistemas Baseados em Regras | RBS | *Rule-Based Systems* |
| TI | Tecnologia da Informação | IT | *Information Technology* |

# RESUMO

A programação de computadores é parte importante da área de Engenharia de Software, sendo o ensino da mesma imprescindível em cursos de Ciência da Computação e afins. Tido por muitos estudantes como difícil e apresentando elevado número de reprovações e desistências, o ensino de programação é assunto discutido infindavelmente. Parte desta discussão é a escolha do paradigma e da linguagem de programação que deve ser utilizada para o ensino aos novatos de programação. Até hoje não existe consenso e existem poucos estudos efetivamente comparativos. Primeiramente, este trabalho descreve a revisão bibliográfica a respeito de paradigmas de programação e estudos comparativos destes paradigmas. Na sequência, apresenta um método proposto neste trabalho para comparação entre paradigmas de programação quanto à aprendizagem. O trabalho apresenta uma primeira validação do método através de experimento realizado com discentes, alunos de curso técnico em informática de nível médio, a fim de comparar o Paradigma Procedimental e o Paradigma Declarativo-Lógico – Sistemas Baseados em Regras, sendo este vislumbrado como mais fácil. A hipótese que o trabalho visa corroborar é a de que é possível comparar paradigmas de programação através das atividades (i.e., erros, latência, performance de codificação etc) geradas durante a codificação de programas de computador, visando colaborar com o processo de ensino-aprendizagem de novatos em programação, aumentando assim a qualidade de artefatos gerados. Resultados parciais são apresentados e discutidos como a maior facilidade do Paradigma Procedimental em comparação à Sistemas Baseados em Regras para o aprendizado de novatos em programação. A partir dos resultados é possível verificar a possibilidade de aplicação do método, mesmo que ainda descrito parcialmente, para comparação entre o Paradigma Procedimental e Sistemas Baseados em Regras e, quiçá, para outros paradigmas.

Palavras chaves: Sistemas Baseados em Regras; Paradigma Declarativo-Lógico; Paradigma Procedimental; Comparação de paradigmas; Aprendizado de Programação.

# ABSTRACT

*The programming of computers is an important part of Software Engineering, and the teaching of it is essential in Computer Science courses. Seen by many students as difficult, and with a high number of failure and drop outs, programming teaching is a highly discussed subject. Part of this discussion is the choice of programming paradigm and programming language that should be used for teaching programming for novices. To date there is no consensus, and there are few effective comparative studies. First, this paper describes the revision of the literature regarding programming paradigms and comparative studies of these paradigms. In the sequence it presents a proposed method for comparison between paradigms with focus in learning. The work presents a first validation of the method through an experiment carried out with students of technical course in computer science of high school, in order to compare the Procedural Paradigm and the Declarative Paradigm – Rule Based Systems, which is perceived as easier. The hypothesis that the work aims to corroborate is that it is possible to compare programming paradigms through the activities (i.e., errors, latency, coding performance etc.) generated during the coding of computer programs aiming to collaborate with the teaching-learning process of beginners in programming, increasing the quality of the generated artifact. Partial results are presented and discussed as, the greater facility of the Procedural Programming compared to Rule-Based Systems for beginners in programming. from them it is possible to verify the possibility of application of the method, even if it still partially described, for comparison between the Procedural Paradigm and Rules Based Systems, and, perhaps, other programming paradigms.*



*Keywords: Rules Based Systems; Declarative Programming; Procedural Programming; Comparative paradigms; Programming Learning.*

# 1. INTRODUÇÃO

A sociedade vem mudando desde o advento do computador. Neste contexto, tarefas antes realizadas manualmente com grande esforço e elevado tempo, hoje são realizadas de forma rápida e sem esforços manuais através da automatização de processos realizados via computador na forma de software. O advento do computador gerou novas atividades profissionais, como a programação de computadores, a qual permite elaborar programas que são a essência de um software no tocante a sua execução.

Programação de computadores é a ação de escrever, depurar, manter e testar programas que podem ser executados pelos computadores. Programas de computador são instruções escritas em uma linguagem de programação, as quais são compiladas para algum tipo de código executável para o computador. Em última instância, estas instruções são compiladas ou transformadas em código binário, o que é realmente executado pela máquina ou computador. Mais precisamente, este binário é resultante do processo de compilação do programa em uma linguagem de programação por um dado compilador, o qual contém o gerador de código binário executável em uma arquitetura de computador (BROOKSHEAR, 2006).

Em arquiteturas tradicionais de computação, nomeadamente a proposta por Von Neumann, as instruções são processadas de maneira sequencial, ainda que haja algum paralelismo como pipeline. Mesmo em arquitetura multinúcleo, ao menos em cada núcleo o processamento se dá de maneira sequencial. Em tempo, programas de computador são construídos para solucionar um problema específico, sendo que as linguagens de programação mais utilizadas impõem uma maneira de pensar sequencial, i.e. um paradigma sequencial (SIMÃO e STADZISZ, 2008; BANASZEWSKI, 2009; LINHARES, SIMÃO e STADZISZ, 2015; BELMONTE et al., 2016).

A programação de computadores, sequencial ou não, permite compor soluções na forma de software, cujo binário é o artefato executável na arquitetura computacional. Assim, a programação faz parte da Engenharia de Software, área que abrange inclusive a Tecnologia da Informação (TI). A Engenharia de Software em si pode ser definida como a aplicação de uma sistemática e disciplinada abordagem quantificável para o desenvolvimento, operação e manutenção de software (BOURQUE e FAIRLEY, 2014).

Embora a programação de computadores esteja intimamente ligada com a área de Engenharia de Software, é pertinente ressaltar o viés do documento do Conselho Nacional de Desenvolvimento Científico e Tecnológico – CNPq (2012) que lista a divisão de áreas do conhecimento, subáreas e especialidades. Neste documento, observa-se que Linguagens de Programação e Engenharia de Software são especialidades distintas, vinculadas à subárea de Metodologia e Técnicas da Computação (CNPq, 2012).

Em todo caso, a programação de computadores é parte imprescindível no processo de desenvolvimento de software, o qual tem seu mercado crescendo inclusive na República Federativa do Brasil. Segundo o relatório da Associação Brasileira das Empresas de Software – ABES (2016), houve um crescimento no mercado de desenvolvimento de software de 30,2%, comparando o ano de 2014 com o ano de 2015. Ainda, o mercado brasileiro representa 2,9% do mercado mundial e conta com 10.140 empresas dedicadas ao desenvolvimento e comercialização de software (ABES, 2016).

O aumento da demanda por profissionais qualificados na área de TI, incluindo o desenvolvimento de software, e a agilidade e eficiência exigida pelo mercado, demandam que a formação dos futuros profissionais seja qualificada. Neste âmbito, o conhecimento em programação é parte fundamental para profissionais do mercado de TI e é parte imprescindível dos currículos de formação dos cursos relativos à Ciência da Computação (ACM/IEEE-CS Joint Task Force on Computing Curricula, 2013; SBC, 2005).

Na verdade, há uma tendência em países desenvolvidos de trazer habilidades de programação para ensino em nível equivalente, em geral, ao ensino médio no Brasil (GRANDELL et al., 2006; Ragonis e Ben-Ari, 2005). No Brasil, isso já ocorre em cursos técnicos pertinentes e integrados ao ensino médio (MEC, 2012; IFPR, 2014). Entretanto, ainda a grande maioria da formação de massa crítica nesse âmbito se dá no ensino superior.

De acordo com o guia para currículos de cursos em Ciência da Computação desenvolvido em conjunto pela ACM e pela IEEE (2013), existem algumas áreas de conhecimento que são consideradas parte do conhecimento introdutório dos cursos de Ciências da Computação. Dentre estas áreas de conhecimento, existem áreas que mantém diretamente o foco em programação, seja em um paradigma de

programação ou em uma linguagem de programação (ACM/IEEE-CS Joint Task Force on Computing Curricula, 2013).

Quanto à escolha do paradigma de programação e da linguagem, o guia da ACM e da IEEE (2013) enfatiza que na década inicial dos anos 2000, em vez de apenas um paradigma ou linguagem de programação ser favorecido ao longo do tempo, houve uma ampliação da lista de paradigmas e linguagens que vêm sendo utilizados com sucesso na parte introdutória dos cursos de Ciência da Computação (ACM/IEEE-CS Joint Task Force on Computing Curricula, 2013).

Nos últimos cinquenta anos, centenas de linguagens de programação foram introduzidas, as quais pertencem a diferentes (sub) paradigmas de programação. Em suma e em tempo, um paradigma de programação se constitui em um conjunto de princípios que regem um conjunto de linguagens de programação. No entanto, apesar da diversidade numérica de linguagens e mesmo paradigmas, não há muitas linguagens que sobrevivam mais do que uma década (VUJOŠEVIĆ-JANIČIĆ e TOŠIĆ, 2008).

Com esta diversidade, é importante que se detecte o que é mais efetivo para o ensino dos futuros profissionais de TI, visando prepará-los da melhor forma, sem criar barreiras de aprendizagem. O ensino de programação e, consequentemente, a escolha do paradigma e linguagem que serão ensinados inicialmente, é de suma importância para melhor preparo dos futuros profissionais. Neste contexto, pertinente ressaltar que disciplinas introdutórias à programação são tidas como difíceis pelos estudantes e apresentam grande índice de desistência e reprovação (ROBINS, ROUNTREE e ROUNTREE, 2003; VUJOŠEVIĆ-JANIČIĆ e TOŠIĆ, 2008; KUNKLE e ALLEN, 2016).

A escolha do paradigma e da linguagem a serem ensinados não é unanimidade e é algo que muda ao longo do tempo, sendo inclusive ligada a tendências ditadas pelo mercado. Os paradigmas mais comuns utilizados no ensino de programação para novatos são o Paradigma Procedimental (PP) e o Paradigma Orientado a Objetos (POO), ambos sendo englobados no (mais amplo) Paradigma Imperativo (PI) e cada qual com suas vantagens e desvantagens. Já outros paradigmas, mesmo entre os mais conhecidos ou dominantes, como o Paradigma Funcional (PF) e o Paradigma Lógico (PL), ambos sendo englogados no (mais amplo) Paradigma Declarativo (PD), não são tão utilizados para o ensino de programação conforme relatado por Vujošević-Janičić e Tošić (2008).

É de suma importância que, no processo de escolha do paradigma, a pessoa que está o realizando baseie-se em fatos que corroborem com a escolha. Atualmente a comparação entre paradigmas encontrada na literatura seria baseada nas impressões dos autores a respeito das características de cada linguagem e (ou) paradigma, conforme revisão realizada por Krug (2016b).

Isto considerado, o presente trabalho visa propor um método de comparação entre distintos paradigmas de programação a partir das atividades geradas durante a codificação de programas de computador, utilizando também a classificação dos erros cometidos no processo de codificação.

A proposta inicial para o trabalho era "apenas" realizar o comparativo entre o Paradigma Imperativo-Procedimental com o Paradigma Declarativo-Lógico, especificamente Sistemas Baseados em Regras (SBRs), quanto ao aprendizado de novatos em programação. Entretanto, durante o levantamento da literatura, poucas informações de como realizar este comparativo foram encontradas.

Desta forma, durante a evolução do trabalho, os objetivos do mesmo evoluíram de forma tal a propor um método de comparação que, por sua vez, será validado com a comparação entre os paradigmas justo acima mencionados.

A escolha dos paradigmas justifica-se devido ao fato do Paradigma Imperativo-Procedimental ser o paradigma mais escolhido como primeiro paradigma, seja devido à inércia nas pesquisas relacionadas aos paradigmas para ensino ou a tendência em seguir o mercado (VUJOŠEVIĆ-JANIČIĆ e TOŠIĆ, 2008). Por sua vez, a escolha do Paradigma Lógico, especificamente Sistemas Baseados em Regras, deve-se ao fato do mesmo mostrar-se promissor quanto à facilidade de programação e entendimento, o que encontra apoio na literatura (SHIMOKURA, NAKANISHI e OHTA, 2008; ARAKLIOTIS, NIKOLOS e KALLIGEROS, 2016).

A suposta facilidade de aprendizagem de SBRs é relatada por Shimokura, Nakanishi e Ohta (2008), que descreve brevemente um experimento com jovens estudantes, entre 16 e 17 anos, sem experiência com programação, que tiveram uma breve explicação sobre o desenvolvimento com a linguagem de SBR nominada STAR/ESTR e apresentaram grande desempenho no desenvolvimento com esta linguagem.

Da mesma maneira, Arakliotis, Nikolos e Kalligeros (2016) descrevem brevemente a escolha de SBR para ensino de programação para estudantes

primários. Nesse trabalho relatam que escolheram SBR devido a esta forma de programação aparentar ser mais fácil para o aprendizado.

A escolha de SBR mostra-se promissora também devido ao crescimento de interesse desta técnica pelo mercado. Neste sentido, Berstel e Lecont (2010) afirmam que SBRs vêm ganhando o interesse da indústria, pois demonstram uma forma de separar as regras de negócios das aplicações das entidades tratadas por ela. Desta forma, com esse desacoplamento, baixa-se o custo das frequentes alterações, causadas por fatores como atualizações de legislação ou competitividade do negócio.

Sendo assim, neste quadro exposto, a hipótese que o trabalho visa a corroborar é de que é possível comparar paradigmas de programação através das atividades (i.e. erros, latência, performance de codificação etc) geradas durante a codificação de programas de computador. Desta forma, procurando colaborar com o processo de ensino-aprendizagem de novatos em programação, aumentando assim a qualidade do artefato gerado.

## 1.1. OBJETIVOS

Em linha com a hipótese, o objetivo geral e os objetivos específicos a seguir foram elencados. Os objetivos são válidos para a dissertação como um todo.

### 1.1.1. Objetivo geral

O objetivo geral para deste trabalho é propor método de comparação entre paradigmas de programação, no tocante à aprendizagem de novatos em programação, por meio de dados coletados em plano de fundo (e.g. dados linguísticos, latência e performance de codificação) e a subsequente categorização de erros visando o ajuste de ruídos pedagógicos no ensino de programação e a consequente melhoria da qualidade do artefato gerado.

### 1.1.2. Objetivos específicos

Os objetivos específicos são:

- Definir método de comparação de paradigmas de programação utilizando a categorização de erros gerados na codificação de programas de computador e por meio da coleta de informações em

plano de fundo, para verificar o desempenho dos discentes em cada (técnica de) cada paradigma de programação considerado;

- Aplicar e validar o método através do comparativo entre o Paradigma Imperativo-Procedimental e o Paradigma Declarativo-Lógico, especificamente Sistemas Baseados em Regras;

- Utilizar o método, através do comparativo, para avaliar se o ensino de Paradigma Lógico, especificamente Sistemas Baseados em Regras, pode ser utilizado com novatos em programação.

## 1.2. ESTRUTURA DO TRABALHO

O presente trabalho está dividido em 7 capítulos. O primeiro apresenta a introdução do trabalho, incluindo hipótese e objetivos.

O segundo capítulo apresenta a revisão bibliográfica do tema, sendo que o conteúdo está distribuído em seções que apresentam a definição de paradigmas e de paradigmas de programação, os paradigmas discutidos e trabalhos relacionados com comparativo entre paradigmas de programação.

Por sua vez, o terceiro capítulo apresenta o método para comparação de paradigmas proposto, incluindo a coleta de dados, a classificação e a categorização dos artefatos.

Na sequência, o quarto capítulo apresenta os experimentos realizados até então, incluindo a escolha de paradigmas, a população do experimento, os problemas aplicados e a coleta de dados.

O capítulo 5 apresenta os dados coletados durante o experimento, enquanto o capítulo 6 apresenta a discussão dos dados apresentados.

Finalmente o sétimo e último capítulo apresenta a conclusão do trabalho.

# 2. REVISÃO BIBLIOGRÁFICA

Com o objetivo de dar base a discussões e descrever o cenário de pesquisa atual, trabalhos anteriores foram escritos e são reaproveitados e melhorados neste capítulo.

O primeiro trabalho, apresentado no Seminário I, trata sobre paradigmas de programação, característica dos paradigmas quanto à aprendizagem, ensino de programação, modelo de compreensão de programas e estudos comparativos entre paradigmas (Krug, 2016b).

O segundo trabalho é uma Revisão Sistemática da Literatura escrita sobre Sistemas Baseados em Regras que descreve ferramentas de SBRs e soluções implementadas com esta técnica de programação (Krug, 2017). A revisão mencionada encontra-se na Seção Complementar A deste trabalho.

## 2.1. PARADIGMA[1]

A definição de paradigma, no sentido amplo, pode ser dada como um modelo ou padrão seguido ou a ser seguido por uma comunidade ou grupo de pessoas. Para uma definição mais completa, levando em consideração as ciências físicas, pode-se tomar como base a definição proposta na primeira edição da obra “The Structure of Scientific Revolutions” (KUHN, 1962) e transcrita na definição de paradigma pelo dicionário Aurélio (FERREIRA, 2006) como as “realizações científicas que geram modelos que, por um período mais ou menos longo e de modo mais ou menos explícito, orientam o desenvolvimento posterior das pesquisas exclusivamente na busca da solução para os problemas por elas suscitados” (FERREIRA, 2006).

Embora Kuhn pretendesse dar esta definição para o termo paradigma, ela não ficou claramente representada em sua obra, levando a diferentes interpretações do termo. A utilização de diferentes interpretações do termo pôde ser observada logo em seguida à publicação da obra de Kuhn. Desde então, o termo começou a ser utilizado em textos científicos, nos quais raramente a definição pretendida era utilizada (MASTERMAN, 1970; BANASZEWSKI, 2009).

---

1 Conteúdo reaproveitado e melhorado a partir do documento utilizado para Seminário I, 1. PPGCA/DAINF/UTFPR. Adaptado de Krug (2016b).

Durante o Seminário Internacional sobre Filosofia da Ciência em Londres, no ano de 1965, várias críticas foram realizadas sobre a definição imprecisa do termo paradigma feita por Kuhn, sendo a crítica mais relevante a realizada por Margareth Masterman (MASTERMAN, 1970). Segundo Masterman, o termo foi definido em 21 formas diferentes por Kuhn, porém, preservando a similaridade entre as definições. Analisando as similaridades, Masterman reduziu o leque de interpretações, agrupando em três categorias: Paradigma Metafísico; Paradigma Sociológico; e Paradigma Exemplar (MASTERMAN, 1970; BANASZEWSKI, 2009).

De maneira sucinta, é possível detalhar as categorizações realizadas por Masterman (1970) da seguinte forma:

**Paradigma Metafísico:** representado por uma crença profunda em um modelo de ciência dentro de uma área particular de conhecimento. A comunidade que crê em determinada ciência apresenta forte fidelidade aos conceitos defendidos, mesmo que estes sejam criticados e contestados por outros modelos.

**Paradigma Sociológico:** representado por fatores (crenças, valores e técnicas) que unem uma determinada comunidade (científica). Os fatores são entendidos como o interesse em comum da comunidade sobre um conhecimento particular.

**Paradigma Exemplar:** representa o significado para paradigma segundo a intenção de Kuhn (1962), definição do termo adotada pelo dicionário Aurélio, sendo “realizações científicas que geram modelos que, por um período mais ou menos longo e de modo mais ou menos explícito, orientam o desenvolvimento posterior das pesquisas exclusivamente na busca da solução para os problemas por elas suscitados” (FERREIRA, 2006).

No posfácio da segunda edição da obra escrita por Kuhn, o autor faz uma autocrítica, reconhecendo a imprecisão da definição do termo paradigma, atribuindo a diferença de interpretações às “incongruências estilísticas” (KUHN, 1970).

Das três categorias definidas por Masterman (MASTERMAN, 1970), Kuhn reconheceu apenas duas, sendo uma mais genérica, semelhante à definição de Paradigma Sociológico, e outra mais específica, semelhante à definição de Paradigma Exemplar.

Na definição específica de Kuhn, alinhada com a definição de Paradigma Exemplar realizada por Masterman (1970), paradigma se refere a um modelo seguido por uma comunidade. Um modelo consiste no conhecimento adquirido por

uma comunidade sobre um determinado domínio, utilizado como base para a comunidade expandir o seu conhecimento e o entendimento sobre o referido domínio. Os modelos são constituídos de métodos e técnicas concebidas a partir de trabalhos realizados pela comunidade a fim de guiar os participantes na solução de novos problemas, podendo resolvê-los com a ajuda de ferramentas (KUHN, 1970; BANASZEWSKI, 2009).

## **2.2.** PARADIGMAS DE PROGRAMAÇÃO[2]

Em computação, a palavra paradigma pode ser aplicada em diversos contextos, como paradigma de arquitetura computacional, paradigma de projeto de sistemas e paradigma de programação. Entretanto, quando se fala de paradigma em Ciência da Computação, não raro se remete a paradigmas de programação (KAISLER, 2005). Em todo caso, este trabalho em particular foca nos paradigmas de programação.

De forma sucinta, paradigma de programação é o meio (ou modelo) utilizado para compreender problemas do mundo real, buscando elaborar uma solução computacional (WATT, 2004; ROY e HARIDI, 2004; BANASZEWSKI, 2009).

Um paradigma de programação é materializado através de uma linguagem de programação, acompanhada também de outras ferramentas de análise e projeto (KAISLER, 2005). Importante mencionar que existem linguagens de programação de um único paradigma e de múltiplos paradigmas (ROY e HARIDI, 2004). Geralmente é possível associar o paradigma de programação baseando-se na linguagem de programação que está sendo utilizada (BROOKSHEAR, 2006).

Isto dito, as definições utilizadas para paradigma nas demais ciências também são válidas para definir paradigma de programação. Mais precisamente, as definições escritas por Kuhn (1962) e categorizadas por Masterman (1970), podem ser utilizadas como auxílio na compreensão e definição de paradigma de programação:

– É possível associar o Paradigma Metafísico com uma comunidade de programadores que utiliza e apoia apenas uma linguagem e um paradigma de programação, ignorando as críticas e contestações realizadas a eles.

---

2 Conteúdo reaproveitado e melhorado a partir do documento utilizado para Seminário I, 1. PPGCA/DAINF/UTFPR. Adaptado de Krug (2016b).

– Por sua vez, a definição de Paradigma Sociológico condiz com a definição de paradigma de programação devido ao fato de que um grupo de programadores deve compartilhar uma mesma forma de ver e pensar sobre os problemas, permitindo colaboração em suas ações.

– A categorização preferida por Kuhn, o Paradigma Exemplar, também se adéqua à definição de paradigma de programação, pois uma comunidade científica segue um modelo que representa uma forma particular de enxergar os problemas do mundo real a serem tratados computacionalmente (BANASZEWSKI, 2009).

Pertinente mencionar que a origem da categorização de paradigma realizada por Masterman (1970) surgiu de uma única ideia de definição de paradigma. Desta forma, existe semelhança entre as categorizações.

Apesar das definições de paradigma para as ciências em um modo geral adequarem-se à definição de paradigma para a Ciência da Computação, é importante ter clara a definição utilizada de forma específica para um paradigma de programação.

Um paradigma de programação consiste e difere-se dos demais por meio da: (a) seleção de conceitos de programação (e.g. tipos de dados, escopo, abstração etc.) e a forma de suas utilizações; e (b) interação com qual se dita a forma de execução do programa, em conjunto com a forma e o estilo de programação (ROY e HARIDI, 2004; WATT, 2004; BROOKSHEAR, 2006; BANASZEWSKI, 2009; RONSZCKA, 2012).

Peter Van Roy define um paradigma de programação como um sistema formal, o qual define como a programação é realizada, sendo que cada paradigma tem o seu próprio conjunto de técnicas para programação e uma forma de estruturar o pensamento na concepção dos programas (ROY e HARIDI, 2004; VAN ROY, 2009; BANASZEWSKI, 2009).

Atualmente, é possível classificar os paradigmas de programação existentes em paradigmas dominantes e paradigmas emergentes. Os paradigmas dominantes são aqueles que vêm sendo utilizados ao longo do tempo por um considerável número de pessoas e que estão consolidados há algum tempo (BANASZEWSKI, 2009).

Por sua vez, entende-se por paradigmas emergentes àqueles que ainda não estão consolidados ou não receberam um grau de importância tão grande quanto os dominantes. Geralmente, os paradigmas emergentes surgem para melhorar os

paradigmas dominantes, atuando nos pontos falhos apresentados por estes (BANASZEWSKI, 2009).

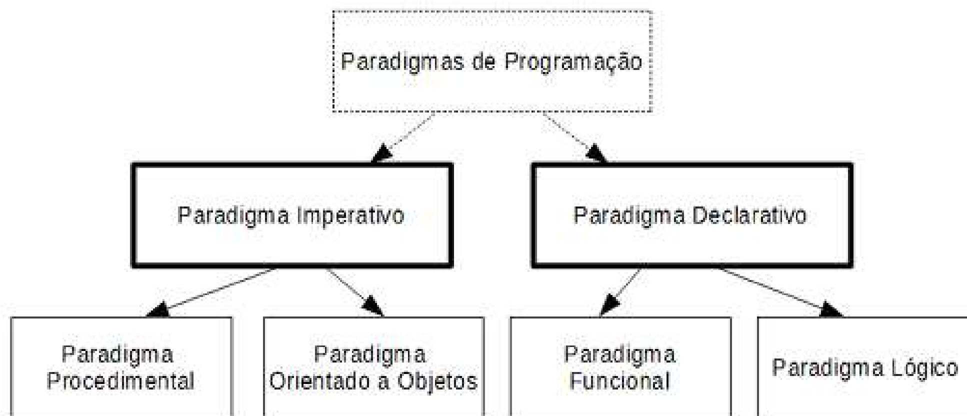



Figura 1 – Paradigmas dominantes – Adaptado de BANASZEWSKI, 2009.

Isto considerado, é possível categorizar os paradigmas dominantes em dois grupos, conforme ilustrado na Figura 1, Paradigma Imperativo (PI) e Paradigma Declarativo (PD). Dentro destes grupos ainda é possível nova divisão, sendo que para o PI ter-se-ia o Paradigma Procedimental (PP) e o Paradigma Orientado a Objetos (POO). Por sua vez, para o PD ter-se-ia o Paradigma Funcional (PF) e o Paradigma Lógico (PL). Pertinente ressaltar que esta classificação, embora não unânime, tem concordância de vários autores (WATT, 2004; SCOTT, 2000; BROOKSHEAR, 2006).

Ainda assim, na prática há intersecções entre os paradigmas sendo esta representação apenas um modelo geral das tipificações. Neste sentido, Peter Van Roy (2009) apresentou uma taxonomia de como os paradigmas de programação são relacionados e qual é o caminho das linguagens até os paradigmas e conceitos relacionados a eles. Esta taxonomia está ilustrada na Figura 2 sendo pertinente mencionar que cada linguagem de programação pertence a um ou vários paradigmas. Cada paradigma é definido por um conjunto de conceitos de programação e organizado em uma linguagem básica simples, chamada de *kernel language* (VAN ROY, 2009).

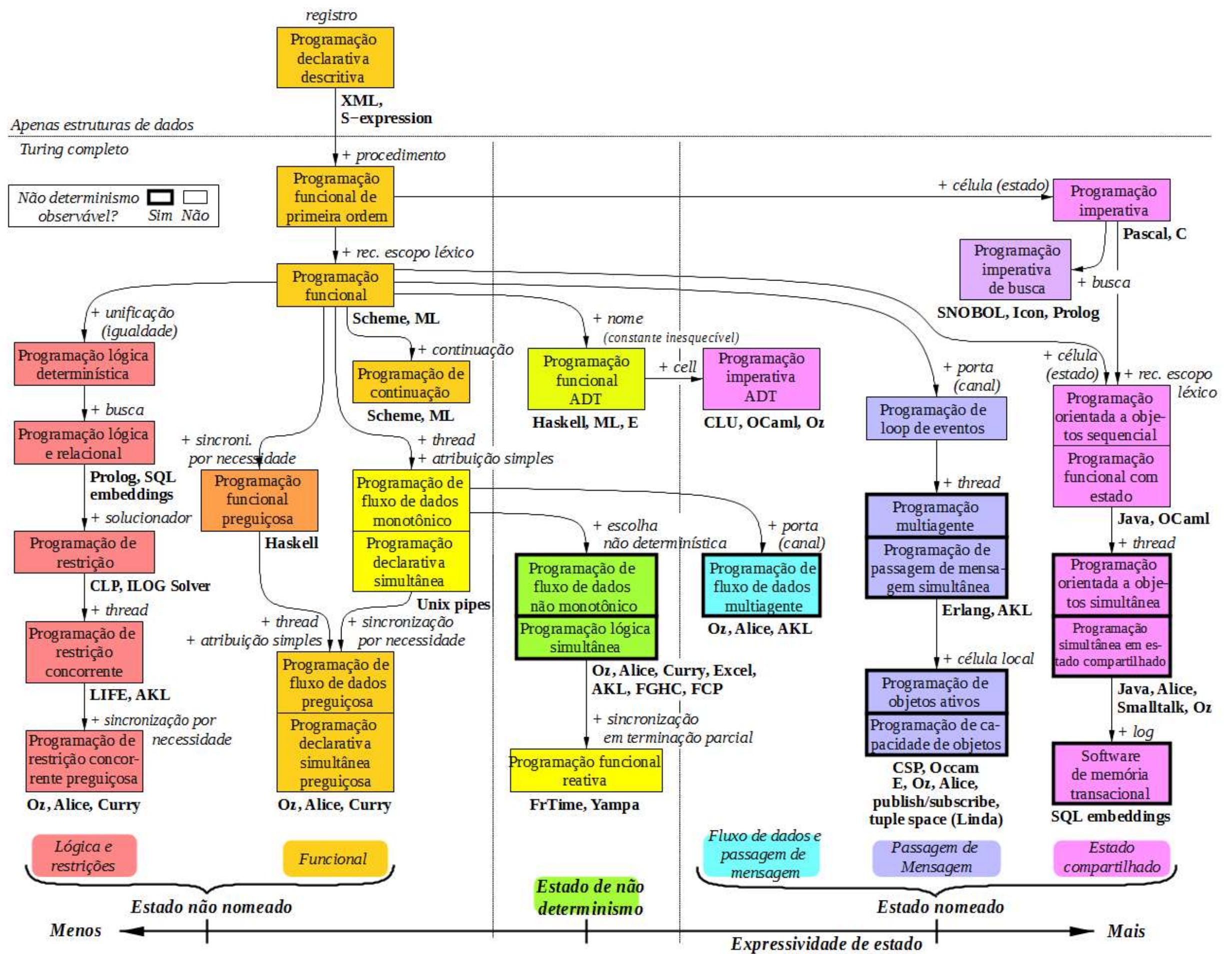



Figura 2 – Taxonomia de paradigmas de programação - Adaptado de VAN ROY, 2009.

A classificação de paradigmas pode ser complexa à luz do trabalho de Van Roy (2009), classificando os paradigmas de acordo com conceitos de registros (*record*), recipientes com escopo léxico (*closures*), independência (concorrência) e estado nomeado (*named state*), além de determinismo observável. Esta classificação pode ser considerada uma forma de comparação entre paradigmas e exemplifica a complexidade do tema.

Os paradigmas pertinentes ao trabalho são relatados nas próximas subseções.

A fim de exemplificar a diversidade de linguagens utilizadas atualmente, a Figura 3 ilustra um *ranking* das linguagens mais populares. Este ranking, elaborado pela IEEE (2016), combina 12 métricas de 10 fontes distintas para produzir os resultados de popularidade das linguagens. A pontuação é balanceada, desta forma, a pontuação da linguagem mais bem ranqueada é de 100 pontos e a pontuação final

das demais é calculada de forma proporcional de acordo com a mais bem ranqueada.

Entre as primeiras linguagens temos linguagens dos paradigmas PP (C e GO) e POO (Java, C++ e C#), além de linguagens consideradas multiparadigmas (Python, R, PHP, JavaScript e Ruby). Como entre as 10 mais populares não foram apresentadas linguagens do PD. Estas foram retiradas dentre o restante da lista, sendo linguagens do PF (Haskell, Lisp, Erlang e Clojure) e do PL (SQL e Prolog).

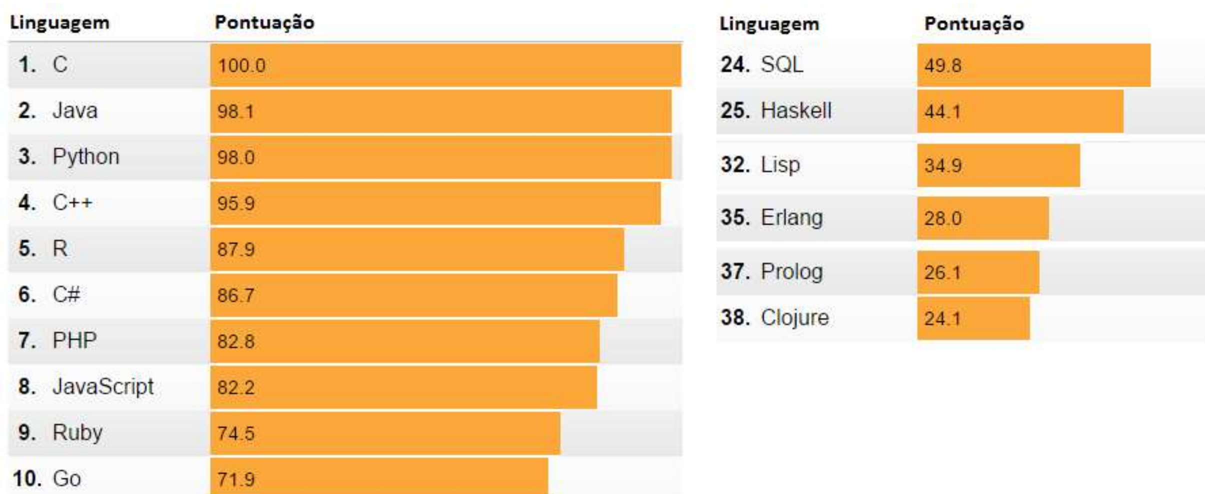



Figura 3 – Ranking de linguagens de programação (IEEE, 2016).

Existem outros *rankings* de linguagens de programação, por exemplo, o ranking TIOBE[3]. Entretanto, o *ranking* escolhido foi o da IEEE[4] por apresentar um conjunto de fontes de informação utilizado para chegar à pontuação de cada linguagem.

Tendo esta diversidade de paradigmas, as próximas duas subseções tratam especificamente os paradigmas de programação considerados como dominantes.

## **2.3.** PARADIGMA IMPERATIVO[5]

O Paradigma Imperativo (PI) recebeu este nome devido à forma sequencial que suas instruções são processadas pelos computadores. Esta sequencialidade é uma das principais características deste paradigma, estando presente nos (sub)

---

3 http://www.tiobe.com/tiobe-index/

4 http://spectrum.ieee.org/computing/software/the-2016-top-programming-languages

5 Conteúdo reaproveitado e melhorado a partir do documento utilizado para Seminário I, 1. PPGCA/DAINF/UTFPR. Adaptado de Krug (2016b).

Paradigmas Procedimental (PP) e Orientado a Objetos (POO), que fazem parte de seu escopo (WATT, 2004; SCOTT, 2000; BROOKSHEAR, 2006).

Os paradigmas que fazem parte do PI, principalmente através de suas linguagens mais antigas, também são conhecidos pela sua proximidade e controle do hardware, tornando o programador mais próximo da máquina. Isto permite melhores usos de recurso computacional, embora traga alguma ou mesmo considerável complexidade na codificação dos programas (VUJOŠEVIĆ-JANIČIĆ e TOŠIĆ, 2008).

Dentre os paradigmas do PI, pode-se classificar dois paradigmas dominantes, o PP e o POO. Ambos os paradigmas são similares, principalmente nas características que os classificam como PI, mas entre suas diferenças destaca-se a forma como são representados e na organização de suas instruções (WATT, 2004; SCOTT, 2000; BROOKSHEAR, 2006).

Conforme Banaszewski (2009), a essência do PI consiste na organização sequencial do programa por meio de comandos em uma linguagem procedimental ou orientada a objetos, os quais são executados sequencialmente pelo mecanismo interno destas linguagens. Basicamente, este mecanismo consiste em buscas sobre as entidades passivas, os dados (e.g. variáveis, listas e vetores) e estruturas de decisão (e.g. *se-então* e *escolha-de-casos*).

O Paradigma Procedimental permaneceu como o paradigma mais utilizado até a década de 1990, quando foi desafiado pelo POO. Grande parte dos softwares comerciais eram desenvolvidos utilizando o PP, assim como vários softwares ainda continuam sendo desenvolvidos neste paradigma (WATT, 2004; KING, 2008; BANASZEWSKI, 2009).

Conforme Watt (2004) são conceitos chaves do PP variáveis, comandos e procedimentos. Entidades do mundo real podem ser modeladas por variáveis e processos do mundo real por comandos, que inspecionam e atualizam o valor destas variáveis. As variáveis também são utilizadas para armazenar valores auxiliares em determinados momentos. Procedimentos, também conhecidos como funções, são utilizados para modularizar o programa em PP, podendo dar certa abstração ao programa.

Em resumo, um programa escrito utilizando o PP é uma sequência de ordens (comandos) para executar ações. Entre estas ações pode-se ter a atribuição de valores às variáveis, e também a comparação de valores atribuídos a estas

variáveis. Levando sempre em consideração a ordem em que as atribuições e comparações ocorrem, pois elas devem ser sequenciais de acordo com o objetivo do programa.

Embora o PP não seja atualmente o paradigma mais utilizado, ele ainda é bastante utilizado devido sua facilidade de modularização, sua simplicidade e eficiência.

Dentre as linguagens de programação do PP, destacam-se o ALGOL, BASIC, C, COBOL, Fortran e Pascal.

O Paradigma Orientado a Objetos tem semelhanças com o PP, por exemplo, a forma de execução, por isso faz parte do PI, utilizando-se de comandos executados de forma sequencial. Um dos fatos que difere o POO do PP é a sua capacidade de abstração e a melhor forma de estruturação do código.

No POO a estrutura simulada no programa deve refletir a estrutura do ambiente de mundo real que está sendo utilizado como base. Comparando com outros paradigmas, o POO altera a ênfase de dados como elementos passivos para elementos ativos interagindo com o ambiente.

Ainda assim, é interessante uma passagem de Robins, Rountree e Rountree (2003) que cita que o POO não é diferente do PP, ele “é mais”, pois o POO adiciona “despesas” da estrutura de classes a um sistema procedural. Entretanto, o conceito de abstração, destaque do POO, tem-se mostrado cada vez mais necessário devido à complexidade dos aplicativos computacionais e essencial para a atividade de programação e mesmo a Engenharia de Software em geral. Dentre as linguagens de programação do POO, destacam-se o C++, C#, Java, Python, sendo as linguagens pioneiras do POO o Simula 67 e o Smalltalk.

Em tempo, objetos na POO podem ser compreendidos como representação de entidades ou objeto do mundo (real ou imaginário). Tais entidades podem ser elementos ativos (e.g. pessoa e pássaro), inativos (e.g. mesa e porta) e o abstrato (e.g. arquivo, tabela e botão de uma interface gráfica). Fazem parte dos objetos os estados e os comportamentos que agem sobre seus próprios estados (e.g. a *pessoa* X estava *sentada* e agora está *em pé*) ou ainda agem sobre o comportamento de outros objetos (e.g. *pessoa* X abriu a *porta* Y) (BANASZEWSKI, 2009).

Watt (2004) descreve os conceitos chaves do POO sendo objetos, classes, herança e polimorfismo. Objetos são formados por atributos e métodos, e permitem uma forma natural de mapear as entidades do mundo real. Classe é a forma de criar

modelos para os objetos, com atributos e métodos similares. Herança é a possibilidade de criar subclasses, onde estas herdam as características da classe referenciada. Polimorfismo permite que um objeto de uma subclasse possa ser tratado como um objeto de uma superclasse, e por consequência permite que seja construída uma coleção heterogênea de objetos de classes distintas.

A partir da explicação das características dos paradigmas PP e POO, algumas das diferenças entre os paradigmas são demonstradas através de um exemplo utilizado por Banaszewski (2009)[6] onde um mesmo programa é escrito em PP utilizando a linguagem C e em POO utilizando a linguagem C++.

```c
#include <stdio.h>

struct Pessoa
{
    int dia;
    int mes;
    int ano;
    int idade;
};

int CalcularIdade(struct Pessoa p, int ano)
{
    int idade = ano - p.ano;
    return idade;
}

int main()
{
    struct Pessoa Einstein, Newton;

    Einstein.dia = 14;
    Einstein.mes = 3;
    Einstein.ano = 1879;

    Newton.dia = 4;
    Newton.mes = 1;
    Newton.ano = 1643;

    Einstein.idade = CalcularIdade(Einstein, 2016);
    Newton.idade = CalcularIdade(Newton, 2016);

    printf("A idade de Einstein seria %d \n",
    Einstein.idade);
    printf("A idade de Newton seria %d   \n",
    Newton.idade);

    getchar();
    return 0;
}
```

```cpp
#include <stdio.h>

class Pessoa
{
  private:
    int diaNascimento, mesNascimento, anoNascimento;
    int idade;

  public:
    Pessoa (int diaNasc, int mesNasc, int anoNasc) {
        diaNascimento = diaNasc;
        mesNascimento = mesNasc;
        anoNascimento = anoNasc;
    }

    void calcularIdade (int anoAtual) {
        idade = anoAtual - anoNascimento;
    }

    int getIdade () {
        return idade;
    }

};

int main()
{
    Pessoa Einstein (14, 3, 1879);
    Pessoa Newton (4, 1, 1643);

    Einstein.calcularIdade(2016);
    Newton.calcularIdade(2016);

    printf("A idade de Einstein seria %d \n",
    Einstein.getIdade());
    printf("A idade de Newton seria %d   \n",
    Newton.getIdade());

    getchar();
    return 0;
}
```

Figura 4 – Comparação entre C e C++.

O programa está ilustrado na Figura 4, sendo a imagem da esquerda o programa escrito na linguagem C e a da direita escrito em C++. A função de ambos

6 Exemplo criado com auxílio dos materiais da disciplina “Fundamentos de Programação”, disponíveis em http://ppgca.dainf.ct.utfpr.edu.br/~jeansimao/Fundamentos1/LinguagemC++/Fundamentos1-LinguagemC++.htm. Acesso em: 30 nov. 2016.

os programas é calcular a idade de uma pessoa a partir de um ano passado como parâmetro.

O programa escrito na linguagem C utiliza uma estrutura *struct*, que é uma coleção de variáveis a fim de representar uma entidade. Pertinente mencionar que uma estrutura é considerada uma evolução do PP, visto que versões mais antigas de linguagens deste paradigma não contavam com este recurso. Apesar de possibilidade de representar uma entidade do mundo real, uma estrutura não suporta a definição de funções. Esta é uma das características que diferencia esta estrutura modular das classes do POO. Desta forma, uma classe também se apresenta como uma coleção, mas com a capacidade de organizar dados e também funções (LAFORE, 2002; BANASZEWSKI, 2009).

No programa escrito em C foi criada uma função *CalcularIdade*, a qual é independente da estrutura *Pessoa*, devendo assim receber o endereço da estrutura para que possa atuar sob seus estados (variáveis). Já no exemplo escrito em C++ as funções da classe (chamadas de métodos) acessam seus estados (chamados de atributos) sem a necessidade de utilizá-los como parâmetros (BANASZEWSKI, 2009).

Apesar da semelhança do PP e do POO neste exemplo, estes ainda se mostram diferentes na essência da programação. No PP, a essência é decompor o programa em variáveis, estruturas de dados e funções. No POO, a decomposição ocorre de forma mais abstrata, por exemplo, classes, objetos, métodos e atributos.

Ainda que POO e PP tenha diferenças, um fato é que eles têm a mesma essência baseada no PI, o qual apresenta deficiências. Em relação a deficiências do PI, Banaszewski (2009) cita que devido à sequencialidade de busca e a passividade dos elementos nas linguagens do PI, as linhas de códigos se tornam interdependentes e existem problemas de redundância na execução dos programas. As expressões causais são avaliadas passivamente causando redundâncias temporais e redundâncias estruturais. Redundância temporal consiste na avaliação desnecessária e repetida de expressões causais na presença de estados já avaliados e inalterados. Redundância estrutural ocorre quando o conhecimento de um valor Booleano (verdadeiro ou falso) de uma expressão lógica não é compartilhado com outras expressões pertinentes, causando reavaliações desnecessárias.

Para exemplificar as redundâncias temporais e redundâncias estruturais a Figura 5 ilustra um exemplo adaptado de Simão et al. (2012). No exemplo, cada expressão causal tem três premissas lógicas e uma estrutura de repetição exige a validação sequencial de todas as expressões causais. Entretanto, a maioria das validações não são necessárias porque, usualmente, apenas alguns atributos (i.e. variável) têm seu valor alterado a cada iteração (SIMÃO et al., 2012).

No exemplo, a redundância temporal ocorre através da validação repetida de elementos já validados e com seu valor inalterado (SIMÃO et al., 2012).

A redundância estrutural, por sua vez, é exemplificada através da utilização recorrente da mesma expressão lógica em duas ou mais expressões causais. Neste caso, a expressão lógica (objeto1.atributo1 = 1) é replicada em diversas expressões causais (i.e., se-então) (SIMÃO et al., 2012).

```
1   enquanto (verdade) faça
2       se ((objeto1.atributo1 = 1) e
3           (objeto2.atributo1 = 1) e
4           (objeto3.atributo1 = 1))
5       então
6           objeto1.método1();
7           objeto2.método1();
8           objeto3.método1();
9       fim-se
10      se ((objeto1.atributo2 = 2) e
11          (objeto2.atributo2 = 2) e
12          (objeto3.atributo2 = 2))
13      então
14          objeto1.método2();
15          objeto2.método2();
16          objeto3.método2();
17      fim-se
18      ...
19      se ((objeto1.atributoN = n) e
20          (objeto2.atributoN = n) e
21          (objeto3.atributoN = n))
22      então
23          objeto1.métodoN();
24          objeto2.métodoN();
25          objeto3.métodoN();
26      fim-se
27  fim_enquanto
```

Figura 5 – Exemplo de redundância temporal e estrutural – Adaptado de SIMÃO et al., 2012.

## **2.4.** PARADIGMA DECLARATIVO[7]

O Paradigma Declarativo (PD), diferentemente do PI, propicia ao programador a possibilidade de focar mais na organização do conhecimento para a solução do problema computacional do que na forma de implementação do mesmo. O PD

7 Conteúdo reaproveitado e melhorado a partir do documento utilizado para Seminário I, 1. PPGCA/DAINF/UTFPR. Adaptado de Krug (2016b).

possibilita uma interação mais simples do programador com o computador através de meios que ocultam as particularidades de implementação, em outras palavras a utilização do PI implica em “dizer como fazer algo”, enquanto a utilização do PD implica em “dizer o que é requisitado e deixar o sistema determinar como alcançar isso” (ROY e HARIDI, 2004; FRIEDMAN-HILL, 2003).

Porém, para que estas facilidades sejam possíveis, o PD perde em velocidade de execução para o PI e também em flexibilidades, principalmente no que se refere a otimizações algorítmicas e facilidades de acesso ao hardware (SCOTT, 2000; BANASZEWSKI, 2009). Na verdade e em tempo, em algumas linguagens do PI de mais alto nível (como Java e C# do POO), a questão de velocidade de execução também pode se fazer presente.

Conforme Specht et al. (2007), linguagens do PD de propósito geral são conhecidas por permitir descrições em um alto nível de abstração e prover assertividade ao programa, linguagens declarativas podem ainda explorar concorrência sem requerer conhecimento aprofundado do programador sobre concorrência (SEBESTA, 2005). Elas são baseadas em fundamentos da matemática e têm semântica bem definida.

Assim como no PI cabe a divisão em PP e POO, pode-se subdividir o PD em Paradigma Funcional (PF) e Paradigma Lógico (PL).

No Paradigma Funcional (PF) o modelo computacional baseia-se em funções e em seus argumentos, onde funções podem invocar funções e até mesmo serem passados como argumentos para outras funções. As linguagens de programação consideradas puramente funcionais são Miranda e Haskell, porém existem outras linguagens que também fazem parte do PF, entre elas Clojure, ML, Lisp e Scheme.

O PF é baseado na teoria matemática de funções, principalmente no cálculo Lambda. Desta forma, assim como se calcula substituindo parâmetros em expressões no cálculo Lambda, também se calcula em um programa funcional de alto nível, passando argumentos em funções (SCOTT, 2000). O PF permite ao programador pensar no problema em alto nível de abstração, pensando na natureza do problema em vez de pensar na sequência de execução por um computador (VUJOŠEVIĆ-JANIČIĆ E TOŠIĆ, 2008).

Conforme Watt (2004) o modelo computacional do PF é baseado na aplicação de funções aos argumentos, os conceitos chaves deste paradigma são expressões, funções e parâmetros polimórficos. Expressões são definidas como sendo uma das

essências do paradigma, e têm o propósito de computar valores novos através dos valores antigos. Funções abstraem as expressões, as funções recebem valores passados por argumentos e podem também ser passadas como argumentos. Parâmetros polimórficos permitem que uma função opere com valores de diversos tipos.

Ainda, embora existam diferenças entre o PI (PP e POO) e o PF, ambos têm algo em comum, o programa lê a entrada e escreve a saída, sendo que a saída é dependente da entrada. Desta forma um programa escrito com base no PI (PP e POO) e no PF pode ser visto abstratamente como a implementação de um mapeamento entre entradas e saídas. Por sua vez, o Paradigma Lógico (PL) implementa um relacionamento, apresentando-se como um paradigma de maior nível do que o PI (PP e POO) e o PF (WATT, 2004).

O PL é baseado no cálculo de primeira ordem e enfatiza a descrição declarativa do problema em vez de decompor o problema em uma sequência de comandos.

Existe uma divisão do PL em dois modelos: (1) Sistemas de Dedução (*Backward Chaining*) que inicia com a hipótese e trabalha verificando se existem fatos disponíveis que suportam a hipótese; e (2) Sistemas de Produção (*Forward Chaining*) que utiliza os fatos e usa as regras para extrair mais dados até atingir o objetivo (RICH e KNIGHT, 1991).

Rich e Knight (1991) indicam a utilização de *backward chaining* quando a resolução do problema é direcionada ao objetivo. Por sua vez, indicam a utilização de *forward chaining* quando é necessário solucionar o problema de acordo com os fatos recebidos de fora.

Conforme Coppin (2010), com *backward chaining* parte-se de uma conclusão (hipótese) e tem-se por objetivo mostrar como aquela conclusão pode ser alcançada a partir de regras e fatos da base de dados. Com o *forward chaining* o sistema parte de um conjunto de fatos e de um conjunto de regras e tenta encontrar um meio de usar tais regras e fatos para deduzir uma conclusão ou traçar uma linha de ação.

Isto considerado, pertinente sublinhar que esta subseção trata a definição de Paradigma Lógico de forma geral enquanto a próxima subseção, sobre SBRs, atém-se a definição de Sistemas de Produção.

Um programa escrito no PL pode ser visto como uma coleção de declarações lógicas que descrevem o problema a ser resolvido. Um programa utilizando o PL

contém: (a) axiomas, que definem os fatos a respeito dos objetos; (b) regras, definem maneiras de verificar os fatos; e (c) objetivo, define um teorema, que pode ser testado pelos axiomas e pelas regras.

PL é caracterizado pela programação com relações e inferências. O programador é responsável por especificar as relações lógicas e não por especificar a maneira como as regras de inferência serão aplicadas (VUJOŠEVIĆ-JANIČIĆ E TOŠIĆ, 2008)

A essência do PL se fundamenta no modelo relacional, por meio deste modelo, um mecanismo de inferência realiza busca de valores (fatos) que satisfaçam uma relação (regras) com a finalidade de provar um teorema (objetivo) (BANASZEWSKI, 2009; SCOTT, 2000).

Exemplos de linguagens de programação do PL são PROLOG, CLIPS, Jess e OPS5.

Mesmo que na essência o PI e o PD apresentem diversas diferenças, segundo Banaszewski (2009), ambos combinam quanto a deficiências relacionadas ao acoplamento, visto que no PD também é possível gerar dependência entre os componentes. Entretanto, as redundâncias são mitigadas no PL através de algoritmos que implementam soluções inteligentes para evitar este tipo de problema.

Para exemplificar as diferenças do PI (PP e POO) com o PD (PF e PL) foram escritos programas com o mesmo propósito do utilizado na subseção precedente, baseados em Banaszewski (2009)[8].

O programa do PF foi escrito na linguagem de programação LISP. O programa é apresentado na Figura 6. Nele é possível observar que também foi utilizado estrutura e os valores a ela vinculados. A função *calculaidade* foi criada tendo como parâmetros o ano de nascimento e o ano atual.

Para o PL, o programa ilustrado na Figura 7 foi desenvolvido na linguagem CLIPS, adicionando a verificação de qual das duas pessoas é a mais velha. No programa escrito em CLIPS é possível verificar a utilização da base de fatos, através de uma estrutura para armazenar as informações de cada pessoa e também o ano atual, que é informado em tempo de execução. Também foram criadas as regras, sem se importar com a sequencialidade da execução do programa.

---

8 Exemplo criado com auxílio dos materiais da disciplina “Fundamentos de Programação”, disponíveis em http://ppgca.dainf.ct.utfpr.edu.br/~jeansimao/Fundamentos1/LinguagemC++/Fundamentos1-LinguagemC++.htm. Acesso em: 30 nov. 2016.

```
1  (defstruct pessoa
2      dia
3      mes
4      ano
5  )
6
7  (defun calculaidade (anoNasc anoAtual)
8      (- anoNasc anoAtual)
9  )
10
11 ( setq einstein ( make-pessoa
12     :dia 14
13     :mes 3
14     :ano 1870)
15 )
16
17 ( setq newton ( make-pessoa
18     :dia 4
19     :mes 1
20     :ano 1643)
21 )
22
23 (format t "A idade de Einstein seria: ~F ~%" (calculaidade 2016 (pessoa-ano einstein)))
24 (format t "A idade de Newton seria: ~F ~%" (calculaidade 2016 (pessoa-ano newton)))
```

Figura 6 – Programa em LISP.

```
1  (deftemplate dados-pessoais
2      (slot nome)
3      (slot dia)
4      (slot mes)
5      (slot ano)
6      (slot idade)
7  )
8
9  (deffacts pessoas
10     (dados-pessoais (nome Einstein) (dia 14) (mes 3) (ano 1879) (idade 0))
11     (dados-pessoais (nome Newton) (dia 4) (mes 1) (ano 1643) (idade 0))
12 )
13
14 (defrule calcula-idade
15     (ano-atual ?ano-atual)
16     (dados-pessoais (nome ?nome) (idade 0))
17     ?pessoa <- (dados-pessoais (nome ?nome) (ano ?ano) (idade ?idade))
18 =>
19     (modify ?pessoa (idade (- ?ano-atual ?ano)))
20     (printout t ?nome " tem " (- ?ano-atual ?ano) " anos." crlf)
21 )
22
23 (defrule e-mais-velho
24     (ano-atual ?)
25     (dados-pessoais (nome Einstein) (idade ?ie))
26     (dados-pessoais (nome Newton) (idade ?in))
27     (test (> ?ie ?in))
28 =>
29     (printout t "Einstein é o mais velho." crlf)
30 )
31
32 (defrule n-mais-velho
33     (ano-atual ?)
34     (dados-pessoais (nome Einstein) (idade ?ie))
35     (dados-pessoais (nome Newton) (idade ?in))
36     (test (> ?in ?ie))
37 =>
38     (printout t "Newton é o mais velho." crlf)
39 )
40
41 (defrule recebe-ano
42     (not (ano-atual ?))
43 =>
44     (printout t "Digite o ano atual: ")
45     (assert (ano-atual (read))))
```

Figura 7 – Programa em CLIPS.

Como o PL, especificamente Sistemas Baseados em Regras (SBR), foi escolhido para validar o método de comparação, a próxima subseção trata especificamente das definições de SBR, aproveitando-se de alguns conceitos vistos para o PL.

## 2.5. SISTEMAS BASEADOS EM REGRAS[9]

Os Sistemas Baseados em Regras (SBR) foram baseados nos trabalhos de Newell e Simon (1972), no qual pretendiam simular o raciocínio humano através de um mecanismo de inferência que relaciona fatos e regras. Eles observaram que muitos dos problemas que o ser humano resolve podem ser expressos em regras lógico-causais. Também foi possível constatar que o cérebro é estimulado a partir de entradas sensórias, i.e. fatos.

Com isso, considerou-se que há dois tipos de bases de conhecimento. Há a base de fatos e há também a base de regras para relacionar fatos. Esses podem ser armazenados em entidades chamadas “elementos da base de fatos” e aqueles em regras de uma “base de regras”. Os elementos da base de fatos são usados para armazenar temporariamente o conhecimento necessário para a resolução de problemas (RICH e KNIGHT, 1991). Os estímulos (fatos) ativam algumas regras, da base de regras, com o intuito de produzir uma resposta apropriada para um determinado problema (NEWELL e SIMON, 1972).

Assim, a arquitetura de um SBR, baseado nas observações de Newell e Simon (1972), consiste em uma memória para armazenar as regras (Base de Regras), uma memória para armazenar os fatos (Base de Fatos) e a Máquina de Inferência (MI). Em suma, a MI é um elemento processador capaz de inferir sobre estas duas bases. A ilustração desta arquitetura é apresentada na Figura 8 (FRIEDMAN-HILL, 2003).

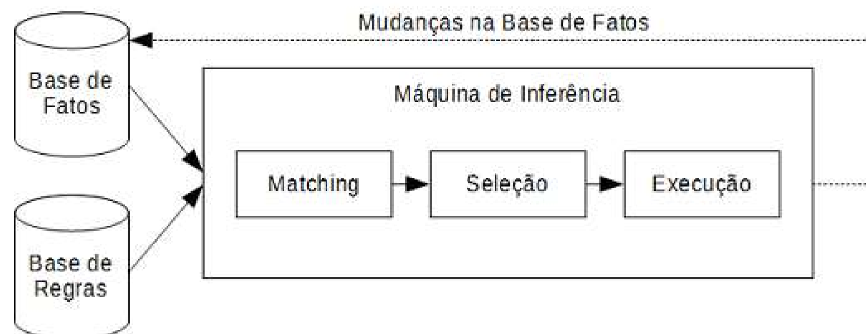



Figura 8 – Arquitetura de um Sistema Baseado em Regras

9 Conteúdo reaproveitado e melhorado a partir de Revisão Sistemática da Literatura escrita sobre Sistemas Baseados em Regras. Estudo Individual, PPGCA/DAINF/UTFPR. Adaptado de Krug (2017).

Utilizando o exemplo ilustrado na Figura 7, a base de fatos é composta inicialmente pela estrutura *pessoas*, que contém dados a respeito de Einstein e Newton. Estes fatos são carregados no momento de execução de programa. Durante a execução do programa ocorre a criação de um novo fato, que recebe o ano atual para poder calcular a idade de cada pessoa.

Por sua vez, a base de regras é composta pelas regras *calcula-idade*, *e-mais-velho*, *n-mais-velho* e *recebe-ano*. Cada um com seus critérios de ativação e ações correspondentes. Pode-se observar que não existe uma sequencialidade dos comandos, como ocorre no PI. A sequência de execução é determinada segundo a execução de cada ciclo de inferência.

Cada ciclo de inferência de um SBR é composto por três fases (FRIEDMAN-HILL, 2003):

**- *Matching* ou casamento**: compara os fatos em relação às regras visando ativá-las. As regras ativadas são armazenadas, de forma desordenada, em um repositório chamado *Conjunto de Conflito*.

**- Seleção**: ordena as regras ativadas de acordo com alguma estratégia de resolução de conflito e/ou organização geral, podendo ser alguma estratégia baseada na prioridade das regras ou na recentidade dos fatos que ativaram as regras, formando um conjunto ordenado de regras chamado *Agenda*.

**- Execução**: seleciona a primeira regra da *Agenda* e executa a sua ação. Durante a execução da ação, a regra pode adicionar novos elementos na Base de Fatos ou ainda solicitar um serviço externo.

A fase de *Matching* é a que mais compromete a performance dos SBRs quanto ao desempenho, principalmente para aqueles SBRs que não tem um mecanismo de inferência suficientemente eficiente. Este fato ocorria principalmente nos primeiros SBRs, nos quais a perda de desempenho era ocasionada devido à avaliação redundante entre fatos e regras, sendo que muitos dos testes realizados em um ciclo de *matching* apresentam os mesmos resultados dos ciclos antecedentes (FRIEDMAN-HILL, 2003; BANASZEWSKI, 2009).

Algumas soluções para evitar a perda de desempenho nesta fase foram propostas, armazenando os estados já avaliados em ciclos anteriores, realizando as comparações somente das regras contra o estado dos elementos da Base de Fatos atualizados recentemente. Estas soluções foram implementadas em algoritmos

como o RETE (FORGY, 1982), o TREAT (MIRANKER, 1987), o LEAPS (MIRANKER et al., 1990) e o HAL (LEE e CHENG, 2002).

A título de exemplo do funcionamento de um sistema escrito em SBR, um pequeno código em CLIPS é apresentado na Figura 9. Neste programa existem duas regras que simulam a reação que um robô deve ter ao "ver" ou perceber a cor da luz indicada em um semáforo. Caso a cor da luz esteja vermelha ele deve parar, caso a cor da luz esteja verde ele deve andar.

Neste exemplo, as regras ficam carregadas em memória aguardando que um fato seja criado. A partir do momento da criação de um fato, a Máquina de Inferência executa a fase de casamento buscando as regras onde é possível combinar o fato inserido. Caso uma das regras seja satisfeita, ela é alocada na agenda e a ordem de sua execução é indicada. Após isso a ação é então executada.

```
1  (defrule sinal-vermelho
2       (cor-luz vermelha)
3  =>
4       (printout t "Pare!" crlf)
5  )
6
7  (defrule sinal-verde
8       (cor-luz verde)
9  =>
10      (printout t "Ande!" crlf)
11 )
```

Figura 9 – Exemplo de programa em CLIPS – Adaptado de GIARRATANO, 2015.

Embora seja possível utilizar a linguagem de programação PROLOG para desenvolvimento de SBRs, na maioria das vezes ele se demonstra inviável devido à sua sintaxe atípica e também a suas limitações de lógica de primeira ordem e pela falta de suporte a diferentes algoritmos de busca em sua forma pura (BANASZEWSKI, 2009).

Devido às limitações citadas no parágrafo anterior, ferramentas foram desenvolvidas para que seja possível criar os SBRs, estas ferramentas são chamadas de *shells*, e são compostas por uma linguagem, por um mecanismo de inferência eficiente e outras ferramentas úteis, como editor gráfico ou textual, gerenciador de arquivos e um depurador.

Entre as linguagens/*shells* mais conhecidas podemos citar CLIPS[10], Drools[11], ILOG Rules[12], Jess[13], OPS5[14] e RuleWorks[15]. Diferentemente da linguagem PROLOG, estas linguagens utilizam *forward chaining*. Na próxima subseção relata algumas ferramentas de SBR que foram encontradas a partir de uma Revisão Sistemática da Literatura sobre o tema (Krug, 2017).

Pertinente mencionar que por vezes SBRs são também "tradicionalmente" chamados de Sistemas Especialistas ou, em inglês, *Expert Systems*. Um Sistema Especialista trabalha em um domínio específico de aplicação, com conhecimento causal e factual substanciais (SIMÃO e STADZISZ, 2002). Usualmente, o conhecimento causal é obtido por especialistas humanos, enquanto o conhecimento factual pode vir de fontes diversas, como a observação humana ou automática, como outros sistemas ou sensores. (SIMÃO, 2005).

Um Sistema Especialista é um sistema interativo que auxilia o usuário com conhecimento de uma área específica. Os elementos básicos de um Sistema Especialista são a base de conhecimento (fatos e regras), que contém o conhecimento específico, e a máquina de inferência, que resolve o problema interpretando a base de conhecimento (MACKERLE, 1989). Entretanto, atualmente faz-se Sistema Especialista com outras técnicas além de SBR, como Multi-agentes, POO e redes neurais.

Outrossim, as características do SBR podem apresentar certa facilidade para o ensino de programação para novatos. Um dos indícios a respeito desta facilidade é apresentado através do experimento realizado por Krug (2016a) com o Paradigma Orientado a Notificações (PON). O experimento está apresentado na Seção Complementar B deste trabalho. Em tempo, o PON é um novo paradigma que tem inspirações em partes do SBR, sendo orientada a regras em termos de expressão em sua linguagem atual, chamada de LingPON (FERREIRA, 2015). De fato, a LingPON pode ser considerada um SBR em termos de expressão. Isto porque se

---

10 http://clipsrules.sourceforge.net/

11 https://www.drools.org/

12 http://www-01.ibm.com/software/info/ilog/

13 http://www.jessrules.com/

14 http://www.pcai.com/web/ai_info/pcai_ops.html

15 http://www.ruleworks.co.uk/uguide/rwug1.html

escreve regras e bases de fatos, ainda que seu sistema de inferência seja de todo inovador sendo orientado a notificações e não a buscas.

Quanto ao experimento em si já mencionado nele foi desenvolvido um algoritmo para solucionar a Torre de Hanói. Primeiramente foi feito uma versão do algoritmo em PP utilizando a linguagem C. Depois foi feita também uma outra versão do algoritmo, sendo equivalente mas orientada a regras (SBR-*like*) utilizando a LingPON. Neste experimento é observado e relatado maior facilidade em solucionar o problema da Torre de Hanói utilizando abordagem baseada em regras. Em tempo, essa implementação feita em LingPON seria replicável de maneira quase direta em CLIPS ou afins por ser orientada a regras.

A próxima subseção trata a respeito de linguagens e ferramentas para SBR.

### ***2.5.1.*** Linguagens e ferramentas para SBR[16]

Krug (2017) descreve em uma Revisão Sistemática da Literatura algumas ferramentas e linguagens para SBRs. Os trabalhos considerados para análise relatam a utilização de diferentes *shells* e linguagens para implementação de SBRs. Algumas destas *shells* são mais conhecidas (CLIPS, Drools e Jess) e outras propostas pelos próprios trabalhos (JAPLO, LAWRIS e Netlog). A seguir será relatado um pouco de cada uma delas.

Conforme Giarratano (2015), CLIPS é um acrônimo para *C Language Integrated Production System* e foi desenvolvido em 1986 pela NASA – USA. CLIPS foi projetada para facilitar o desenvolvimento de software através da modelagem do modelo de conhecimento humano e experiência (NASA, Lyndon B. Johnson Space Center, 1991).

Ainda segundo Giarratano (2015), CLIPS é considerada uma ferramenta para sistemas especialistas pois é um ambiente completo para desenvolvimento destes sistemas, incluindo editor integrado e ferramentas para depuração. Ela contém todos os elementos básicos de um sistema especialista: base de fatos, base de conhecimento e máquina de inferência.

16 Conteúdo reaproveitado e melhorado a partir de Revisão Sistemática da Literatura escrita sobre Sistemas Baseados em Regras. Estudo Individual, PPGCA/DAINF/UTFPR. Adaptado de Krug (2017).

Drools, conforme informações de sua documentação[17], é uma *rule engine* que é baseada em regras e é utilizada para produzir sistemas especialistas, mais corretamente classificado como Sistema de Produção. Drools é um BRMS (*Business Rule Management Systems*), baseado em Java e utiliza o algoritmo Rete.

Conforme Vlas e Robinson (2011) JAPE (*Java Annotation Pattern Engine*) é uma *engine* baseada em regras que suporta a linguagem Java e expressões regulares. JAPE faz parte de uma solução de processamento de texto chamada GATE (*General Architecture for Text Engineering*).

JAPLO, segundo Espák (2006), é uma extensão da linguagem Java que foi criada para permitir a criação de regras, estilo Paradigma Lógico, por programadores familiarizados com Java. JAPLO permite a utilização de regras semelhantes ao Prolog com sintaxe semelhante à da linguagem Java. O nome é uma junção de Java com Prolog.

Conforme Friedman-Hill (2003), Jess é uma *shell*, desenvolvida em Java, baseada em regras desenvolvida pela Sandia National Laboratories no final da década de 1990. Embora Jess apresente grandes semelhanças com CLIPS, elas foram desenvolvidas por grupos diferentes de pessoas. Jess assemelha-se em alguns pontos com CLIPS, porém é baseada em Java e permite integração com esta linguagem.

Segundo Arakliotis, Nikolos e Kalligeros (2016), LAWRIS (*Learning-Arduino-With-Rules Introductory System*) é uma plataforma de aprendizado baseada na Web que permite desenvolvimento de programas para Arduino[18] utilizando regras. Esta plataforma foi desenvolvida para permitir que estudantes do ensino fundamental pudessem ter contato com desenvolvimento de Arduino de uma maneira simples, sendo o desenvolvimento baseado em regras e utilizando um editor gráfico com blocos estilo *drag and drop*.

Grumbach e Wang (2010) propuseram a linguagem Netlog, uma linguagem declarativa, baseada em regras, para ser utilizada com aplicações distribuídas, como protocolos de comunicação e aplicações *peer-to-peer*.

Macioł et al. (2015) apresentam em seu trabalho a *shell* REBIT (*Business and Technological Rules Management System*) desenvolvido pela *Faculty of Management* da *AGH University of Science and Technology* na Cracóvia. Conforme

17 https://docs.jboss.org/drools/release/5.3.0.Final/drools-expert-docs/html/

18 https://www.arduino.cc/

os autores, a *shell* REBIT é composta por uma linguagem e um algoritmo de otimização próprios e tem em sua essência uma plataforma para a troca de informações de forma não distorcida entre o sistema baseado em regras e modelos de simulação.

*Structured Query Language* (SQL) é uma linguagem declarativa utilizada diretamente em bancos de dados. Algumas aplicações utilizam SQL baseado em regras, conforme relatam Abdullah, Sawar e Ahmed (2009) e Sawar, Abdullah e Ahmed (2010).

Conforme Yamashita, Ohta e Takami (2010), STAR (*SofTware Architecture using Rule-based language*) foi proposta com o intuito de desenvolver softwares em menor tempo. STAR utiliza uma linguagem baseada em regras chamada ESTR (*Enhanced State Transition Rule*), que apresenta uma especificação simples e um baixo volume de programação.

Em suma, é possível verificar diversas linguagens e ferramentas para SBR, algumas destas ferramentas são consolidadas e de ampla utilização, como CLIPS, Drools e Jess, enquanto outras foram propostas pelos próprios trabalhos que as descrevem sendo (até então) de uso não amplo.

Outrossim, a próxima subseção relata algumas das aplicações com SBR encontradas por Krug (2017).

### ***2.5.2.*** Aplicações com SBR[19]

Soluções desenvolvidas com SBRs estão sendo utilizadas em diversas áreas, sendo elas, seguro, finanças, saúde, biologia, jogos para computadores, viagens e engenharia de software, conforme descreve Zacharias (2008). Neste sentido, implementações com SBRs vem sendo relatadas em artigos científicos, sendo que algumas delas serão relatadas a seguir.

Panescu, Pascal e Olaeru (2015) relatam a utilização de SBR para controlar uma aplicação multi robôs em processos de manufatura, na qual os robôs trabalham com áreas de armazenamento individual, porém utilizando uma área de montagem em comum. Devido a isso, os robôs devem ser sincronizados para as tarefas de montagem e empilhamento. Comparando a solução construída com SBR e uma

---

19 Conteúdo reaproveitado e melhorado a partir de Revisão Sistemática da Literatura escrita sobre Sistemas Baseados em Regras. Estudo Individual, PPGCA/DAINF/UTFPR. Adaptado de Krug (2017).

solução sequencial (implementada com o Paradigma Imperativo), na qual os robôs não trabalham de forma coordenada e paralela, é possível observar um ganho de performance na solução implementada com SBR.

Uma solução utilizando SBR para controle de robôs também é apresentada por Shimbuichi, Matsuoka e Takami (2010), na qual relatam uma aplicação para controle de mais de um robô, de modelos distintos, coordenados para solucionar uma mesma ação.

Por sua vez, Shimokura, Nakanishi e Ohta (2008) propõe um sistema desenvolvido com os preceitos de SBR para promover uma convivência simbiótica entre humanos e robôs, sendo que através da utilização de regras é possível facilitar a descrição das tarefas que devem ser executadas pelos robôs de maneira simples e segura na opinião dos autores.

O controle de robôs pela rede e também uma rede doméstica através da utilização de SBR é apresentada por Shimokura, Nakanishi e Ohta (2007).

Vlas e Robinson (2011) descrevem a utilização de SBR para descobrir requisitos de software *open-source*, os quais normalmente são descritos em meios não tradicionais de documentação de requisitos, como fóruns, chats e e-mails. Estes requisitos estão escritos em linguagem natural e a utilização de SBR facilita a forma como os requisitos são encontrados e classificados.

A utilização de SBRs em software voltado a medicina é comum, principalmente quanto ao auxílio no diagnóstico e melhor tratamento de doenças, sendo um dos pioneiros o MYCIN[20], o qual foi criado nos primeiros anos da década de 1970.

Pesquisas recentes também trazem novas implementações de SBRs voltados para medicina. Wang et al. (2010) descreve a utilização de SBR para predição e diferenciação de destino para células-tronco mesenquimais, sendo que a base de conhecimento deste sistema para composição das regras foi retirada de estudos anteriores.

No mesmo campo da medicina, agora voltado ao auxílio de diagnóstico, Alharbi, Berri e El-Masri (2015) propuseram um sistema de apoio a decisão clínica para diagnóstico de diabetes utilizando SBR. Este sistema considera informações do paciente, sintomas, sinais, fatores de risco e testes de laboratório para sugerir um tratamento de acordo com o tipo de diabetes do paciente.

20 https://en.wikipedia.org/wiki/Mycin

Ainda na área médica, mas agora com outra função, Abdullah, Sawar e Ahmed (2009) e Sawar, Abdullah e Ahmed (2010) propuseram a utilização de SBR para processar os documentos necessários para reembolso de procedimentos médicos.

Também é possível verificar a utilização de SBR para o ensino de programação. Cohen, Ritter e Haynes (2009) utilizaram SBR para ensino de Inteligência Artificial e Arakliotis, Nikolos e Kalligeros (2016) para ensino de programação para estudantes do ensino fundamental, utilizando Arduino e blocos com programação baseada em regras.

Para expressar a amplitude de soluções que podem ser criadas utilizando SBR, é possível relatar o trabalho escrito por Yamashita, Ohta e Takami (2010), o qual apresenta a utilização de SBR para criar Web Services. Park, Lee e Jang (2015) apresentam a utilização de SBR para uma implementação de *Web of Things* (WoT). Ainda, El-Khayat e Mabrouk (2014) apresentam a utilização de SBRs para seleção/atribuição de candidatos a cursos de ensino superior de acordo com suas habilidades. Finalizando, Macioł et al. (2015) apresentam a utilização de SBR para tomada de decisões em processos de manufatura.

## **2.6.** ENSINO DE PROGRAMAÇÃO[21]

A programação de computadores faz parte dos currículos de cursos da área de TI, seja em nível superior ou técnico profissionalizante. Durante o decorrer dos cursos, ela pode ser abordada em diversos níveis, do mais básico ao mais aprofundado. Neste âmbito uma das disciplinas introdutórias da grande parte dos currículos destes cursos é uma disciplina chamada Introdução a Programação, dentre outros nomes similares ou afins.

Nesta disciplina introdutória é onde grande parte dos estudantes tem seu primeiro contato com a programação de computadores. Assim, tal disciplina serve como base para conceitos subsequente mais aprofundados de programação. Portanto, tal disciplina é de suma importância para os futuros profissionais no âmbito da Engenharia de Software.

---

21 Conteúdo reaproveitado e melhorado a partir do documento utilizado para Seminário I, 1. PPGCA/DAINF/UTFPR. Adaptado de Krug (2016b).

Isto dito, existem diversas metodologias que podem ser utilizadas para o ensino de programação, como fluxogramas, algoritmos em pseudocódigo e mesmo a utilização direta de uma linguagem de programação.

A utilização de metodologias e ferramentas para auxiliar o ensino de programação sempre fizeram parte de estudos e experimentos. Isto se deve inclusive às suas múltiplas variáveis como paradigmas de programação, linguagens de programação e forma de aprendizado. Assim, possivelmente este assunto deve permanecer objeto de estudo por muitos anos.

A opção por uma metodologia ou ferramenta varia de professor para professor, ou até mesmo de turma para turma do mesmo professor, sempre sendo necessária uma avaliação a respeito da escolha.

Assim como a escolha das metodologias não é a mesma para todos, a escolha de uma primeira linguagem de programação e por consequência o seu paradigma também não é consenso entre os professores e mesmo pesquisadores no assunto.

A escolha do primeiro paradigma de programação e da primeira linguagem de programação, tem grande importância para professores e principalmente para os alunos. A título de paralelo, assim como a primeira linguagem falada apresenta grande influência na maneira de pensar, a linguagem de programação também influencia o porvir do futuro programador. Esta escolha tem de fato grande impacto no estilo de programação, nas técnicas de codificação e também na qualidade do código, além de ser crucial para adquirir conceitos básicos em ciência da computação e na futura aprendizagem de outros paradigmas e linguagens de programação (VUJOŠEVIĆ-JANIČIĆ e TOŠIĆ, 2008).

Embora a preferência do professor por um determinado paradigma de programação ou uma determinada linguagem de programação tenha grande influência nesta escolha, o mercado de trabalho também influencia a escolha da primeira linguagem e do primeiro paradigma. Neste sentido, até a década de 90 linguagens como Pascal e C eram muito utilizadas para o ensino de programação. Entretanto, elas foram perdendo espaço para linguagens como C++ e Java, devido à tendência da indústria na utilização de linguagens do POO (JANKE, BRUNE e WAGNER, 2015).

Em conclusão, a influência da indústria faz com que naturalmente o foco no ensino seja utilizando o PI (PP e POO), deixando o PD (PF e PL) periférico. Para

apresentar diferentes opções de escolha é necessário que comparativos entre diferentes linguagens e paradigmas sejam realizados, evidenciando as vantagens, desvantagens e diferenças. Alguns dos comparativos encontrados serão descritos na próxima seção.

## **2.7.** COMPARATIVO ENTRE PARADIGMAS[22]

Comparativos entre paradigmas e linguagens de programação devem servir como norte para a escolha do paradigma que será ensinado, ou ainda, que será aprendido.

Dentre as formas de comparativo encontradas nos trabalhos realizados anteriormente, citados na introdução desta seção, foi possível encontrar classificações e comparativos no tocante às características dos paradigmas quanto à aprendizagem, quanto ao ensino de programação e comparativo utilizando modelo de compreensão de programas.

Estes comparativos estão relatados nas subseções a seguir, finalizando com um resumo a respeito dos estudos de comparativos citados.

### 2.7.1. Características dos paradigmas quanto à aprendizagem

Uma das maneiras de diferenciação entre paradigmas é através das características de aprendizagem.

A aprendizagem dos paradigmas que pertencem ao grupo do PI tendem a ser mais sistemáticas, pois a forma como o paradigma se comporta é de forma sequencial, embora para o POO exista a complexidade devido à utilização de classes e objetos. Temas como variáveis, tipos de dados, estruturas lógico-causais, estruturas de repetição são itens primordiais para aprendizagem dos paradigmas do PI.

Conforme relatam Janke, Brune e Wagner (2015), a abordagem clássica para ensino de POO começa com princípios básicos do PI, como variáveis, tipos de dados, estruturas de controles, funções ou métodos e algoritmos. Depois é realizado a introdução de classes e objetos. Esta forma de ensino reflete a evolução histórica das linguagens de programação e, por consequência, é a ordem em que os programadores acabam aprendendo por si mesmos.

---

22 Conteúdo reaproveitado e melhorado a partir do documento utilizado para Seminário I, 1. PPGCA/DAINF/UTFPR. Adaptado de Krug (2016b).

Em contrapartida, nos últimos anos algumas outras abordagens para o ensino de POO estão aparecendo. Elas sugerem uma abordagem invertida, iniciando pelos conceitos de classes, objetos, interfaces e funções, as vezes em conjunto com os conceitos básicos do PI e as vezes mesmo antes destes conceitos (JANKE, BRUNE e WAGNER, 2015).

Quanto as dificuldades de programação em si e, por consequência, uma característica que afeta aprendizagem, Banaszewski (2009) relata que geralmente para os programas criados com os conceitos do PI, o código que envolve a lógica da aplicação se encontra disperso entre os comandos e as expressões de controle da linguagem. Isto torna difícil a leitura e entendimento do código, além de desviar a atenção do que realmente importa, a lógica da solução para o problema a ser resolvido. Ainda as linguagens do PI apresentam sintaxes pouco intuitivas e não raro complexas.

Especificamente quanto a programas que utilizam os conceitos do POO, Banaszewski (2009) traz que, devido aos relacionamentos entre os objetos, fica difícil compreender uma funcionalidade analisando apenas uma classe, sendo geralmente necessário considerar as demais classes relacionadas. Este tipo de relacionamento impacta diretamente a manutenção dos programas.

Por sua vez Choppella et al. (2012) traz que a computação tem sua origem na matemática e na lógica, sendo que o ensino de programação pode utilizar o conhecimento prévio dos alunos em matemática para ter mais sucesso. Este conhecimento pode ser útil se for ensinado o PF inicialmente, pois fica mais fácil de compreender os conceitos de programação tendo uma analogia com conhecimento anterior. Pensando em PF, pode-se associar um programa com fórmulas que podem ser simplificadas utilizando substituição. A sequência de etapas de simplificações, que são as aplicações de regras de um algoritmo, constitui uma execução do programa em uma dada entrada. Quando nenhuma regra mais pode ser aplicada, a execução do programa é encerrada, trazendo a resposta final. Esta sequência de passos de simplificação é a derivação da saída de acordo com a entrada.

Ainda, segundo Choppella et al. (2012), programar utilizando o PF não implica em trazer preocupações com memória, armazenamento e outros problemas de hardware, sendo que o estudante pode focar em ter a lógica de forma correta, sem se preocupar com outros problemas.

Huck (2007) afirma que utilizando a linguagem Erlang, uma linguagem pertencente ao PF, para o ensino de programação para novatos, evita que se tenha uma sobrecarga sintática. Assim, os novatos podem concentrar nos reais desafios de programar, pois eles não precisam lidar com conceitos diferentes de itens similares, por exemplo, diferentes tipos de laços de repetição.

A respeito do PL, Kumar (2002) descreve a utilização de *templates* para ensinar PL (especificamente PROLOG) para praticantes do PI. Segundo Kumar (2002), ensinar PL para quem já conhece PI pode causar bastante dificuldade.

As linguagens dos paradigmas PF e PL, segundo Banaszewski (2009), previnem o programador da interação direta com comandos das linguagens imperativas, encapsulando os comandos imperativos e assim evitando seu manuseio direto. De forma semelhante, os *shells* do SBR permitem a composição da lógica da aplicação por meio de regras em ferramentas amigáveis, evitando contato com as particularidades das linguagens imperativas.

Por fim, embora as linguagens do PD facilitem a programação, elas não apresentam a mesma flexibilidade das linguagens imperativas, o que conforme Banaszewski (2009) pode limitar a criatividade do programador e também o acesso aos componentes do hardware, ou ainda impossibilitar a construção de códigos mais eficientes.

A respeito dos paradigmas declarativos, o aprendizado passa por ater mais a atenção na forma de resolver o problema e não como ele será feito em meios computacionais. Especificamente no PF é possível atrelar conhecimentos da matemática junto ao aprendizado deste paradigma.

Na próxima subseção será tratado com mais detalhes a respeito do comparativo de paradigmas quanto ao ensino da programação.

### 2.7.2. Comparativo quanto ao ensino de programação

Pensando na escolha da primeira linguagem de programação, Mannila e Raadt (2006) apresentaram um conjunto de critérios para escolher a melhor linguagem de programação para cursos de introdução de programação. No artigo relatam que grande parte dos trabalhos que comparam linguagens carregam certas opiniões levadas pelo emocional. Como resposta a isto listaram critérios que esperam que permita uma escolha objetiva. Os critérios foram sugeridos por criadores de linguagens consideradas linguagens para ensino: Seymour Papert

(criador do LOGO), Niklaus Wirth (criador do Pascal), Guido van Rossum (criador do Python), e Bertrand Meyer (criador do Eiffel).

Os critérios escolhidos por Mannila e Raadt (2006) são: 1) a linguagem é adequada para o ensino; 2) a linguagem pode ser utilizada para aplicar analogias físicas; 3) a linguagem oferece um *framework* geral; 4) a linguagem promove uma nova forma de ensino de software; 5) a linguagem é interativa e facilita rápido desenvolvimento de código; 6) a linguagem promove a escrita correta de programas; 7) a linguagem permite resolver problemas de forma modularizada; 8) a linguagem provê um ambiente de desenvolvimento sem costuras; 9) a linguagem tem uma comunidade de suporte; 10) a linguagem é *open source*; 11) a linguagem é multiplataforma; 12) a linguagem é gratuita e disponível facilmente; 13) a linguagem dispõe de um bom material para o ensino; 14) a linguagem não é utilizada apenas na educação; 15) a linguagem é extensível; 16) a linguagem é confiável e eficiente; 17) a linguagem não é um fenômeno com vida curta.

Vujošević-Janičić e Tošić (2008) fizeram um apanhado da literatura comparando os paradigmas dominantes quanto à utilização no ensino de programação. A respeito do PP, os autores apontam como um paradigma de fácil entendimento e, em alguns casos, fácil de converter algoritmos intuitivos em código, muito embora existam alguns conceitos de difícil entendimento, como atribuição, sequência, interação e recursão.

Conforme Vujošević-Janičić e Tošić (2008), realizar as tarefas em sequência, como realizado no PP, é algo que faz parte do cotidiano. Esta é uma das grandes vantagens do paradigma e muitos professores esperam que os estudantes entendam facilmente os conceitos de sequência e interação. Porém, nem sempre é o que ocorre, existindo relatos de problemas no entendimento destes conceitos básicos, fazendo com que os professores devam prestar muita atenção se houve realmente o entendimento dos conceitos desde o início.

Para o ensino do PP, Vujošević-Janičić e Tošić (2008) relatam a utilização de duas linguagens, Pascal e C. Pascal foi a principal escolha para ensino do paradigma até a década de 90, mas foi perdendo espaço para a linguagem C, devido ao propósito geral da linguagem e ao fato de ser melhor vista pela indústria de TI.

Vujošević-Janičić e Tošić (2008) apontam duas grandes razões para utilizar o POO como primeiro paradigma de programação. A primeira é a grande popularidade

e importância deste paradigma e a segunda é a aparente dificuldade de mudar de paradigma. Assim, se o ensino for realizado utilizando o PP, ficará difícil utilizar o POO. Entretanto, alguns pontos negativos são apontados, indicando também que é difícil o aprendizado do POO por novatos.

Uma das maiores dificuldades é que a programação em POO requer mais análise e projeto antes da codificação, em comparação ao PI. Além disso, se o ensino iniciar com o POO, os estudantes terão sua formação sem habilidades básicas de programação, requeridas para uma implementação de baixo nível. O ensino de POO em sua maioria é realizado utilizando as linguagens C++ e Java, sendo que a utilização de Java apresenta grande adesão (VUJOŠEVIĆ-JANIČIĆ e TOŠIĆ, 2008).

Para o PF, por sua vez, Vujošević-Janičić e Tošić (2008) apresentam que existe muita controvérsia a respeito da utilização deste paradigma como o primeiro a ser ensinado. Alguns professores entendem que é o melhor paradigma para ser ensinado como primeiro, caso o foco seja em conceitos gerais e não na programação funcional em si. Esta opinião deve-se ao fato de que com o PF é possível focar nos problemas, sem tanta preocupação com o hardware. Por outro lado, alguns professores defendem que o ensino do PF é complicado devido à necessidade de entendimento pelos alunos do conceito do cálculo Lambda e do mecanismo de substituição de argumentos. Finalmente ressaltam que o PF é utilizado para ensino multiparadigmas, pois grande parte das linguagens funcionais apresentam suporte a outros paradigmas, como o PP e o POO. Uma das linguagens mais utilizadas é o Scheme.

Para o PL, Vujošević-Janičić e Tošić (2008) associam diretamente o ensino deste paradigma com a linguagem PROLOG e não trazem muitas informações a respeito. Eles apenas indicam que uma grande parte da dificuldade de aprendizagem do PROLOG está relacionado à complexidade da linguagem e a dificuldade dos alunos em entender os princípios matemáticos no qual o PROLOG é baseado.

A próxima subseção relata uma comparação entre os paradigmas utilizando o modelo de compreensão de programas.

2.7.3. Modelo de compreensão de programas

Os trabalhos escritos por Wiedenbeck et al. (1999), Wiedenbeck e Ramalingam (1999) e Khazaei e Jackson (2002) descrevem uma comparação entre os paradigmas utilizando como base o modelo de compreensão de programas de computadores descrito por Pennington (1987). Desta forma esta subseção detalha o que é este modelo de compreensão de programas de computadores.

O modelo de compreensão de programas de computadores, desenvolvido por Pennington (PENNINGTON, 1987a; PENNINGTON, 1987b), é baseado no modelo de compreensão de texto de Dijk e Kintsch (1983), no qual o leitor forma uma representação mental de um texto em dois níveis diferentes. Um nível é na representação do texto-base, que é a forma do texto, as proposições do texto e as relações entre estas proposições, além da organização. O outro nível é o modelo de situação, ou modelo mental, que é a representação do leitor a respeito da situação do mundo real referida pelo texto. Este segundo nível é derivado da inferência a partir do texto e depende diretamente do conhecimento do leitor.

No modelo de Pennington (1987), também se tem dois níveis, o do programa e o do domínio, que se baseiam respectivamente no nível do texto-base e no modelo de situação.

O nível do programa é formado por dois tipos de informação contidos no programa: 1) operações elementares – pequenas unidades do programa que correspondem a um comando (e.g. atribuição de um valor para uma variável); e 2) fluxo de controle – define o fluxo de execução de programa (e.g. sequência dos comandos, laços/*loops* e chamadas de procedimentos). O fluxo de controle representa a maneira como as operações elementares são operacionalizadas durante a execução do programa. Ambos os tipos de informação, operações elementares e fluxo de controle, podem ser compreendidos mesmo que o programador não tenha conhecimento a respeito do propósito do programa (WIEDENBECK et al., 1999).

Por sua vez, o nível de domínio é formado pelas informações de: 1) fluxo de dados – a transformação dos dados durante a execução do programa, tendo em conta que o objetivo principal da maioria dos programas é transformar dados de entrada e gerar uma saída; e 2) função do programa – expressa o que o programa faz em termos de entidades, relacionamentos e ações, sendo informações que

geralmente não são expressas no texto do programa, mas que podem ser obtidas de acordo com o programa e o conhecimento do mundo real (WIEDENBECK et al., 1999).

Em estudos realizados com base no modelo de Pennington (1987), foi possível verificar que novatos em programação compreendem melhor o nível do programa e, conforme cresce a experiência, aumenta também a compreensão do nível do domínio (WIEDENBECK et al., 1999).

Segundo Wiedenbeck et al. (1999) a compreensão é muito importante para programadores novatos, porque grande parte de seu aprendizado depende da compreensão de programas exemplos. Além disso, eles devem compreender um programa escrito para que seja possível depurar e realizar modificações neste programa.

Dada esta breve explanação a respeito do modelo de compreensão de programas de computadores escrito por Pennington (1987), a próxima subseção trata sobre estudos comparativos a respeito do aprendizado de programação à luz de paradigmas de programação.

#### *2.7.4.* Estudos comparativos

Considerando o exposto nas subseções anteriores, esta subseção resume os estudos comparativos encontrados no tocante ao aprendizado de programação à luz de paradigmas de programação.

Nos artigos escritos por Robins, Rountree e Rountree (2003) e Vujošević-Janičić e Tošić (2008) são apresentados alguns conceitos a respeito dos paradigmas de programação e é realizado um breve apanhado a respeito dos detalhes de alguns paradigmas, não chegando ao nível de uma comparação entre os paradigmas de programação quanto ao aprendizado.

O artigo escrito por Wiedenbeck et al. (1999) compara o PP e o POO quanto à compreensão dos programas por novatos em programação, utilizando o modelo de Pennington (1987). O teste foi realizado utilizando programas impressos e uma série de questões a respeito dos programas para mapear a compreensão dos estudantes a respeito de cada paradigma. Os estudantes foram submetidos a duas etapas, sendo uma com um programa mais curto e a outra com um programa mais extenso. Para o PP foi utilizada a linguagem Pascal, enquanto para o POO foi utilizada a linguagem C++.

Na primeira etapa não foi possível observar diferenças estatisticamente relevantes entre o PP e o POO, porém foi possível notar um melhor desempenho dos praticantes de POO quanto à compreensão de função do programa, sugerindo que o grupo de POO tem uma melhor compreensão do modelo de domínio. Na segunda etapa, com programa mais extenso, o grupo de estudantes do PP teve um desempenho superior ao grupo de POO, em torno de 15% superior no geral. Também quanto à compreensão de função do programa e fluxo de dados, o desempenho do grupo de POO foi muito baixo.

Através do estudo realizado por Wiedenbeck et al. (1999), foi concluído que uma das explicações para o pior desempenho do grupo de POO é que o paradigma é mais complexo para novatos em programação devido ao fato de não ser hierarquizado, tornando difícil o mapeamento mental das funcionalidades e do controle de fluxo. Além disso, os autores mencionam que talvez o ensino de POO como primeiro paradigma não seja o ideal, devido à complexidade com abstração que demanda muito esforço para compreensão. Uma possibilidade é incluir o POO em um estágio mais avançado de ensino.

Em outro trabalho escrito por Wiedenbeck e Ramalingam (1999), o foco está no objetivo de mapear que tipo de informações os programadores novatos extraem de programas pequenos e ligar esta informação à representação mental formada durante a compreensão de programas. Embora o foco do trabalho de Wiedenbeck e Ramalingam (1999) não seja comparar os paradigmas PP e POO propriamente dito, busca verificar se a representação mental de ambos os paradigmas é diferente. Também foi utilizado o modelo escrito por Pennington (1987), adicionando uma informação de conhecimento do estado, que compreende o estado de todos os aspectos do programa no momento em que uma determinada ação ocorre no programa. Para o PP foi utilizada a linguagem C, enquanto para o POO foi utilizada a linguagem C++.

A hipótese do trabalho escrito por Wiedenbeck e Ramalingam (1999) é que para programas do POO será mais fácil extrair informações relacionadas ao domínio do programa do que de programas do PP, devido à estrutura do programa, como a utilização de classes, objetos e encapsulamento, em POO. De acordo com a hipótese das autoras, a notação das classes destaca informações relevantes sobre os objetos de domínio do problema do programa e funções associadas a estes objetos.

Analisando a quantidade total de erros nas respostas do questionário aplicado posteriormente à análise do programa, não houve diferença entre os grupos de PP e POO. Entretanto, quando se isola as categorias de informação, o estudo revela que para compreensão do programa, os novatos extraem informações diferentes dos programas, implicando em diferentes representações mentais entre os paradigmas. Para o grupo de PP a quantidade de erros foi menor para o nível do programa, operações elementares e fluxo de controle. Para o grupo de POO, por sua vez, a quantidade de erros foi menor para o nível de domínio, levando em consideração as informações relativas à função do programa. Ainda no nível do domínio, para as informações de fluxo de dados não houve diferença do número de erros, o mesmo aconteceu para as informações de estado. Estas observações sugerem que os dois estilos diferentes (PP e POO) destacam certas informações ao custo de outras informações (WIEDENBECK e RAMALINGAM, 1999).

A pesquisa de Wiedenbeck e Ramalingam (1999) conclui que programadores novatos que utilizam o POO formam uma forte compreensão do nível de domínio, principalmente das funções do programa, diferente dos programadores novatos que utilizam o PP, sugerindo que o POO facilita o entendimento da função do programa para novatos que trabalham com programas pequenos. Porém para o sucesso do entendimento de um programa isso não é suficiente, dado que bons entendedores de programas devem ter bem desenvolvido tanto o nível do programa quanto o nível de domínio e ter a capacidade de correlacioná-los.

Aproveitando o trabalho prévio escrito por Wiedenbeck e Ramalingam (1999), no qual é estudada a compreensão de novatos para pequenos programas em PP e POO, Khazaei e Jackson (2002) realizaram a comparação entre a compreensão de pequenos programas entre o POO e o Desenvolvimento Dirigido a Eventos. Para o POO foi utilizada a linguagem Java, enquanto para o Desenvolvimento Dirigido a Eventos foi utilizada a linguagem Visual Basic.

O Desenvolvimento Dirigido a Eventos ou *Event-Driven Development*, é considerado por alguns como um paradigma de programação, vinculado ao POO e por outros apenas um estilo de programação. Nele o fluxo do programa é determinado por eventos (i.e. ações de usuários, dados coletados de sensores e mensagens de outros programas). A programação neste estilo pode ser realizada em diversas linguagens de programação, principalmente nas que propiciam maior nível de abstração.

Através do resultado das respostas obtidas após o teste descrito por Khazaei e Jackson (2002), foi possível verificar que a representação mental para POO e Desenvolvimento Dirigido a Eventos são semelhantes, não apresentando diferença estatisticamente significante, apresentando apenas uma diferença quanto ao fluxo de dados, demonstrando melhores resultados para Desenvolvimento Dirigido a Eventos do que para POO. Alguns pontos para POO são semelhantes ao teste realizado por Wiedenbeck e Ramalingam (1999), como as informações do fluxo de dados e das funções do programa foram boas em ambos os estudos. Também foi possível verificar a semelhança nos resultados quanto as operações elementares, com desempenho ruim em ambos os estudos. Quanto ao fluxo de controle, por sua vez, o resultado obtido no estudo realizado por Wiedenbeck e Ramalingam (1999) foi ruim, e o obtido por Khazaei e Jackson (2002) foi bom. Finalmente, foi possível concluir que a forma de compreensão entre Desenvolvimento Dirigido a Eventos e POO são semelhantes e, por sua vez, mais semelhantes entre si do que comparando com PP.

Quanto ao comparativo de linguagens de programação especificamente, Mannila e Raadt (2006) estabeleceram alguns critérios, citados na seção 2.7.2, e aplicaram estes critérios em algumas linguagens, escolhidas dentre as listadas em um censo realizado por de Raadt, Watson e Toleman (2003). As linguagens escolhidas para comparação foram: C, C++, Eiffel, Haskell, Java, JavaScript, Logo, Pascal, Python, Scheme e Visual Basic.

Aplicando os critérios especificados por Mannila e Raadt (2006), as linguagens Eiffel e Python foram as que tiveram a melhor pontuação, seguidas de perto pela linguagem Java. Embora os autores considerem que o trabalho realizado foi mais objetivo do que outros encontrados por eles, deixam aberto a possibilidade de ampliar ou modificar os critérios, inclusive incluir outras linguagens de programação que foram deixadas de lado nesta comparação.

Especificamente quanto à comparação entre paradigmas, os estudos apresentados relataram comparações entre PP e POO e entre POO e Desenvolvimento Dirigido a Eventos, ambos os estudos observaram os níveis de informações formados por praticantes de cada paradigma. Como conclusão os estudos observaram que os níveis de compreensão são distintos em alguns pontos, mas no geral a compreensão dos programas assemelha-se entre os paradigmas. No estudo escrito por Wiedenbeck et al. (1999) sugere-se que PP é mais recomendado

para ensino como primeiro paradigma de programação e o POO pode ser ensinado em um nível mais avançado.

*2.7.5.* Reflexão sobre Estudos Comparativos

Alguns dos trabalhos citados atém-se à descrição e diferenciação dos conceitos relativos aos paradigmas, como os trabalhos escritos por van Roy (2009), Robins, Rountree e Rountree (2003) e Vujošević-Janičić e Tošić (2008), outros realizam uma comparação entre a compreensão do programa, baseado no método escrito por Pennington (1987), baseado no modelo de compreensão de texto de Dijk e Kintsch (1983), como os trabalhos escritos por Wiedenbeck et al. (1999), Wiedenbeck e Ramalingam (1999) e Khazaei e Jackson (2002). Ainda existe o trabalho escrito por Mannila e Raadt (2006) que realiza o comparativo entre linguagens de programação.

É possível encontrar também na literatura trabalhos a respeito de soluções desenvolvidas especificamente em um paradigma, destacando as vantagens e desvantagens do uso do paradigma escolhido.

Comparações diretas entre paradigmas são raras, ainda mais utilizando coleta de dados a respeito do desenvolvimento de soluções em distintos paradigmas.

Desta forma, visando prover dados comparativos entre distintos paradigmas, o presente trabalho propõe um método, ainda em fase de construção, que busca utilizar informações coletadas durante o desenvolvimento de programas de computador, aplicando-as na classificação dos paradigmas quanto ao processo de aprendizagem, buscando a melhoria da qualidade de software.

# 3. MÉTODO PARA COMPARAÇÃO ENTRE PARADIGMAS

Neste capítulo, está descrito o método de comparação entre paradigmas de programação quanto ao aprendizado proposto por este trabalho, o qual contém as fases de coleta de dados, classificação dos artefatos, processamento das atividades geradas, análise e comparação.

## 3.1. MÉTODO PROPOSTO

O método que está sendo proposto consiste em analisar o programa escrito, classificando a solução e categorizando os erros cometidos no desenvolvimento do programa, além de utilizar informações capturadas durante o desenvolvimento considerando-as para gerar dados que possibilitem a comparação entre distintos paradigmas quanto à aprendizagem de programação.

A captura de informações em plano de fundo, que possam auxiliar no comparativo entre paradigmas, é um dos diferenciais do método proposto pois, além do artefato final, é possível observar dados coletados durante o desenvolvimento do mesmo.

Durante a análise de um programa, a simples verificação do artefato gerado pode ser utilizada para classificar se o mesmo atende ou não à solução de um determinado problema, porém nem sempre os erros no artefato são levados em consideração para classificação do aprendizado.

Partindo do pressuposto de que o conhecimento dos erros cometidos durante o desenvolvimento de um programa possa auxiliar no mapeamento das dificuldades enfrentadas por quem o desenvolveu, o método proposto utiliza a categorização dos erros encontrados nos programas, servindo também como um dado para a comparação entre paradigmas de programação.

Esta informação, além de servir como dado comparativo, também auxilia em mapeamentos necessários para correção de possíveis falhas no processo do ensino de programação. Visto que se a categoria de erros cometidos por um grupo de estudantes segue um padrão, é necessária uma intervenção específica para solucionar o entendimento da causa dos erros, auxiliando em correções no processo de ensino aprendizagem.

Por sua vez, dados coletados em plano de fundo durante o desenvolvimento do artefato podem ser utilizados para gerar métricas como: tempo de

desenvolvimento; tempo ocioso durante o desenvolvimento; velocidade de digitação; e também a exclusão de caracteres.

Os dados relacionados com medição de tempo podem indicar uma eventual facilidade de programação, ou ainda, medir o desempenho de produtividade em um paradigma específico.

Desta forma o método proposto divide-se em 4 fases: coleta de dados; classificação dos artefatos; processamento das atividades geradas; e análise e comparação.

As próximas subseções descrevem cada uma das etapas.

### 3.1.1. Coleta de Dados

No método proposto, são considerados para comparação entre os paradigmas o artefato gerado e também as atividades geradas durante o desenvolvimento do programa.

O artefato completo, i.e., o programa, é levado em consideração para validar se o mesmo atende ao que foi solicitado e também para o mapeamento de erros cometidos por quem escreveu o programa.

O programa pode ser escrito em qualquer editor de texto ou interface de desenvolvimento e é analisado em sua totalidade.

As atividades geradas pelos alunos durante o desenvolvimento do programa também são consideradas neste método. Atividades como troca de janela, movimentos do mouse e teclas pressionadas podem ser capturadas por um software que é executado em plano de fundo. Neste trabalho está sendo utilizado o software Inputlog[23], um software que executa em plano de fundo coletando todas as atividades realizadas em um computador. O processo do Inputlog é descrito por Leijten e Van Waes (2013).

### 3.1.2. Classificação dos Artefatos

A partir da coleta dos dados a próxima etapa do método é a classificação dos artefatos gerados.

Com base na análise de cada artefato ele pode ser classificado em:

- Funcional: o programa desenvolvido soluciona o problema descrito.

---

23 http://www.inputlog.net/index.html

- Imperfeito: o programa atende parcialmente o que foi solicitado, exigindo alguns ajustes para que esteja completo.

- Não funcional: o programa apresenta diversos erros e demanda diversos ajustes para que esteja funcional.

Os artefatos classificados como “Imperfeito” ou “Não funcional” passam por uma segunda análise, onde seus erros são categorizados, possibilitando identificar a dificuldade de acordo com cada paradigma.

Os erros podem ser categorizados em:

a) Erros de lógica: a lógica para escrita da solução do problema não atende ao solicitado.

b) Erros sintáticos: comandos escritos de forma incorreta; utilização de separadores ou finalizadores (parênteses, chaves e ponto e vírgula) de forma incorreta, seja pela ausência ou local incorreto.

c) Erros de fluxo: atribuição ou comparação de valores realizadas em ordem incorreta, acarretando na solução incorreta do solicitado.

d) Erros em expressões aritméticas: formulação incorreta dos cálculos necessários para solução do problema, seja na falta de valores, utilização incorreta de nomes (variáveis) ou ausência de separadores para ordem do cálculo.

e) Erros em expressões lógicas: problemas na utilização de conectivos lógicos de comparação.

f) Erros em técnicas: utilização incorreta de condições de comparação ou técnicas de repetição.

O mesmo artefato pode ter mais de um tipo de erro atrelado a ele.

Em caso do artefato gerado estar incompleto este é categorizado como incompleto.

### 3.1.3. Processamento das Atividades Geradas

A partir do arquivo gerado pelo software Inputlog, é possível observar uma lista completa de atividades coletadas em plano de fundo. Para cada uma destas atividades é possível identificar o momento em que cada evento ocorreu.

Com base na atividade e no horário registrado para cada atividade é possível calcular o tempo total de desenvolvimento, o tempo em que o editor de texto ou o ambiente de programação ficou ativo, a quantidade de teclas acionadas, a média de velocidade de digitação e também a quantidade de caracteres excluídos.

Embora seja possível mapear estas informações manualmente, este mapeamento demanda muito tempo e é passível de erro. Desta forma, um software está sendo desenvolvido para que elas sejam obtidas de forma automática.

### 3.1.4. Análise e Comparação

A partir das informações coletadas é possível comparar distintos paradigmas de programação quanto à facilidade de desenvolvimento, tipos de erros cometidos, velocidade de desenvolvimento e também produtividade.

A facilidade pode ser comparada através dos artefatos classificados como funcionais, imperfeitos e não funcionais.

Os tipos de erros podem indicar pontos críticos em cada paradigma, facilitando inclusive na identificação de pontos que devem ser melhor trabalhados no ensino de cada paradigma, como forma de mitigar os erros.

A velocidade e a produtividade podem ser comparadas a partir das métricas de tempo que são coletadas e processadas através das informações coletadas em plano de fundo.

### 3.1.5. Próximas Etapas

O método proposto ainda está em desenvolvimento, as categorizações de erros estão sendo revistas e devem ser complementadas com padrões dos erros cometidos e também exemplos.

A classificação dos erros de codificação em programas possui um eco na literatura. Esta literatura está sendo revisada e será levada em consideração nas adequações necessárias ao método para a conclusão da dissertação.

As informações relacionadas a tempo ainda estão sendo processadas, a partir do término do processamento, será possível complementar também estas informações, realizando a análise das mesmas.

Um experimento já foi realizado, comparando 2 paradigmas de programação por estudantes de um curso técnico em informática, matriculados nas disciplinas de Lógica e Linguagem de Programação e Programação I, da 1ª e 2ª série, respectivamente. O experimento e os resultados estão descritos no próximo capítulo.

# 4. EXPERIMENTO

Entende-se que uma das etapas para a proposição do método de comparação entre paradigmas de programação é a aplicação deste método comparando dois paradigmas de programação quanto à aprendizagem de novatos em programação. Desta forma, um experimento foi realizado e será descrito nesta seção.

Serão descritos a escolha dos paradigmas, a população utilizada para coleta de dados, a utilização do Termo de Consentimento Livre e Esclarecido, as etapas utilizadas para coleta de dados e, por fim, os questionários aplicados.

## 4.1. ESCOLHA DOS PARADIGMAS

Os paradigmas escolhidos para validar o método proposto foram os Paradigmas Imperativo-Procedimental e o Paradigma Lógico, utilizando mais precisamente Sistemas Baseados em Regras.

A escolha do PP deve-se ao fato deste paradigma ser, conforme Vujošević-Janičić e Tošić (2008), usualmente o paradigma mais utilizado para o ensino de programação de computadores para novatos. Embora existam estudos que demonstrem a escolha do POO como primeiro paradigma.

O segundo paradigma, o PL, especificamente utilizando SBRs, deve-se ao fato da suposta facilidade de aprendizagem deste paradigma, relatado por Shimokura, Nakanishi e Ohta (2008) e Arakliotis, Nikolos e Kalligeros (2016).

Com a escolha de SBR também se busca reforçar a facilidade de utilização do PON e sua LingPON, relatada por Krug (2016a), sendo ambos temática de pesquisa do grupo de pesquisa no qual este trabalho se insere. Apesar do PON ser um outro paradigma, descrito por Simão e Stadzisz (2008), este reserva características semelhantes às de um SBR no tocante ao desenvolvimento orientado a regras, particularmente na sua materialização chamada LingPON.

Outrossim, para cada paradigma uma linguagem de programação foi escolhida, sendo Pascal e C para o PP e CLIPS para o PL-SBR.

O Pascal foi escolhida devido ao fato de ser uma programação utilizada com muita frequência como primeira linguagem de programação, inclusive uma das razões de sua criação foi a educação (WIEDENBECK et al., 1999).

A linguagem C foi escolhida devido ao fato de uma parte da população que participou do teste já ter aprendido esta linguagem, permitindo a comparação com uma linguagem mais familiar aos participantes.

Para SBR a linguagem CLIPS foi escolhida devido a sua sintaxe simples, reservando certa similaridade ao Pascal, conforme observado pelo autor durante a busca por linguagens de SBR.

## 4.2. POPULAÇÃO

Para validação do método proposto, a população escolhida foi a de alunos matriculados na 1ª e na 2ª série do Curso Técnico em Informática Integrado ao Ensino Médio do Instituto Federal do Paraná – IFPR, campus União da Vitória. Em tempo, o autor deste trabalho é docente do quadro de professores deste IFPR.

Isto considerado, a grande maioria destes alunos estão tendo seu primeiro contato com programação de computadores durante o curso, nos componentes curriculares de Lógica e Linguagem de Programação e Programação I.

Os alunos da 1ª série, que estão cursando o componente curricular de Lógica e Linguagem de Programação, tiveram apenas um primeiro contato com pseudocódigo procedimental antes do início do experimento.

Os alunos da 2ª série, que estão cursando o componente curricular de Programação I, tiveram um ano de programação utilizando o Paradigma Procedimental, aprendendo a linguagem de programação C.

Participaram do experimento 78 alunos da 1ª série, divididos em 2 turmas, nomeadas Turma A e Turma B, e 40 alunos da 2ª série, de uma única turma.

### 4.2.1. Termo de Consentimento Livre e Esclarecido

Para obter permissão da utilização dos dados dos alunos, os responsáveis assinaram um Termo de Consentimento Livre e Esclarecido, que esclarece os objetivos da pesquisa, deixando claro que em hipótese alguma os dados serão demonstrados individualmente ou haverá identificação dos participantes. O documento utilizando encontra-se na Seção Complementar C.

## 4.3. ETAPAS DA COLETA DE DADOS

Nesta subseção serão explicadas as etapas utilizadas para coleta de dados. As etapas foram distintas de acordo com a série em que os alunos estão matriculados, pois os alunos da 2ª série já tiveram contato anterior com o PP.

As etapas de coleta de dados ocorreram durante as aulas dos componentes curriculares de Lógica e Linguagem de Programação e Programação I, para a 1ª e 2ª série, respectivamente. Como a participação no experimento é livre, os alunos que não quiseram participar do experimento não tiveram seus dados utilizados, embora estes tenham participado dos encontros utilizados para o experimento, sem prejuízo para os mesmos.

Os alunos da 1ª série, conforme mencionado anteriormente, estão divididos em 2 turmas, turma A e turma B. Devido à limitação de espaço físico do laboratório utilizado no experimento, estas turmas foram subdividas em 2 grupos cada, tendo no máximo 20 alunos em cada grupo.

A mesma subdivisão foi realizada com a turma da 2ª série, também cada grupo ficou com no máximo 20 alunos.

Para as turmas da 1ª série a coleta de dados seguiu as mesmas etapas para ambos os paradigmas, sendo que a turma A iniciou com um paradigma e a turma B com outro, posteriormente houve inversão dos paradigmas e reinício das etapas.

As etapas realizadas foram as seguintes, sendo elas modeladas na Figura 10 e na Figura 11:

- Instrução: foram realizados 4 encontros de instrução, com duração de 01h42 cada (2 aulas de 51 minutos), a respeito do paradigma e da linguagem de programação escolhidos, estes encontros foram realizados em laboratório de informática no qual cada aluno teve um computador à sua disposição. As aulas foram divididas em teoria e prática e foram baseadas nas apostilas criadas pelo autor para este fim. As apostilas encontram-se nas Seções Complementares D e E. Etapa representada pelas atividades 1 e 6 no diagrama de atividades ilustrado na Figura 10 e atividade 4 no diagrama ilustrado na Figura 11.

- Atividade avaliativa: após a realização dos encontros de instrução os alunos foram submetidos a uma atividade avaliativa, na qual os alunos deveriam desenvolver uma solução para um dado problema utilizando o paradigma e a linguagem de programação escolhidos. Para desenvolver a atividade avaliativa os

alunos puderam consultar a apostila e as anotações realizadas por eles durante os encontros de instrução, os computadores estavam sem acesso à Internet. As atividades realizadas pelo aluno foram coletadas pelo programa Inputlog. Etapa representada pelas atividades 2 e 7 no diagrama de atividades ilustrado na Figura 10 e atividades 1 e 5 no diagrama ilustrado na Figura 11.

- Questionário de avaliação: após concluir a atividade avaliativa, os alunos responderam a um breve questionário avaliando as atividades de instrução, a atividade avaliativa, o material utilizado pelo professor e o paradigma e linguagem utilizados. Etapa representada pelas atividades 3 e 8 no diagrama de atividades ilustrado na Figura 10 e atividades 2 e 6 no diagrama ilustrado na Figura 11.

- Correção: foi realizada a correção dos programas gerados pelos alunos e informado aos alunos se o programa desenvolvido atendeu ao que foi solicitado ou se houve algum problema. Etapa representada pelas atividades 4 e 9 no diagrama de atividades ilustrado na Figura 10 e atividades 3 e 7 no diagrama ilustrado na Figura 11.

- Inversão do paradigma: após realizar as atividades para um paradigma, caso o aluno tenha realizado as etapas apenas para um deles, as etapas foram reiniciadas utilizando o outro paradigma. Etapa representada pelas atividades 5 e 10 no diagrama de atividades ilustrado na Figura 10.

Para a turma da 2ª série, devido à experiência prévia de programação com o PP e com a linguagem C, os alunos iniciaram realizando a resolução de um problema utilizando o PP, etapa representada pela atividade 1 no diagrama de atividades ilustrado na Figura 11. Desta forma, para o PP a etapa de instrução não foi realizada com os grupos da 2ª série.

Após a realização da atividade avaliativa, os alunos da 2ª série iniciaram com SBR, a partir da etapa de instrução. A apostila utilizada com os alunos da 1ª série foi complementada com algumas informações necessárias para os alunos da 2ª série, aumentando um pouco a quantidade de recursos explicados, especificamente a execução de ações por repetidas vezes.

A turma da 1ª série A iniciou com o PP e a turma da 1ª série B iniciou com SBR. Conforme previsto houve inversão dos paradigmas após a atividade avaliativa.

As etapas estão ilustradas nas Figuras 10 e 11.

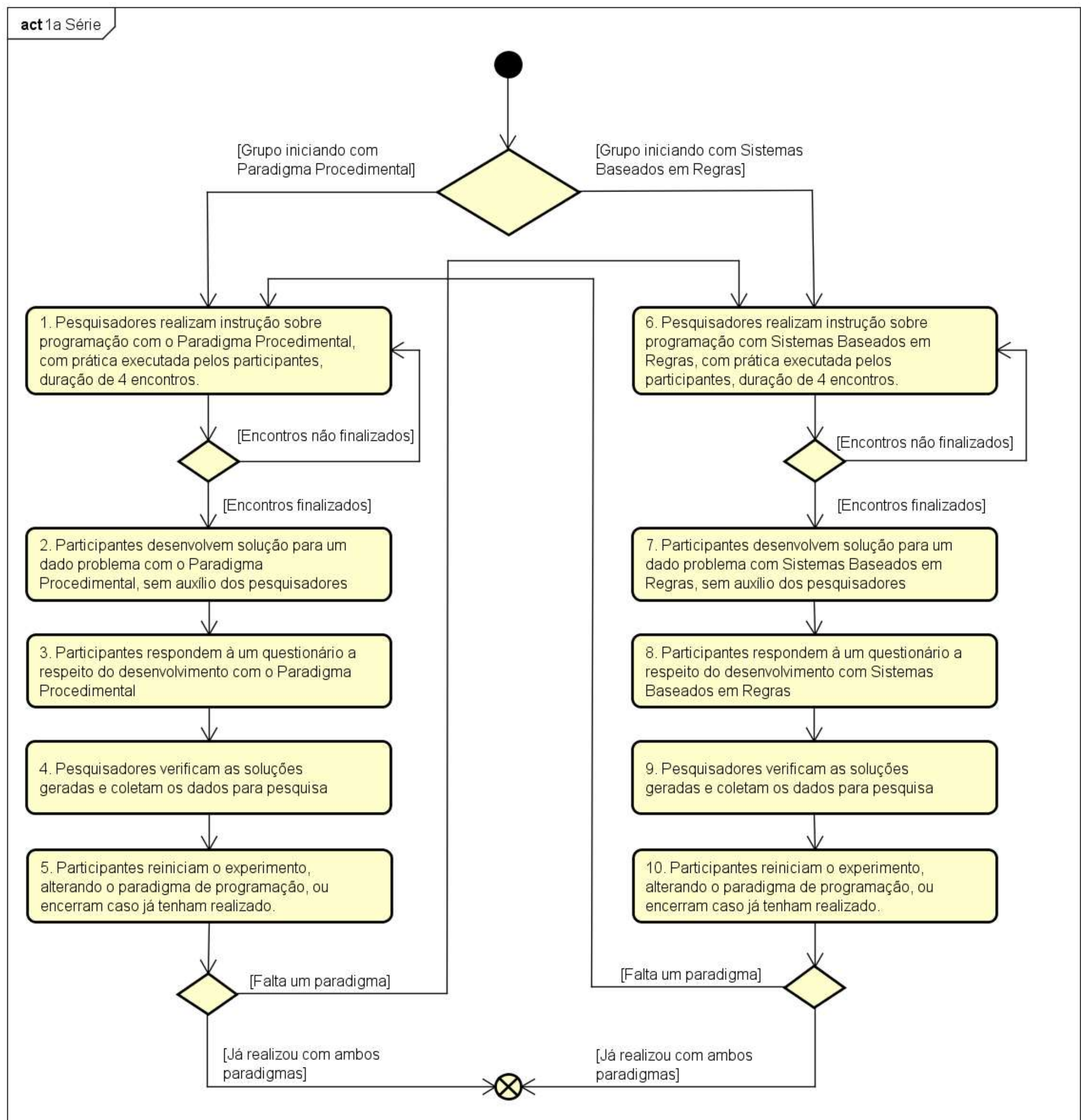



Figura 10 – Etapas - alunos da 1a série.

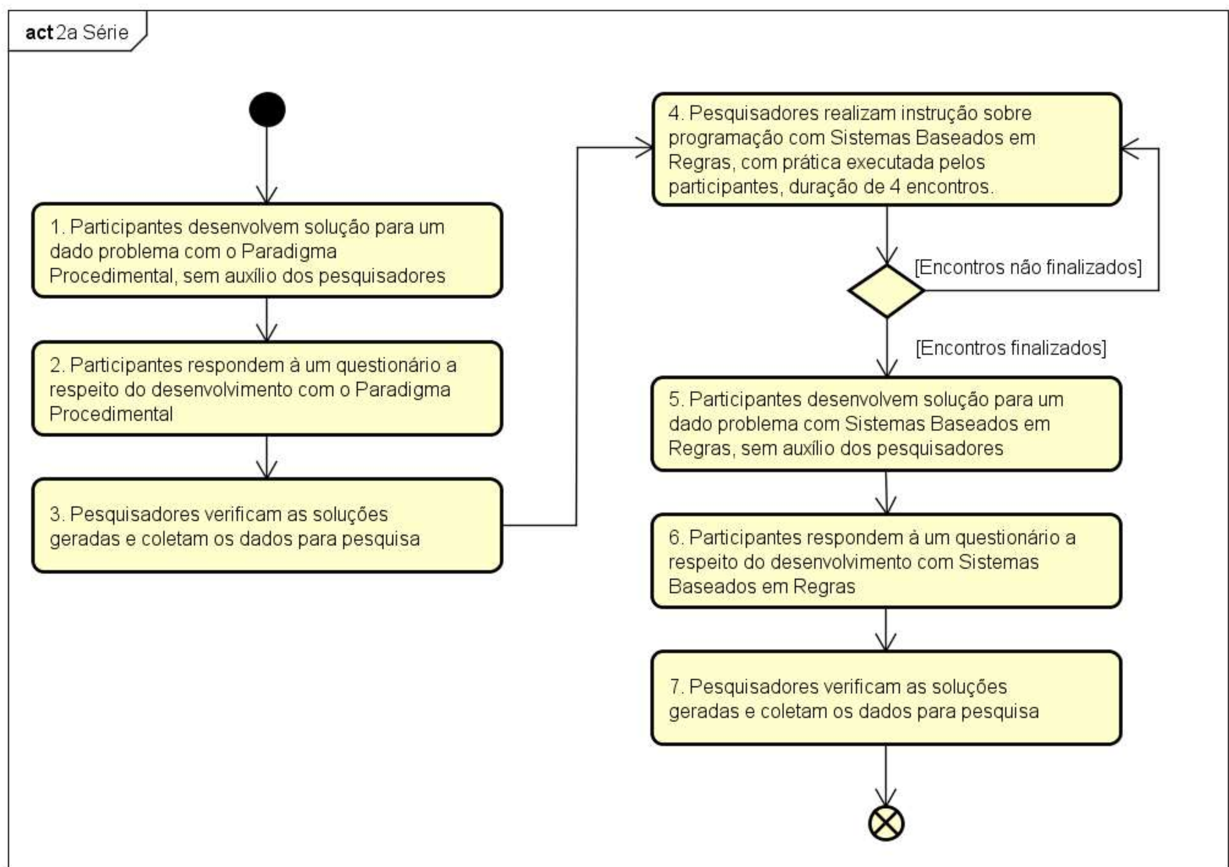



Figura 11 – Etapas - alunos da 2a série.

## 4.4. PROBLEMAS PROPOSTOS

Para a etapa de atividade avaliativa, os alunos tiveram que criar a solução para um problema no paradigma que estavam aprendendo no momento. O mesmo problema foi solucionado em ambos os paradigmas após o final da instrução de cada paradigma.

A complexidade dos problemas foi distinta entre as 2 séries participantes, o enunciado de cada problema está descrito a seguir.

**- Problema para a 1ª série:**

Escreva um programa utilizando a Linguagem Pascal / CLIPS que informe a renda *per capita* de uma família com 4 pessoas. Para isso solicite o salário de cada uma das pessoas e calcule a média (renda *per capita*).

Após calcular a média o programa deve informar em qual faixa de renda esta família se encaixa, as faixas de renda estão listadas a seguir:

Faixa 1 – Renda *per capita* até R$ 800,00

Faixa 2 – Renda *per capita* entre R$ 800,01 e R$ 1600,00

Faixa 3 – Renda *per capita* entre R$ 1600,01 e R$ 3000,00

Faixa 4 – Renda *per capita* acima de R$ 3000,00

**- Problema para a 2ª série:**

Escreva um programa utilizando a Linguagem C / CLIPS que informe os 20 primeiros números da sequência de Fibonacci. O programa deve solicitar ao usuário os 2 primeiros números da sequência.

A sequência de Fibonacci calcula o valor do próximo número através da soma dos 2 números imediatamente anteriores. Por exemplo, 1, 2, 3, 5, 8….

## 4.5. QUESTIONÁRIO

Após a realização da atividade avaliativa utilizada para comparação dos paradigmas de programação, cada aluno participante do experimento respondeu um questionário de avaliação.

Este questionário é composto por perguntas para avaliar o experimento, os encontros de instrução, a apostila, o paradigma de programação a partir do ponto de vista do aluno quanto ao entendimento do código e quanto à facilidade de programação, a facilidade dos exercícios e a facilidade do exercício final.

O questionário aplicado para a turma da 2ª série é um pouco distinto para o PP, pois o exercício avaliativo foi aplicado diretamente, sem os encontros de instrução.

Os questionários estão descritos na Seção Complementar F e foram baseados em trabalhos correlatos (BINI, 2010; TISSOT, 2015).

Todos os questionários foram aplicados via Google Forms[24].

Os resultados do questionário, da classificação dos artefatos e da categorização de erros estão descritos no próximo capítulo.

---

24 https://www.google.com/forms/about/

# 5. RESULTADOS

A partir do experimento realizado é possível observar resultados parciais e indicar o comparativo entre os paradigmas PP e PL-SBR.

Nesta seção estão descritos os resultados de acordo com o método proposto e também os resultados dos questionários aplicados.

Para melhor identificar os participantes os grupos serão nomeados conforme ilustrado no Quadro 1.

| IDENTIFICAÇÃO | DESCRIÇÃO | PARTICIPANTES |
|---|---|---|
| 1A | Todos os alunos da 1ª Série – Turma A | 38 |
| 1B | Todos os alunos da 1ª Série – Turma B | 40 |
| 2 | Todos os alunos da 2ª Série | 40 |
| 1A-1 | Alunos da 1ª Série – Turma A – Grupo 1 | 20 |
| 1A-2 | Alunos da 1ª Série – Turma A – Grupo 2 | 18 |
| 1B-1 | Alunos da 1ª Série – Turma B – Grupo 1 | 20 |
| 1B-2 | Alunos da 1ª Série – Turma B – Grupo 2 | 20 |
| 2-1 | Alunos da 2ª Série – Grupo 1 | 21 |
| 2-2 | Alunos da 2ª Série – Grupo 2 | 19 |

Quadro 1 – Participantes do experimento.

## 5.1. CLASSIFICAÇÃO DOS ARTEFATOS

Cada turma criou um artefato para cada paradigma, servindo como atividade avaliativa após o ciclo de instruções. A exceção é a turma da 2ª série, que criou o artefato de PP sem etapa prévia de instrução, pois o paradigma já vem sendo trabalhado com eles há, pelo menos, um ano.

Os artefatos foram analisados e classificados, conforme proposto pelo método, em: Funcional (FC); Imperfeito (IP); e Não Funcional (NF).

Na Tabela 1 está listado o percentual para cada classificação em cada turma e cada paradigma.

Tabela 1 - Percentual das classificações para cada turma e cada paradigma.

| TURMA | FC-PP | FC-SBR | IP-PP | IP-SBR | NF-PP | NF-SBR |
|---|---|---|---|---|---|---|
| 1A | 47,4 % | 31,6 % | 47,4 % | 42,1 % | 5,3 % | 26,3 % |
| 1B | 50,0 % | 17,5 % | 22,5 % | 30,0 % | 27,5 % | 52,5 % |
| 2 | 40,0 % | 22,5 % | 22,5 % | 15,0 % | 37,5 % | 62,5 % |
| 1A-1 | 45,0 % | 45,0 % | 50,0 % | 25,0 % | 5,0 % | 30,0 % |
| 1A-2 | 50,0 % | 16,7 % | 44,4 % | 61,1 % | 5,6 % | 22,2 % |
| 1B-1 | 50,0 % | 25,0 % | 15,0 % | 20,0 % | 35,0 % | 55,0 % |
| 1B-2 | 50,0 % | 10,0 % | 30 % | 40,0 % | 20,0 % | 50,0 % |
| 2-1 | 28,3 % | 9,5 % | 23,8 % | 19,0 % | 47,6 % | 71,4 % |

| TURMA | FC-PP | FC-SBR | IP-PP | IP-SBR | NF-PP | NF-SBR |
|---|---|---|---|---|---|---|
| 2-2 | 52,6 % | 36,8 % | 21,1 % | 10,5 % | 26,3 % | 52,6 % |

Fonte: Autoria própria.

A distribuição também está ilustrada na Figura 12.

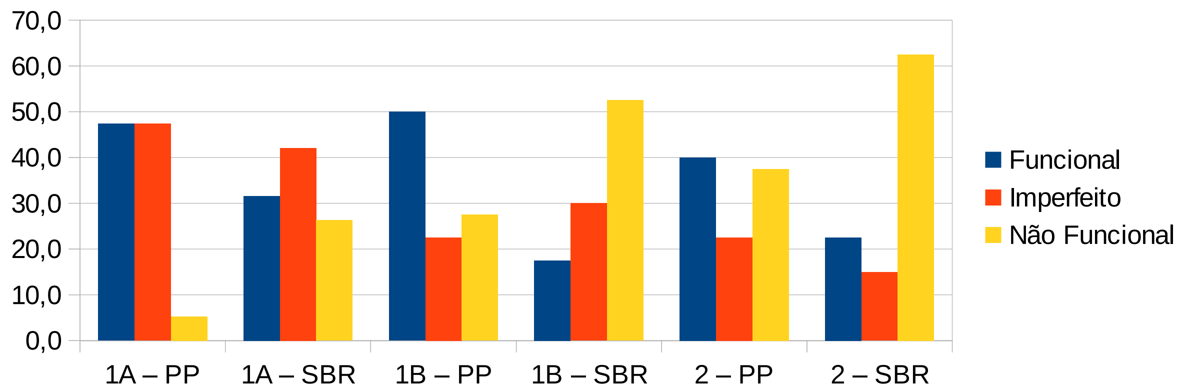



Figura 12 – Gráfico com a classificação dos artefatos.

## 5.2. CATEGORIZAÇÃO DE ERROS

Durante a classificação dos artefatos gerados, também foi realizada a categorização de erros para aqueles que não foram classificados como Funcional.

Os erros foram categorizados em: Erros de Lógica (LOG); Erros Sintáticos (SIN); Erros de Fluxo (FLX); Erros em Expressões Aritméticas (EXA); Erros em Expressões Lógicas (EXL); Erros em Técnicas (ET); e Incompleto (IN).

Tabela 2 - Percentual das classificações para cada turma e cada paradigma.

| TURMA/ PARADIGMA | LOG | SIN | FLX | EXA | EXL | ET | IN |
|---|---|---|---|---|---|---|---|
| 1A-PP | 20,0 % | 10,0 % | 5,0 % | 35,0 % | 55,0 % | 15,0 % | 10,0 % |
| 1A-SBR | 42,3 % | 53,8 % | 3,8 % | 3,8 % | 26,9 % | 46,2 % | 19,2 % |
| 1B-PP | 60,0 % | 20,0 % | 5,0 % | 50,0 % | 35,0 % | 40,0 % | 20,0 % |
| 1B-SBR | 54,5 % | 60,6 % | 12,1 % | 27,3 % | 3,0 % | 51,5 % | 57,6 % |
| 2-PP | 100 % | 4,2 % | 50,0 % | 4,2 % | 0,0 % | 25,0 % | 4,2 % |
| 2-SBR | 83,9 % | 58,1 % | 22,6 % | 6,5 % | 0,0 % | 64,5 % | 16,1 % |
| 1A-1-PP | 18,2 % | 9,1 % | 9,1 % | 45,5 % | 72,7 % | 9,1 % | 9,1 % |
| 1A-1-SBR | 27,3 % | 54,5 % | 0,0 % | 0,0 % | 27,3 % | 63,6 % | 36,4 % |
| 1A-2-PP | 22,2 % | 11,1 % | 0,0 % | 22,2 % | 33,3 % | 22,2 % | 11,1 % |
| 1A-2-SBR | 53,3 % | 53,3 % | 6,7 % | 6,7 % | 26,7 % | 33,3 % | 6,7 % |
| 1B-1-PP | 60,0 % | 40,0 % | 0,0 % | 50,0 % | 30,0 % | 50,0 % | 30,0 % |
| 1B-1-SBR | 46,7 % | 66,7 % | 6,7 % | 33,3 % | 6,7 % | 53,5 % | 66,7 % |
| 1B-2-PP | 60,0 % | 0,0 % | 10,0 % | 50,0 % | 40,0 % | 30,0 % | 10,0 % |
| 1B-2-SBR | 61,1 % | 55,6 % | 16,7 % | 22,2 % | 0,0 % | 50,0 % | 50,0 % |
| 2-1-PP | 100 % | 0,0 % | 60,0 % | 6,7 % | 0,0 % | 33,3 % | 0,0 % |
| 2-1-SBR | 84,2 % | 47,4 % | 21,1 % | 0,0 % | 0,0 % | 57,9 % | 15,8 % |

| TURMA/ PARADIGMA | LOG | SIN | FLX | EXA | EXL | ET | IN |
|---|---|---|---|---|---|---|---|
| 2-2-PP | 100 % | 11,1 % | 33,3 % | 0,0 % | 0,0 % | 11,1 % | 11,1 % |
| 2-2-SBR | 83,3 % | 75,0 % | 25,0 % | 16,7 % | 0,0 % | 75,0 % | 16,7 % |

Fonte: Autoria própria.

Nota: Percentual calculado com base no total de artefatos classificados como Imperfeito ou Não Funcional.

A distribuição também pode ser observada nas Figuras 13 e 14.

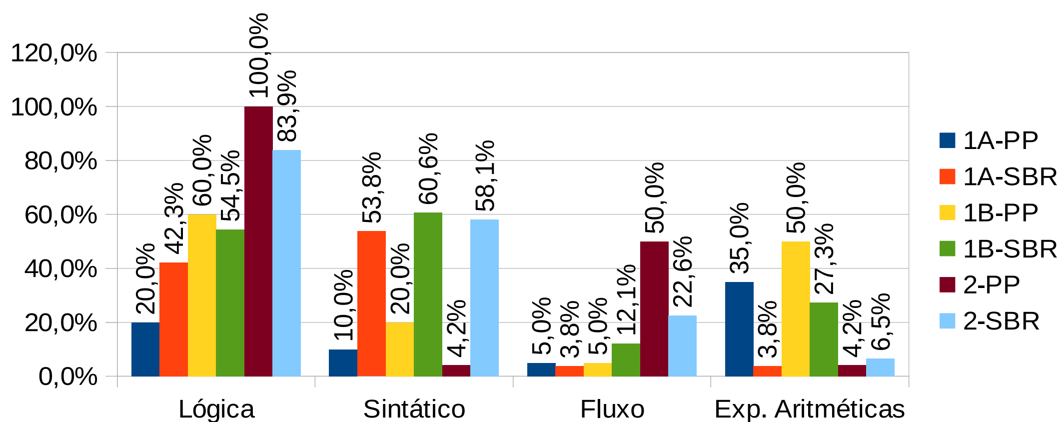



Figura 13 – Gráfico com a classificação dos artefatos.

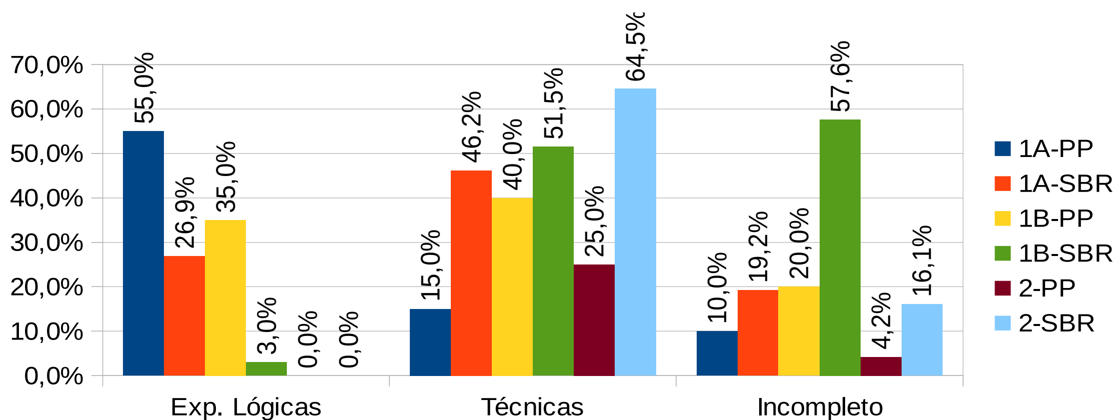



Figura 14 – Gráfico com a classificação dos artefatos.

## 5.3. QUESTIONÁRIOS APLICADOS

Após a realização da atividade avaliativa os alunos responderam um questionário avaliando os encontros de instrução, o material de apoio, o paradigma e a linguagem de programação utilizado.

As questões podem ser divididas em duas categorias, a primeira com informações pessoais do respondente e a segunda com a avaliação do experimento. Nesta seção estão descritos os resultados do questionário aplicado.

### 5.3.1. Informações Pessoais

No questionário foi perguntado ao respondente a idade, o gênero e se o mesmo tinha algum conhecimento de programação antes de iniciar o curso.

O resultado destas questões está descrito nas Tabelas 3, 4 e 5.

Tabela 3 - Percentual dos participantes por idade.

| TURMA | 14 | 15 | 16 | 17 | 18 |
|---|---|---|---|---|---|
| 1A-PP | 13,2 % | 68,4 % | 15,8 % | 2,6 % | 0,0 % |
| 1A-SBR | 5,3 % | 63,2 % | 26,3 % | 5,3 % | 0,0 % |
| 1B-PP | 15,0 % | 75,0 % | 10,0 % | 0,0 % | 0,0 % |
| 1B-SBR | 32,5 % | 65,0 % | 2,5 % | 0,0 % | 0,0 % |
| 2-PP | 0,0 % | 26,2 % | 61,9 % | 7,1 % | 4,8 % |
| 2-SBR | 0,0 % | 25,0 % | 52,5 % | 17,5 % | 5,0 % |

Fonte: Questionário aplicado aos alunos.

Nota: O percentual pode variar na mesma turma devido à aplicação do questionário em datas diferentes.

Tabela 4 - Percentual dos participantes por gênero.

| TURMA | FEMININO | MASCULINO |
|---|---|---|
| 1A | 55,3 % | 44,7 % |
| 1B | 45,0 % | 55,0 % |
| 2 | 32,5 % | 67,5 % |

Fonte: Questionário aplicado aos alunos.

Tabela 5 - Percentual dos participantes por conhecimento anterior em programação.

| TURMA | SIM | NÃO |
|---|---|---|
| 1A | 7,9 % | 91,1 % |
| 1B | 5,0 % | 95,0 % |
| 2 | 20,0 % | 80,0 % |

Fonte: Questionário aplicado aos alunos.

### 5.3.2. Avaliação Relacionada ao Paradigma

As demais questões foram relacionadas ao paradigma e ao experimento, iniciando com uma questão ao aluno se o mesmo dedicou tempo fora dos encontros para estudo do paradigma.

Os resultados estão listados na Tabela 6.

Tabela 6 - Percentual dos participantes que dedicou tempo de estudo.

| TURMA | SIM | NÃO |
|---|---|---|
| 1A-PP | 57,9 % | 42,1 % |
| 1A-SBR | 50,0 % | 50,0 % |
| 1B-PP | 37,5 % | 62,5 % |
| 1B-SBR | 60,0 % | 40,0 % |
| 2-PP | - | - |
| 2-SBR | 67,5 % | 32,5 % |

Fonte: Questionário aplicado aos alunos.

As demais questões foram com respostas de avaliação linear, na qual a resposta com menor classificação é 1 e a com maior classificação é 5.

Para cada questão foi aplicado o percentual de acordo com a escala escolhida e o número de respondentes.

Algumas questões estão sem respostas para a turma da 2ª série com o PP, pois este paradigma teve distinção no experimento devido ao conhecimento anterior do paradigma pela turma.

Nas tabelas a seguir estão listadas as avaliações dos respondentes quanto à motivação (Tabela 7), facilidade da ferramenta (Tabela 8), clareza das explicações (Tabela 9), clareza da apostila utilizada (Tabela 10), utilidade dos exemplos (Tabela 11), facilidade dos exercícios (Tabela 12), facilidade do exercício avaliativo (Tabela 13), entendimento do código (Tabela 14), facilidade de programação (Tabela 15) e experiência geral do experimento (Tabela 16).

As questões podem ser verificadas na integra na Seção Complementar F.

Tabela 7 – Respostas quanto à motivação.

| TURMA | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1A-PP | 0,0 % | 0,0 % | 18,4 % | 42,1 % | 39,5 % |
| 1A-SBR | 0,0 % | 5,3 % | 18,4 % | 42,1 % | 34,2 % |
| 1B-PP | 2,5 % | 2,5 % | 17,5 % | 25,0 % | 52,5 % |
| 1B-SBR | 0,0 % | 7,5 % | 25,0 % | 55,0 % | 12,5 % |
| 2-PP | - | - | - | - | - |
| 2-SBR | 0,0 % | 7,5 % | 30,0 % | 37,5 % | 25,0 % |

Fonte: Questionário aplicado aos alunos.

Tabela 8 – Respostas quanto à facilidade da ferramenta.

| TURMA | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1A-PP | 0,0 % | 13,2 % | 26,3 % | 36,8 % | 23,7 % |
| 1A-SBR | 10,5 % | 13,2 % | 34,2 % | 21,1 % | 21,1 % |
| 1B-PP | 2,5 % | 0,0 % | 15,0 % | 30,0 % | 52,5 % |
| 1B-SBR | 2,5 % | 22,5 % | 42,5 % | 20,0 % | 12,5 % |
| 2-PP | 4,8 % | 11,9 % | 19,0 % | 40,5 % | 23,8 % |
| 2-SBR | 12,5 % | 12,5 % | 42,5 % | 22,5 % | 10,0 % |

Fonte: Questionário aplicado aos alunos.

Tabela 9 – Respostas quanto à clareza das explicações.

| TURMA | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1A-PP | 0,0 % | 0,0 % | 2,6 % | 13,2 % | 84,2 % |
| 1A-SBR | 2,6 % | 0,0 % | 13,2 % | 7,9 % | 76,3 % |
| 1B-PP | 2,5 % | 0,0 % | 2,5 % | 7,5 % | 87,5 % |
| 1B-SBR | 0,0 % | 0,0 % | 7,5 % | 42,5 % | 50,0 % |
| 2-PP | - | - | - | - | - |
| 2-SBR | 0,0 % | 2,5 % | 10,0 % | 22,5 % | 65,0 % |

Fonte: Questionário aplicado aos alunos.

Tabela 10 – Respostas quanto à clareza da apostila.

| TURMA | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1A-PP | 0,0 % | 2,6 % | 5,3 % | 34,2 % | 57,9 % |
| 1A-SBR | 0,0 % | 5,3 % | 13,2 % | 31,6 % | 50,0 % |
| 1B-PP | 2,5 % | 0,0 % | 2,5 % | 25,0 % | 70,0 % |
| 1B-SBR | 0,0 % | 5,0 % | 15,0 % | 45,0 % | 35,0 % |
| 2-PP | - | - | - | - | - |
| 2-SBR | 5,0 % | 2,5 % | 17,5 % | 40,0 % | 35,0 % |

Fonte: Questionário aplicado aos alunos.

Tabela 11 – Respostas quanto à utilidade dos exemplos.

| TURMA | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1A-PP | 2,6 % | 0,0 % | 2,6 % | 31,6 % | 63,2 % |
| 1A-SBR | 2,6 % | 0,0 % | 13,2 % | 31,6 % | 52,6 % |
| 1B-PP | 2,5 % | 0,0 % | 12,5 % | 32,5 % | 52,5 % |
| 1B-SBR | 0,0 % | 5,0 % | 17,5 % | 52,5 % | 25,0 % |
| 2-PP | - | - | - | - | - |
| 2-SBR | 2,5 % | 2,5 % | 20,0 % | 32,5 % | 42,5 % |

Fonte: Questionário aplicado aos alunos.

Tabela 12 – Respostas quanto à facilidade dos exercícios.

| TURMA | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1A-PP | 0,0 % | 10,5 % | 39,5 % | 31,6 % | 18,4 % |
| 1A-SBR | 5,3 % | 7,9 % | 28,9 % | 31,6 % | 26,3 % |
| 1B-PP | 2,5 % | 2,5 % | 22,5 % | 30,0 % | 42,5 % |

| TURMA | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1B-SBR | 5,0 % | 17,5 % | 32,5 % | 32,5 % | 12,5 % |
| 2-PP | - | - | - | - | - |
| 2-SBR | 7,5 % | 12,5 % | 40,0 % | 25,0 % | 15,0 % |

Fonte: Questionário aplicado aos alunos.

Tabela 13 – Respostas quanto à facilidade do exercício final.

| TURMA | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1A-PP | 0,0 % | 23,7 % | 36,8 % | 26,3 % | 13,2 % |
| 1A-SBR | 10,5 % | 10,5 % | 23,7 % | 18,4 % | 36,8 % |
| 1B-PP | 5,0 % | 7,5 % | 12,5 % | 37,5 % | 37,5 % |
| 1B-SBR | 15,0 % | 10,0 % | 40,0 % | 30,0 % | 5,0 % |
| 2-PP | 11,9 % | 9,5 % | 47,6 % | 19,0 % | 11,9 % |
| 2-SBR | 27,5 % | 17,5 % | 27,5 % | 15,0 % | 12,5 % |

Fonte: Questionário aplicado aos alunos.

Tabela 14 – Respostas quanto ao entendimento do código.

| TURMA | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1A-PP | 0,0 % | 5,3 % | 36,8 % | 44,7 % | 13,2 % |
| 1A-SBR | 5,3 % | 7,9 % | 36,8 % | 28,9 % | 21,1 % |
| 1B-PP | 0,0 % | 2,5 % | 27,5 % | 37,5 % | 32,5 % |
| 1B-SBR | 5,0 % | 7,5 % | 45,0 % | 37,5 % | 5,0 % |
| 2-PP | 2,4 % | 2,4 % | 45,2 % | 23,8 % | 26,2 % |
| 2-SBR | 7,5 % | 5,0 % | 40,0 % | 37,5 % | 10,0 % |

Fonte: Questionário aplicado aos alunos.

Tabela 15 – Respostas quanto à facilidade de programação.

| TURMA | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1A-PP | 7,9 % | 5,3 % | 26,3 % | 36,8 % | 23,7 % |
| 1A-SBR | 7,9 % | 7,9 % | 26,3 % | 31,6 % | 26,3 % |
| 1B-PP | 2,5 % | 2,5 % | 12,5 % | 37,5 % | 45,0 % |
| 1B-SBR | 2,5 % | 12,5 % | 45,0 % | 32,5 % | 7,5 % |
| 2-PP | 4,8 % | 14,3 % | 26,2 % | 35,7 % | 19,0 % |
| 2-SBR | 5,0 % | 5,0 % | 27,5 % | 50,0 % | 12,5 % |

Fonte: Questionário aplicado aos alunos.

Tabela 16 – Respostas quanto à experiência geral do experimento.

| TURMA | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1A-PP | 0,0 % | 0,0 % | 15,8 % | 39,5 % | 44,7 % |
| 1A-SBR | 0,0 % | 7,9 % | 26,3 % | 26,3 % | 39,5 % |
| 1B-PP | 2,5 % | 0,0 % | 17,5 % | 35,0 % | 45,0 % |
| 1B-SBR | 2,5 % | 10,0 % | 32,5 % | 30,0 % | 25,0 % |
| 2-PP | 9,5 % | 7,1 % | 31,0 % | 23,8 % | 28,6 % |
| 2-SBR | 2,5 % | 5,0 % | 22,5 % | 40,0 % | 32,5 % |

Fonte: Questionário aplicado aos alunos.

A discussão sobre os dados descritos neste capítulo é realizada no próximo capítulo.

## 6. DISCUSSÃO

Neste capítulo serão realizadas as discussões sobre o método proposto tendo como base os dados coletados durante o experimento realizado com as turmas das 1ª e 2ª séries, considerando o programa e as respostas ao questionário.

Iniciando com a classificação dos artefatos gerados em "Funcional", "Imperfeito" e "Não funcional" é possível observar que em todas as turmas participantes o percentual de artefatos classificados como "Funcional" é maior para o PP. Seja na turma completa ou ainda nas subdivisões das turmas, com exceção para a subdivisão 1A-1, na qual o percentual de artefatos classificados como "Funcional" é o mesmo (45%).

De forma inversa, o percentual dos artefatos classificados como "Não funcional" é maior para o SBR. Os artefatos classificados como "Imperfeito" variam de acordo com as demais classificações, não apresentando padrão entre os grupos.

Utilizando apenas os dados da série inteira, é possível observar que a turma nominada como 1A (1ª série, turma A) teve uma maior proximidade dos artefatos classificados como "Funcional", sendo 47,4% para o PP e 31,6% para o SBR. Pertinente mencionar que esta turma iniciou o experimento com o PP e finalizou com o SBR.

Poderia levantar-se uma hipótese de que o ensino do SBR após o PP tenha influenciado neste resultado, entretanto, esta hipótese não pode ser corroborada se for considerado o resultado da 2ª série, na qual 40,0% dos artefatos foram classificados como "Funcional" para o PP e apenas 22,5% para o SBR.

Considerando as categorizações de erros realizada para os artefatos classificados como "Imperfeito" ou "Não funcional", é possível observar a diferença entre os erros cometidos em cada paradigma, reforçando a diferença de aprendizado de cada paradigma.

Observando os artefatos em que houve erros de lógica é possível verificar diferenças entre os grupos. A turma 1A teve 20% dos artefatos classificados com este erro para o PP e 42,3% para SBR. Já a turma 1B, que resolveu o mesmo problema, teve 60% para o PP e 54,5% para SBR. A turma da 2ª série teve 100% dos artefatos classificados com este erro para o PP e 83,9% para o SBR. Desta forma, não houve padrão de categorização do erro de lógica entre os grupos.

Por sua vez, nos artefatos onde ocorreu o erro de sintaxe houve padrão. Em todos os grupos o percentual de artefatos com este erro foi maior para o SBR em comparação com o PP. Sendo para a turma 1A 10,0% para o PP e 53,8% para o SBR, para a turma 1B 20,0% para o PP e 60,6% para o SBR e para a 2ª série 4,2% para o PP e 58,1% para o SBR.

Para o erro de fluxo, assim como o erro de lógica, não houve padrão entre os grupos. Nas turmas da 1ª série a diferença entre os paradigmas não foi grande, sendo que para o grupo 1A o percentual de artefatos com erro de fluxo foi de 5,0% para o PP e 3,8% para o SBR, já para o grupo 1B o percentual foi de 5,0% para o PP e 12,1 para o SBR. Para a turma da 2ª série a diferença entre os paradigmas foi grande, sendo que 50,0% dos artefatos em PP apresentaram erro de fluxo enquanto 22,6% para o SBR apresentaram o mesmo erro.

Para os artefatos que apresentaram erro em expressão aritmética houve um padrão nas turmas da 1ª série, sendo que para o PP houve um maior percentual de erros do que para o SBR. Para o grupo 1A 35,0% dos artefatos apresentaram esse tipo de erro em PP e 3,8% em SBR. Para o grupo 1B 50,0% para o PP e 27,3% para o SBR. Já para a turma da 2ª série o percentual de erros não foi muito distinto, sendo 4,2% para o PP e 6,5% para o SBR.

Houve padrão também para os artefatos que tiveram erros em expressões lógicas. Para as turmas da 1ª série, os artefatos do PP apresentaram um percentual maior em comparação aos artefatos do SBR. Sendo que, para a turma 1A, foi de 55,0% para o PP e de 26,9% para o SBR. Para a turma 1B 35,0% para o PP e 3,0% para o SBR. Os artefatos da 2ª série tiveram 0,0% para ambos os paradigmas.

De forma inversa, os artefatos que tiveram erro em técnicas apresentaram um maior percentual de erros para o SBR em comparação ao PP. Para o grupo 1A 15,0% dos artefatos apresentaram erros para o PP e 46,2% para o SBR. Para o grupo 1B 40,0% para o PP e 51,5% para o SBR. Para a 2ª série 25,0% para o PP e 64,5% para o SBR.

Da mesma forma os artefatos classificados como incompletos apresentaram um maior percentual para o SBR em comparação ao PP. Sendo para o grupo 1A 10,0% para o PP e 19,2% para o SBR. Para o grupo 1B 20,0% para o PP e 57,6% para o SBR. Para a 2ª série 4,2% para o PP e 16,1% para o SBR.

Desta forma é possível observar que, de acordo com a classificação de erros, o PP apresentou mais facilidade do que o SBR no que tange à sintaxe e as técnicas. Por

sua vez o SBR apresentou mais facilidade do que o PP nas expressões aritméticas e expressões lógicas. Quanto à lógica de implementação e o fluxo não foi possível observar padrão entre os grupos, desta forma, não é possível indicar qual dos paradigmas apresentou mais facilidade.

Utilizando os dados dos questionários, considerando a impressão dos estudantes que participaram do experimento, é possível observar que a motivação, a clareza das explicações, a clareza da apostila e a utilidade dos exemplos é padrão entre os grupos e os paradigmas.

Por sua vez, as ferramentas utilizadas para o PP foram melhores avaliadas em comparação com as ferramentas utilizadas para o SBR. Esta avaliação é padrão em todos os grupos.

Quanto à facilidade do exercício final, embora este tenha sido o mesmo em ambos os paradigmas, houve diferença na avaliação realizada pelos estudantes. Para o grupo 1A o exercício foi mais fácil no SBR do que no PP. Inversamente para o grupo 1B e para o grupo da 2ª série, o exercício foi mais fácil no PP do que no SBR.

Quanto ao entendimento do código o PP foi melhor avaliado por todos os grupos. A mesma avaliação foi feita quanto à facilidade de programação, reservando um maior equilíbrio para o grupo 1A.

## 7. CONCLUSÃO

A educação na área de ciências da computação é necessária inclusive para suprir a necessidade do mercado através de profissionais qualificados. Cada vez mais, o ensino de programação é fundamental para cursos da área de ciência da computação e afins (ACM/IEEE-CS Joint Task Force on Computing Curricula, 2013; SBC, 2005).

Entretanto, esta disciplina é de difícil compreensão pelos alunos e, às vezes, de difícil ensino pelos professores (ROBINS, ROUNTREE e ROUNTREE, 2003; KUNKLE e ALLEN, 2016). Além disso, não há consenso quanto ao melhor paradigma de programação para novatos e, por consequência, quanto a melhor linguagem (VUJOŠEVIĆ-JANIČIĆ E TOŠIĆ, 2008).

Estudos a respeito de comparativos quanto a aprendizagem destes paradigmas acabam pecando em alguns aspectos, focando em apenas revisões a respeito das características dos paradigmas, sem a comprovação prática.

A hipótese apresentada no trabalho é de que é possível comparar paradigmas de programação através das atividades (i.e., erros, latência, performance de codificação etc) geradas durante a codificação de programas de computador, visando colaborar com o processo de ensino-aprendizagem de novatos em programação, aumentando assim a qualidade do artefato gerado.

Dois paradigmas de programação foram escolhidos para a comparação. Os paradigmas são o Paradigma Procedimental e o Paradigma Lógico, mais especificamente Sistemas Baseados em Regras.

Até o momento os objetivos propostos foram parcialmente atendidos. O método proposto ainda carece de ajustes, principalmente quanto ao embasamento bibliográfico a respeito da categorização de erros e o processamento das informações coletadas em plano de fundo.

A validação do método foi realizada parcialmente através do experimento aplicado com estudantes do Curso Técnico em Informática Integrado ao Ensino Médio do Instituto Federal do Paraná – IFPR, campus União da Vitória.

Durante o experimento foi possível verificar que o PP é mais fácil do que o SBR quanto à aprendizagem. Entretanto, em algumas categorizações de erros o SBR apresentou vantagens perante o PP. Este fato deve ser observado na revisão do método para a conclusão da dissertação.

A respeito da validação se o SBR pode ser utilizado para ensino dos novatos em programação é possível afirmar, mesmo que preliminarmente, que sim. Com base nos resultados obtidos durante a execução do experimento, a utilização de SBR com novatos em programação é viável, mas não necessariamente melhor que o PP.

Como próximas etapas do trabalho, é necessário fazer o levantamento bibliográfico a respeito da classificação de erros gerados durante a codificação de programas de computador, afinando a categorização realizada até o momento.

Ademais, é necessário realizar o processamento dos arquivos gerados durante o desenvolvimento dos artefatos, os quais coletaram as ações em plano de fundo.

Novos experimentos também são necessários, coletando mais dados para embasar o método proposto. Além do desenvolvimento de uma ferramenta que será engenhada e desenvolvida a luz do método proposto.

# REFERÊNCIAS


ABDULLAH, U., SAWAR, M. J., AHMED, A. Design of a Rule Based System Using Structured Query Language. 2009 Eighth IEEE International Conference on Dependable, Autonomic and Secure Computing, Chengdu, 2009, pp. 223-228.

ABES. Mercado Brasileiro de Software: panorama e tendências, 2016, Brazilian Software Market: scenario and trends, 2016 [versão para o inglês: Anselmo Gentile] - 1ª. ed. - São Paulo: ABES - Associação Brasileira das Empresas de Software, 2016.

ACM/IEEE-CS Joint Task Force on Computing Curricula. Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science, ACM, New York, NY, USA, 2013.

ALHARBI, R. F., BERRI, J., EL-MASRI, S. Ontology based clinical decision support system for diabetes diagnostic. 2015 Science and Information Conference (SAI), London, 2015, pp. 597-602.

ARAKLIOTIS, S., NIKOLOS, D. G., KALLIGEROS, E. LAWRIS: A rule-based arduino programming system for young students. 2016 5th International Conference on Modern Circuits and Systems Technologies (MOCAST), Thessaloniki, 2016, pp. 1-4.

BANASZEWSKI, R. F. Paradigma Orientado a Notificações: Avanços e Comparações. Dissertação de Mestrado, CPGEI/UTFPR. Curitiba, 2009.

BELMONTE, D. L., LINHARES, R. R., STADZISZ, P. C. SIMÃO, J. M. A new Method for Dynamic Balancing of Workload and Scalability in Multicore Systems. in IEEE Latin America Transactions, vol. 14, no. 7, pp. 3335-3344, July 2016.

BERSTEL, B., LECONTE, M. Using Constraints to Verify Properties of Rule Programs. 2010 Third International Conference on Software Testing, Verification, and Validation Workshops, 2010, 349-354.

BINI, E. M. Ensino De Programação Com Ênfase Na Solução De Problemas. Dissertação de Mestrado, Universidade Tecnológica Federal do Paraná – UTFPR. Curso de Pós-Graduação em Ensino de Ciência e Tecnologia. Ponta Grossa, 2010.

BOURQUE, P., FAIRLEY, R. E. SWEBOK: Guide to the Software Engineering Body of Knowledge, version 3.0 Edition, IEEE Computer Society, Los Alamitos, CA, 2014.

BROOKSHEAR, J. G. Computer Science: An Overview (9 ed.). Addison Wesley, 2006.

CHOPPELLA, V., KUMAR, H., MANJULA, P., VISWANATH, K. From High-School Algebra to Computing through Functional Programming. In Proceedings of the 2012 IEEE Fourth International Conference on Technology for Education (T4E '12). IEEE Computer Society, Washington, DC, USA, 180-183, 2012.

CONSELHO NACIONAL DE DESENVOLVIMENTO CIENTÍFICO E TECNOLÓGICO. Tabela de áreas do conhecimento. 2012. Disponível em: <http://www.cnpq.br/documents/10157/186158/TabeladeAreasdoConhecimento.pdf>. Acesso em: 22 nov. 2016.

COHEN, M., RITTER, F., HAYNES, S. Evaluating Design: A Formative Evaluation of Agent Development Environments Used For Teaching Rule-Based Programming. 2009.

COPPIN, Ben. Inteligência artificial. Rio de Janeiro, RJ: LTC, 2010. 636 p.

de RAADT, M., WATSON, R., TOLEMAN, M., Introductory programming languages at Australian universities at the beginning of the twenty first century. Journal of Research and Practice in Information Technology, 35(3), 163-167, 2003.

DIJK, T. A. V., KINTSCH, W. Strategies of Discourse Comprehension, Academic Press, New York, 1983.

EL-KHAYAT, G., MABROUK, T. Solving a fit maximization assignment problem using a rule based system and linear programming, The Joint International Symposium on CIE44 and IMSS’14, Istanbul, Turkey, 2014.

ESPÁK, M. Japlo: Rule-based Programming on Java. Journal of Universal Computer Science, 12(9):1177--1189, 2006.

FERREIRA, A. B. Novo Dicionário Aurélio. (5.0). Curitiba, Paraná, Brasil: Positivo, 2006.

FERREIRA, C. A. Linguagem e Compilador para o Paradigma Orientado a Notificações (PON): Avanços e Comparações. Dissertação de Mestrado, PPGCA/UTFPR. Curitiba, 2015.

FORGY, C. RETE: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem. Artificial Intelligence , 19 (1), pp. 17-37, 1982.

FRIEDMAN-HILL, E. Jess in Action: Rule Based System in Java. Greenwich, CT, USA: Manning Publications Co, 2003.

GIARRATANO, J. C. CLIPS 6.3 User’s Guide. Gary Riley: 2015.

GRANDELL, L., PELTOMÄKI, M., BACK, R. B., SALAKOSKI, T. Why complicate things?: introducing programming in high school using Python. In Proceedings of the 8th Australasian Conference on Computing Education - Volume 52 (ACE '06), Denise Tolhurst and Samuel Mann (Eds.), Vol. 52. Australian Computer Society, Inc., Darlinghurst, Australia, Australia, 71-80. 2006.

GRUMBACH, S., WANG, F. Netlog, a rule-based language for distributed programming. In Proceedings of the 12th international conference on Practical Aspects of Declarative Languages (PADL'10), Manuel Carro and Ricardo Peña (Eds.). Springer-Verlag, Berlin, Heidelberg, 2010, 88-103.

HUCH, F. Learning programming with erlang. In Proceedings of the 2007 SIGPLAN workshop on ERLANG Workshop (ERLANG '07). ACM, New York, NY, USA, 93-99, 2007.

IEEE. The 2016 Top Programming Languages. Disponível em: <http://spectrum.ieee.org/computing/software/the-2016-top-programming-languages>. Acesso em: 30 nov. 2016.

INSTITUTO FEDERAL DO PARANÁ. Campus União da Vitória. Projeto Pedagógico do Curso Técnico em Informática Integrado ao Ensino Médio. Paraná, 2014.

JANKE, E., BRUNE, P., WAGNER, S. Does outside-in teaching improve the learning of object-oriented programming?. In Proceedings of the 37th International Conference on Software Engineering - Volume 2 (ICSE '15), Vol. 2. IEEE Press, Piscataway, NJ, USA, 408-417. 2015.

KAISLER, S. Software Paradigm. John Wiley & Sons, 2005.

KHAZAEI, B. JACKSON, M. Is There Any Difference in Novice Comprehension of a Small Program Written in the Event-Driven and Object-Oriented Styles?. In Proceedings of the IEEE 2002 Symposia on Human Centric Computing Languages and Environments (HCC'02) (HCC '02). IEEE Computer Society, Washington, DC, USA. 2002.

KING, K. N. C Programming: A Modern Approach (2ª Edição), W. W. Norton & Company, ISBN: 0393979504, 2008.

KRUG, D. L. Torre de Hanói com LingPON – Paradigma Orientado a Notificações. Aplicação em Ling PON 1.0/1.5. Disciplina sobre Paradigma Orientado a Notificações (PON), CPGEI-PPGCA/UTFPR (Prof. J. M. Simão & Prof. H. Panetto [visitante CPGEI & UL-França]), Curitiba – PR, Brasil, 2016.

KRUG, D. L. Comparativo entre o aprendizado de Programação baseado na abordagem Imperativo-Procedimental e na abordagem de Sistemas Baseados em Regras. Seminário I, PPGCA/DAINF/UTFPR, 2016.

KRUG, D. L. Sistemas Baseados em Regras: uma Revisão Sistemática. Estudo Individual, PPGCA/DAINF/UTFPR, 2017.

KUHN, T. S. The Structure of Scientific Revolutions (1ª Edição). Chicago: University of Chicago, 1962.

KUHN, T. S. The Structure of Scientific Revolutions. Chicago: University of Chicago, 1970.

KUMAR, A. N. Prolog for imperative programmers. J. Comput. Sci. Coll. 17, 6 (May 2002), 167-181, 2002.

KUNKLE, W. M. ALLEN, R. B. 2016. The impact of different teaching approaches and languages on student learning of introductory programming concepts. ACM Trans. Comput. Educ. 16, 1, Article 3 (January 2016).

LAFORE, R. Object-Oriented Programming in C++ (4ª Edição). Sams, 2002.

LEE, P.-Y., CHENG, A. M. HAL: A Faster Match Algorithm. IEEE Transactions on Knowledge and Data Engineering, 14 (5), pp. 1047-1058, 2002.

LEIJTEN, M., VAN WAES, L. Keystroke Logging in Writing Research: Using Inputlog to Analyze and Visualize Writing Processes. Written Communication 30(3), 358-392, 2013.

LINHARES, R. R., SIMÃO, J. M., STADZISZ, P. C. NOCA – A Notification-Oriented Computer Architecture. in IEEE Latin America Transactions, vol. 13, no. 5, pp. 1593-1604, May 2015.

MACIOŁ, A., MACIOŁ, P., JĘDRUSIK, S., LELITO, J. The new hybrid rule-based tool to evaluate processes in manufacturing. International Journal of Advanced Manufacturing Technology 79: 1733-1745. 2015.

MACKERLE, J. A review of expert systems development tools. Engineering Computations. Vol. 6 Iss 1 pp. 2 - 17. 1989.

MANNILA, L., RAADT, M. An objective comparison of languages for teaching introductory programming. In Proceedings of the 6th Baltic Sea conference on Computing education research: Koli Calling 2006 (Baltic Sea '06). ACM, New York, NY, USA, 32-37. 2006.

MASTERMAN, M. The Nature of a Paradigm (I. Lakatos, & A. Musgrave, Eds.) Criticism and the Growth of Knowledge, pp. 59-89, 1970.

MINISTÉRIO DA EDUCAÇÃO. Secretaria de Educação Profissional e Tecnológica. Catálogo Nacional de Cursos Técnicos. Brasília, 2012.

MIRANKER, D. P. TREAT: A better Match Algorithm for AI Production Systems. Sixth National Conference on Artificial Intelligence - AAAI'87, (pp. 42-47), 1987.

MIRANKER, D. P., BRANT, D. A., LOFASO, B., GADBOIS, D. On the Performance of Lazy Matching in Production Systems. 8th National Conference on Artificial Intelligence AAAI (pp. 685-692). AAAI Press / The MIT Press, 1990.

NASA, Lyndon B. Johnson Space Center. CLIPS Basic Programming Guide, Houston, TX, 1991.

NEWELL, A., SIMON, H. A. Human Problem Solving. Englewood Cliffs, NJ, USA: Prentice-Hall, 1972.

PANESCU, D., PASCAL, C., OLAERU, R. M. A rule-based approach for a multi-robot application. 19th International Conference on System Theory, Control and Computing (ICSTCC), Cheile Gradistei, 2015, pp. 75-80.

PARK, N., LEE, H. K., JANG, J. Rule-based modeling tool for web of things applications. 2015 IEEE 5th International Conference on Consumer Electronics - Berlin (ICCE-Berlin), Berlin, 2015, pp. 515-518.

PENNINGTON, N. Comprehension strategies in programming, in: G.M. Olson, S. Sheppard, E. Soloway (Eds.), Empirical Studies of Programmers: Second Workshop, Ablex, Norwood, NJ, 1987, pp. 100–113.

PENNINGTON, N. Stimulus structures and mental representations in expert comprehension of computer programs, Cognitive Psychology 19 (1987) 295–341.

RAGONIS, N., BEN-ARI, M. A Long-Term Investigation of the Comprehension of OOP Concepts by Novices. Computer Science Education 5(3), 203–221. 2005.

RICH, Elaine; KNIGHT, Kevin. Artificial intelligence. 2nd. ed. New York: McGraw-Hill, 1991. xvii, 621 p.

ROBINS, A., ROUNTREE, J., ROUNTREE, N. Learning and teaching programming: A review and discussion. Computer Science Education, 13:2, 137-172, 2003.

RONSZCKA, A. F. Contribuições para a concepção de aplicações no Paradigma Orientado a Notificações (PON) sob o viés de padrões. Dissertação de Mestrado, CPGEI/UTFPR. Curitiba, 2012.

ROY, P. V., HARIDI, S. Concepts, Techniques, and Models of Computer Programming. MIT Press, 2004.

ROY, P. V. Programming Paradigms for Dummies: What Every Programmer Should Know. In New Computational Paradigms for Computer Music. p. 9-47. G. Assayag and A. Gerzso (eds.), IRCAM/Delatour France, 2009.

SAWAR, M. J., ABDULLAH, U., AHMED, A. Enhanced Design of a Rule Based Engine Implemented using Structured Query Language, The 2010 International Conference of Computational Intelligence and Intelligent Systems, at World Congress on Engineering, London, U.K., 30 June - 2 July, 2010. pp67-71.

SEBESTA, R. W. Concepts of Programming Languages. Addison Wesley, Boston, 2005.

SCOTT, M. L. Programming Language Pragmatics. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc, 2000.

SHIMBUICHI, M., MATSUOKA, K., TAKAMI, K. Architecture of network robot control server software written in a Rule-based language. IET 3rd International Conference on Wireless, Mobile and Multimedia Networks (ICWMNN 2010), Beijing, 2010, pp. 373-376.

SHIMOKURA, M., NAKANISHI, S., OHTA, T. Home Network Service Programs described in a Rule-based Language. International Conference on Software Engineering Advances (ICSEA 2007), Cap Esterel, 2007, pp. 62-62.

SHIMOKURA, M., NAKANISHI, S., OHTA, T. Network software architecture for a symbiotic human life with robots. 2008 7th Asia-Pacific Symposium on Information and Telecommunication Technologies, Bandos Island, 2008, pp. 1-6.

SIMÃO, J. M., STADZISZ P. C. An Agent-Oriented Inference Engine applied for Supervisory Control of Automated Manufacturing Systems. Frontiers in Artificial Intelligence and Applications ("Advances in Logic, Artificial Intelligence and Robotics" LAPTEC 2002

Edited by J. M. Abe e J. I. da Silva Filho). Vol. 85 (pp 234-241), IOS Press, Amsterdam - The Netherlands. ISSN: 0922-6389.

SIMÃO, J. M. A Contribution to the Development of a HMS Simulation Tool and Proposition of a Meta-Model for Holonic Control. Doctoral Thesis, UTFPR, CPGEI, Curitiba, Brazil, 2005.

SIMÃO, J. M., STADZISZ, P. C. Paradigma Orientado a Notificações (PON) – Uma Técnica de Composição e Execução de Software Orientado a Notificações. Pedido de Patente submetida ao INPI/Brazil (Instituto Nacional de Propriedade Industrial) em 2008 e a Agência de Inovação/UTFPR em 2007. No. INPI Provisório 015080004262. Patente submetida ao INPI. Brasil, 2008.

SIMÃO, J. M., TACLA, C. A., STADZISZ, P. C., BANASZEWSKI, R. F. Notification Oriented Paradigm (NOP) and Imperative Paradigm: A Comparative Study. Journal of Software Engineering and Applications, Vol. 5 No. 6, 2012, pp. 402-416.

SOCIEDADE BRASILEIRA DE COMPUTAÇÃO. Currículo de referência da SBC para cursos de graduação em bacharelado em ciência da computação e engenharia de computação. 2005.

SPECHT, E., REDIN, R. M., CARRO, L., LAMB, L. C., COTA, E. F., WAGNER, F. R. Analysis of the use of declarative languages for enhanced embedded system software development. In Proceedings of the 20th annual conference on Integrated circuits and systems design (SBCCI '07). ACM, New York, NY, USA, 324-329, 2007.

TISSOT, A. A. Influência da revisão de atividades executadas para melhoria da acurácia na estimativa de software utilizando planning poker. Dissertação (Mestrado em Computação Aplicada) - Universidade Tecnológica Federal do Paraná. Curitiba, 2015.

VLAS, R., ROBINSON, W. N. A Rule-Based Natural Language Technique for Requirements Discovery and Classification in Open-Source Software Development Projects. In Proceedings of the 2011 44th Hawaii International Conference on System Sciences (HICSS '11). IEEE Computer Society, Washington, DC, USA, 2011, 1-10.

VUJOŠEVIĆ-JANIČIĆ, M., TOŠIĆ, D. The Role of Programming Paradigms In The First Programming Courses, The Teaching of Mathematics, 2008, Vol. XI, 2, pp. 63–83.

WANG, W., BAÑARES-ALCÁNTARA, R., CUI, Z., WANG, Y. J., COENEN, F. An association rule-based CLIPS program for interactive prediction of MSC differentiation in vitro. 2010 International Conference on Computer Application and System Modeling (ICCASM 2010), Taiyuan, 2010, pp. V2-406-V2-410.

WATT, D. Programming Language Design Concepts. J. Willey & Sons, 2004.

WIEDENBECK, S., RAMALINGAM, V. Novice Comprehension of small programs written in the procedural and object oriented styles. International Journal of Human Computer Studies 51, 71-87, 1999.

WIEDENBECK, S., RAMALINGAM, V., SARASAMMA, S., CORRITORE, C.L. A comparison of the comprehension of object-oriented and procedural programs by novice programmers. Interacting with Computers, 11, 255–282, 1999.

YAMASHITA, T., OHTA, T., TAKAMI, K. Creating Web Services Using a Rule-Based Language. 2010 Fifth International Conference on Systems and Networks Communications, Nice, 2010, pp. 272-277.

ZACHARIAS, V. Development and Verification of Rule Based Systems - A Survey of Developers. Rule Representation, Interchange and Reasoning on the Web: International Symposium, RuleML 2008, Orlando, FL, USA, October 30-31, 2008.

## SEÇÃO COMPLEMENTAR A – REVISÃO SISTEMÁTICA DA LITERATURA

Revisão Sistemática da Literatura escrita sobre Sistemas Baseados em Regras. Estudo Individual, PPGCA/DAINF/UTFPR.

# Sistemas Baseados em Regras: uma Revisão Sistemática*

**Douglas Lusa Krug[1, 2], Jean Marcelo Simão[1], Laudelino Cordeiro Bastos[1]**

[1]Universidade Tecnológica Federal do Paraná – UTFPR
Departamento Acadêmico de Informática – DAINF
Programa de Pós-graduação em Computação Aplicada – PPGCA
Av. Sete de Setembro, 3165 – Curitiba – PR – Brasil

[2]Instituto Federal do Paraná – IFPR
Av. Paula Freitas s/n – 84.600-000 – União da Vitória – PR – Brasil

***Abstract.*** *Rules Based Systems (RBSs) have been presented as an alternative programming way for software in several areas, such as insurance, finance, medicine, games, and biology. There are some tools and languages available to develop programs using RBS. Interest in RBS is increasing, but there is little data and research on this topic. In order to answer some questions about RBSs, which tools exists, and which solutions can be implemented, this work presents a Systematic Literature Review (SLR). Through this SLR, in agreement with the objectives and the research questions, some tools used in RBS and some types of solutions that can be implemented with this form of programming were listed. However, although some advantages of using RBSs have been shown, it is not possible to perform deeper analysis, just as it is not possible to find in the literature metrics for development with RBSs.*

***Resumo.*** *Os Sistemas Baseados em Regras (SBRs) vêm se apresentando como uma alternativa de programação para software em diversas áreas, tais como: seguros, finanças, medicina, jogos e biologia. Existem algumas ferramentas e linguagens disponíveis para desenvolver programas utilizando SBR. O interesse por SBRs está aumentando, porém existem poucos dados e pesquisas a respeito. Visando responder algumas questões relativas aos SBRs, como quais ferramentas existem e quais soluções podem ser implementadas, este trabalho apresenta uma Revisão Sistemática da Literatura (RSL). Através desta RSL, em acordo com os objetivos e questões de pesquisa, foram listadas algumas ferramentas utilizadas no desenvolvimento e alguns tipos de soluções que podem ser implementados com esta forma de programação. Entretanto, apesar de algumas vantagens de utilização de SBRs terem sido mostradas, não foi possível realizar uma análise mais profunda, da mesma forma que não foi possível encontrar, na literatura, métricas de comparação para desenvolvimento com SBRs.*

## 1. Introdução

Desde o advento do computador a sociedade vem passando por transformações em seus processos, as tarefas que antes eram realizadas com grande esforço hoje são realizadas

---

* Documento gerado para relatar o resultado da disciplina de Estudo Individual da 3ª fase de 2016 realizada para o Programa de Pós-graduação em Computação Aplicada (PPGCA) do DAINF/UTFPR – Curitiba Central, modalidade Mestrado Profissional.

de forma automática. Parte destas transformações ocorre devido a evolução da área de desenvolvimento de software, que acontece por meio do estudo de seus paradigmas e linguagens de programação, bem como pela proposição de novos paradigmas, linguagens, técnicas e ferramentas. Dentre estas técnicas estão os Sistemas Baseados em Regras (SBRs), que vêm se apresentando como uma alternativa de programação para software em diversas áreas, tais como: seguros, finanças, medicina, jogos e biologia (RICH e KNIGHT, 1991; COPPIN, 2010; RUSSELL e NORVIG, 2003).

Os SBRs são uma forma de programação pertencente ao Paradigma Lógico (PL) que, conforme Coppin (2010), usam regras para fornecer recomendações ou diagnósticos, ou ainda, para determinar uma linha de ação em uma situação particular ou para solucionar um problema específico.

O PL é um paradigma de programação pertencente ao grupo dos Paradigmas Declarativos (PD). Detalhes a respeito de cada paradigma, suas características e divisões não estão no escopo deste trabalho, estas informações podem ser verificadas no trabalho escrito por Krug (2016). Outrossim, revisões mais profundas de paradigmas encontram-se em Roy e Haridi (2004), Watt (2004) e Roy (2009).

Isto considerado, retoma-se o tema de SBR. Conforme Zacharias (2008), existe um renovado e crescente interesse a respeito de SBRs e seu desenvolvimento. Porém, ao mesmo tempo, existem poucos dados e trabalhos descrevendo como SBRs são utilizados, seu desenvolvimento e os desafios enfrentados por quem os utiliza.

Berstel e Lecont (2010) afirmam que SBRs vêm ganhando o interesse da indústria, pois demonstram uma forma de separar as regras de negócios das aplicações das entidades tratadas por ela. Desta forma, com esse desacoplamento, baixa-se o custo das frequentes alterações, causadas por fatores como atualizações de legislação ou competitividade do negócio.

Outrossim, este trabalho está sendo escrito como parte de um projeto maior, que visa corroborar a hipótese de que a utilização do Paradigma Lógico, especificamente os SBRs, como primeiro paradigma para novatos em programação, apresenta melhores resultados de aprendizagem do que a utilização de Paradigma Imperativo Procedimental (KRUG, 2016)[1].

Em linha com a hipótese, a suposta facilidade de utilização de SBR é relatada por Shimokura, Nakanishi e Ohta (2008), que descreve brevemente um experimento com jovens estudantes, entre 16 e 17 anos, sem experiência com programação, que tiveram uma breve explicação sobre o desenvolvimento com a linguagem de SBR nominada STAR/ESTR e apresentaram grande desempenho no desenvolvimento com esta linguagem.

Da mesma maneira, Arakliotis, Nikolos e Kalligeros (2016) descrevem brevemente a escolha de SBR para ensino de programação para estudantes primários. Nesse trabalho relatam que escolheram SBR devido a esta forma de programação aparentar ser mais fácil para o aprendizado.

Neste contexto de estudos no tocante à SBR, o presente trabalho apresenta uma Revisão Sistemática da Literatura a respeito dos SBRs.

Para nortear a execução desta revisão os objetivos a seguir foram estabelecidos:

---

1 Pertinente ressaltar que, embora o trabalho tenha o viés do aprendizado, ele visa melhorar a qualidade do software. A escolha para utilizar estudantes e aprendizado deve-se ao fato do material humano disponível para os experimentos.

Objetivo geral: **Identificar e descrever as ferramentas utilizadas para desenvolvimento com SBR, métricas para comparação de desenvolvimento com SBR e vantagens de utilização do desenvolvimento utilizando SBR, descrevendo também tipos de soluções implementadas com utilização de SBR.**

Por sua vez, os objetivos específicos são:

**a) Listar ferramentas utilizadas para desenvolvimento com SBR relatadas em trabalhos científicos que descrevam a implementação de uma solução, identificando o tipo de solução implementada;**

**b) Descrever vantagens e desvantagens de utilização de SBR em desenvolvimento de software;**

**c) Identificar métricas para comparação de desenvolvimento com SBR.**

Nas próximas seções deste trabalho são apresentados os trabalhos relacionados, uma breve contextualização sobre SBRs, o método utilizado nesta revisão, os resultados oriundos desta e, por fim, a conclusão.

## 2. Trabalhos relacionados

Durante a procura por trabalhos para esta revisão, alguns artigos encontrados apresentam trabalhos similares a este. Estes trabalhos são brevemente relatados nesta seção.

Zacharias (2008) descreve em seu trabalho uma pesquisa com desenvolvedores de SBR sobre métodos e ferramentas utilizadas no desenvolvimento de SBR, com foco na validação e depuração. No decorrer de seu trabalho ele destaca alguns pontos importantes de preferência dos desenvolvedores, porém, o que chama a atenção é um questionário de comparação entre SBR e Paradigma Imperativo.

Através do resultado deste questionário é possível observar que, segundo as respostas, o SBR é superior ao Paradigma Imperativo nos aspectos a seguir: facilidade de alteração e manutenção; facilidade de criação; facilidade de reuso; facilidade de entendimento; e confiabilidade. Em contrapartida, a programação convencional, como o Paradigma Imperativo foi chamado no texto, mostra-se superior em outros aspectos: performance em tempo de execução; facilidade de depuração; e suporte à ferramenta de desenvolvimento.

Outros dois trabalhos encontrados focam em comparação de linguagens baseadas em regras voltadas para Web Semântica. Neste contexto, o trabalho escrito por Rivolli, Orlando e Moreira (2011) apresenta algumas técnicas e estratégias desenvolvidas para aprimorar ferramentas utilizadas no desenvolvimento, utilizando *Semantic Web Rule Language* (SWRL) uma linguagem baseada em regras para Web Semântica. O artigo relata um trabalho em andamento, como resultado o trabalho apresenta um ponto de partida para melhorias na linguagem SWRL.

Por sua vez, o trabalho escrito por Rattanasawad et al. (2013) apresenta uma comparação de linguagens e máquinas de inferência utilizados para Web Semântica. O intuito do trabalho é servir como guia para desenvolvedores e pesquisadores na escolha de uma ferramenta que case com os requerimentos necessários. Como resultado, algumas linguagens utilizadas para Web Semântica foram comparadas seguindo critérios estabelecidos no próprio artigo.

Na próxima seção será dada uma breve contextualização a respeito de Sistemas

Baseados em Regras (SBR).

## 3. Sistemas Baseados em Regras – SBR

Os Sistemas Baseados em Regras (SBR) foram baseados nos trabalhos de Newell e Simon (1972), no qual pretendiam simular o raciocínio humano através de um mecanismo de inferência que relaciona fatos e regras. Eles observaram que muitos dos problemas que o ser humano resolve podem ser expressos em regras lógico-causais. Também foi possível constatar que o cérebro é estimulado a partir de entradas sensórias, i.e. fatos.

Com isso, considerou-se que há dois tipos de bases de conhecimento. Há a base de fatos e há também a base de regras para relacionar fatos. Esses podem ser armazenados em entidades chamadas "elementos da base de fatos" e aqueles em regras de uma "base de regras". Os elementos da base de fatos são usados para armazenar temporariamente o conhecimento necessário para a resolução de problemas (RICH e KNIGHT, 1991). Os estímulos (fatos) ativam algumas regras, da base de regras, com o intuito de produzir uma resposta apropriada para um determinado problema (NEWELL e SIMON, 1972).

Assim, a arquitetura de um SBR, baseado nas observações de Newell e Simon (1972), consiste em uma memória para armazenar as regras (Base de Regras), uma memória para armazenar os fatos (Base de Fatos) e a Máquina de Inferência (MI). Em suma, a MI é um elemento processador capaz de inferir sobre estas duas bases. A ilustração desta arquitetura é apresentada na Figura 1 (FRIEDMAN-HILL, 2003).

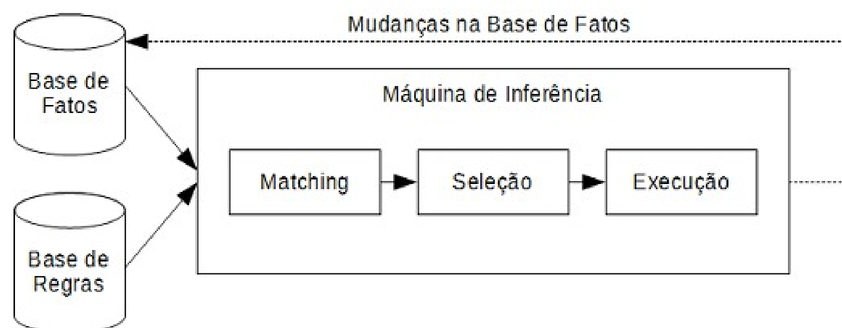



**Figura 1. Arquitetura de um Sistema Baseado em Regras – Adaptado de FRIEDMAN-HILL, 2003.**

A Base de Fatos armazena os fatos do sistema, enquanto que a Base de Regras armazena o conjunto de regras que contém o conhecimento ou decisões de controle do sistema. Tanto a Base de Fatos quanto a Base de Regras são separadas da Máquina de Inferência, que é a responsável por comparar regras e fatos durante os ciclos de inferência (BANASZEWSKI, 2009; FRIEDMAN-HILL, 2003).

Cada ciclo de inferência de um SBR é composto por três fases:

***Matching* ou casamento**: compara os fatos em relação às regras visando ativá-las. As regras ativadas são armazenadas, de forma desordenada, em um repositório chamado *Conjunto de Conflito*.

**Seleção**: ordena as regras ativadas de acordo com alguma estratégia de resolução de conflito, podendo ser alguma estratégia baseada na prioridade das regras ou na recentidade dos fatos que ativaram as regras, formando um conjunto ordenado de regras

chamado *Agenda*.

**Execução**: seleciona a primeira regra da *Agenda* e executa a sua ação. Durante a execução da ação, a regra pode adicionar novos elementos na Base de Fatos ou ainda solicitar um serviço externo.

A fase de *Matching* é a que mais compromete a performance dos SBRs quanto ao desempenho, principalmente para aqueles SBRs que não tem um mecanismo de inferência suficientemente eficiente. Este fato ocorria principalmente nos primeiros SBRs, nos quais a perda de desempenho era ocasionada devido à avaliação redundante entre fatos e regras, sendo que muitos dos testes realizados em um ciclo de *matching* apresentam os mesmos resultados dos ciclos antecedentes (FRIEDMAN-HILL, 2003; BANASZEWSKI, 2009).

Algumas soluções para evitar a perda de desempenho nesta fase foram propostas, armazenando os estados já avaliados em ciclos anteriores e, portanto, realizando as comparações somente das regras contra o estado dos elementos da Base de Fatos atualizados recentemente. Estas soluções foram implementadas em algoritmos como o RETE (FORGY, 1982), o TREAT (MIRANKER, 1987), o LEAPS (MIRANKER et al., 1990) e o HAL (LEE e CHENG, 2002).

A título de exemplo do funcionamento de um sistema escrito em SBR, um pequeno código em CLIPS é apresentado na Figura 2. Neste programa existem duas regras que simulam a reação que um robô deve ter ao ver a cor da luz indicada em um semáforo. Caso a cor da luz seja vermelha ele deve parar, caso a cor da luz seja verde ele deve andar.

Neste exemplo, as regras ficam carregadas em memória aguardando que um fato seja criado. A partir do momento da criação de um fato, a Máquina de Inferência executa a fase de casamento buscando as regras onde é possível combinar o fato inserido. Caso uma das regras seja satisfeita, ela é alocada na agenda e a ordem de sua execução é indicada. Após isso a ação é então executada.

```
1  (defrule sinal-vermelho
2       (cor-luz vermelha)
3  =>
4       (printout t "Pare!" crlf)
5  )
6
7  (defrule sinal-verde
8       (cor-luz verde)
9  =>
10      (printout t "Ande!" crlf)
11 )
```

**Figura 2. Exemplo de programa em CLIPS – Adaptado de GIARRATANO, 2015.**

Em tempo, existe uma divisão da forma como os SBRs trabalham em dois modelos de raciocínio: (1) Sistemas de Dedução (*Backward Chaining*) que inicia com a hipótese e trabalha verificando se existem fatos disponíveis que suportam a hipótese; e (2) Sistemas de Produção (*Forward Chaining*) que utiliza os fatos e usa as regras para extrair mais dados até atingir o objetivo (RICH e KNIGHT, 1991).

Rich e Knight (1991) indicam a utilização de *backward chaining* quando a resolução do problema é direcionada ao objetivo. Por sua vez, indicam a utilização de *forward chaining* quando é necessário solucionar o problema de acordo com os fatos recebidos de fora.

Conforme Coppin (2010), com *backward chaining* parte-se de uma conclusão (hipótese) e tem-se por objetivo mostrar como aquela conclusão pode ser alcançada a partir de regras e fatos da base de dados. Com o *forward chaining* o sistema parte de um conjunto de fatos e de um conjunto de regras e tenta encontrar um meio de usar tais regras e fatos para deduzir uma conclusão ou traçar uma linha de ação. O foco deste artigo está em soluções que utilizam *forward chaining,* os quais são chamados de Sistemas de Produção (SP) ou, em inglês, *Production Systems* (PS).

Ainda, para o desenvolvimento de SBRs algumas ferramentas foram criadas, estas ferramentas são chamadas de *shells,* sendo compostas por uma linguagem, por um mecanismo de inferência eficiente e outras ferramentas úteis, como editor gráfico ou textual, gerenciador de arquivos e um depurador.

Pertinente mencionar que por vezes SBRs são também chamados de Sistemas Especialistas ou, em inglês, *Expert Systems*. Um Sistema Especialista trabalha em um domínio específico de aplicação, com conhecimento causal e factual substanciais (SIMÃO e STADZISZ, 2002). Usualmente, o conhecimento causal é obtido por especialistas humanos, enquanto o conhecimento factual pode vir de fontes diversas, como a observação humana ou automática, por exemplo, outros sistemas ou sensores. (SIMÃO, 2005).

Um Sistema Especialista é um sistema interativo que auxilia o usuário com conhecimento de uma área específica. Os elementos básicos de um Sistema Especialista são a base de conhecimento, que contém o conhecimento específico, e a máquina de inferência, que resolve o problema interpretando a base de conhecimento (MACKERLE, 1989).

Isto considerado, a próxima seção apresenta o método utilizado na revisão no tocante a SBR e suas ferramentas.

## 4. Método

Para realizar esta Revisão Sistemática da Literatura, utilizou-se como base o guia escrito por Kitchenham e Charters (2007). De acordo com o guia utilizado como referência, uma Revisão Sistemática da Literatura é composta pelas fases de planejamento, realização da revisão e relato da revisão.

Para nortear a revisão, em linha com os objetivos, as questões a seguir foram elaboradas:

*QP1*: Quais linguagens e/ou ferramentas utilizam SP-SBRs?

*QP2*: Em que tipos de soluções pode-se utilizar SP-SBRs?

*QP3*: Quais são as vantagens e desvantagens de utilização de SP-SBRs perante os demais modelos de programação?

*QP4*: Quais elementos podem ser utilizados como métrica para desenvolvimento utilizando SP-SBR?

De acordo com os objetivos e as questões de pesquisa, foram estabelecidos os termos para serem utilizados como filtro para encontrar os artigos que foram analisados. Os termos escolhidos foram: *rule based*, *production system*, *programming*, *language*, *solution*, *comparative* e *tool*. Os termos escolhidos foram utilizados em combinações listadas na Tabela 1.

**Tabela 1. Combinação de termos utilizados como filtro.**

| Combinação de Termos |
|---|
| *“rule based” and (programming or language or solution or comparative or tool)* |
| *“production system” and (programming or language or solution or comparative or tool)* |

As bases escolhidas para fazer a busca foram: IEEE Xplore[2], Scopus[3], ACM Digital Library[4] e Science Direct[5]. As buscas de artigos foram realizadas entre os dias 02/01/2017 e 27/01/2017. Foram considerados artigos publicados no período entre 2006 e 2017.

Para seleção dos artigos buscados nas bases supracitadas foram aplicados critérios de inclusão e exclusão listados a seguir, sendo aplicados inicialmente no título e, então, no resumo, introdução e conclusão. Os trabalhos considerados atenderam aos critérios de inclusão (CI), observando também os critérios de exclusão (CE):

*CI1* – Foram incluídos artigos que apresentaram a utilização de SBRs;

*CI2* – Foram incluídos artigos que apresentaram relato de implementação de uma solução com SBR ou ainda a proposição de uma nova ferramenta/linguagem;

*CE1* – Foram excluídos artigos que não estavam totalmente disponíveis.

*CE2* – Foram excluídos artigos que não estavam no idioma Português ou Inglês.

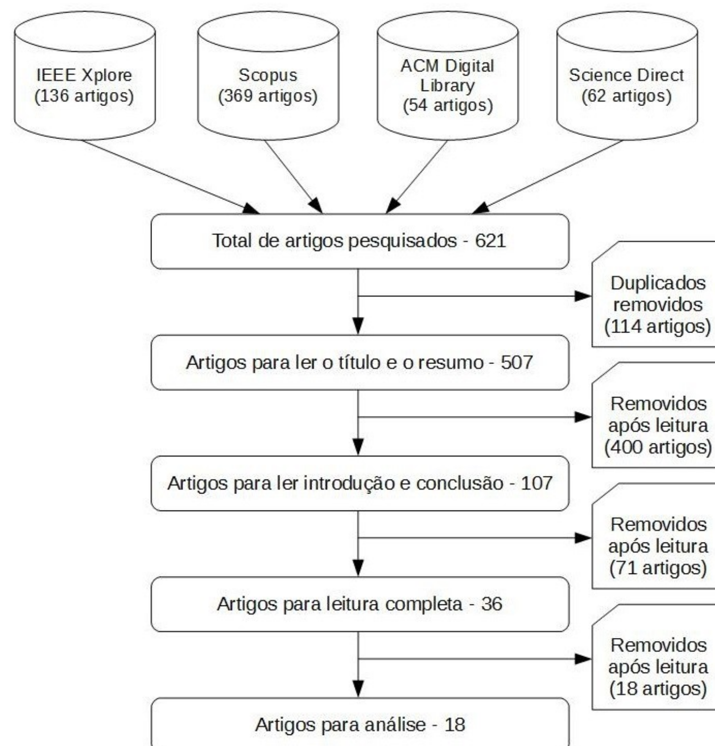



**Figura 3. Triagem de artigos – Inspirado de MOHER et al., 2009.**

2 http://ieeexplore.ieee.org/Xplore/home.jsp
3 https://www.scopus.com/
4 http://dl.acm.org/
5 http://www.sciencedirect.com/

Através de busca realizada nas bases citadas e utilizando os termos relacionados na Tabela 1, foram encontrados 621 artigos. Dentre eles haviam 114 artigos duplicados, após a retirada destes, restaram 507 artigos.

Aplicando os critérios de inclusão e exclusão no título e resumo foram retirados 400 artigos, desta forma, restaram 107 artigos para uma análise mais aprofundada. Após analisar introdução e conclusão restaram 36 artigos. Durante a leitura completa dos artigos foi verificado que 18 destes não atendiam aos critérios, desta forma, restaram 18 artigos para compor o resultado da revisão. Na Figura 3 está ilustrado o fluxo de triagem de artigos. Os artigos selecionados para análise estão listados na Tabela 2.

**Tabela 2. Artigos selecionados para análise.**

| Autor | Nome | Ano |
|---|---|---|
| PANESCU, D., PASCAL, C., OLAERU, R. M | A rule-based approach for a multi-robot application | 2015 |
| VLAS, R., ROBINSON, W. N. | A Rule-Based Natural Language Technique for Requirements Discovery and Classification in Open-Source Software Development Projects | 2011 |
| HATZILYGEROUDIS, I., KOVAS, K. | A tool for automatic creation of rule-based expert systems with CFs | 2010 |
| WANG, W. et al. | An association rule-based CLIPS program for interactive prediction of MSC differentiation in vitro | 2010 |
| SHIMBUICHI, M., MATSUOKA, K, TAKAMI, K. | Architecture of network robot control server software written in a Rule-based language | 2010 |
| YAMASHITA, T., OHTA, T., TAKAMI, K. | Creating Web Services Using a Rule-Based Language | 2010 |
| ABDULLAH, U., SAWAR, M. J., AHMED, A. | Design of a Rule Based System Using Structured Query Language | 2009 |
| SAWAR, M. J., ABDULLAH, U., AHMED, A. | Enhanced design of a rule based engine implemented using structured query language | 2010 |
| COHEN, M., RITTER, F., HAYNES, S. | Evaluating design: A formative evaluation of agent development environments used for teaching rule-Based programming | 2009 |
| SHIMOKURA, M., NAKANISHI, S., OHTA, T. | Home Network Service Programs described in a Rule-based Language | 2007 |
| ESPÁK, M. | Japlo: Rule-based programming on Java | 2006 |
| ARAKLIOTIS, S., NIKOLOS, D. G., KALLIGEROS, E. | LAWRIS: A rule-based arduino programming system for young students | 2016 |

| | | |
|---|---|---|
| GRUMBACH, S., WANG, F. | Netlog, a rule-based language for distributed programming | 2010 |
| SHIMOKURA, M., NAKANISHI, S., OHTA, T. | Network software architecture for a symbiotic human life with robots | 2008 |
| ALHARBI, R. F., BERRI, J., EL-MASRI, S. | Ontology based clinical decision support system for diabetes diagnostic | 2015 |
| PARK, N., LEE, H.-K., JANG, J. | Rule-based modeling tool for web of things applications | 2015 |
| EL-KHAYAT, G., MABROUK, T. | Solving a fit maximization assignment problem using a rule based system and linear programming | 2014 |
| MACIOŁ, A. et al. | The new hybrid rule-based tool to evaluate processes in manufacturing | 2015 |

Na próxima seção são apresentados os resultados encontrados nos artigos selecionados de acordo com as questões de pesquisa apresentadas anteriormente.

## 5. Resultados e discussões

Por meio do estudo dos trabalhos selecionados algumas informações foram analisadas e serão descritas a seguir para responder as perguntas levantadas na seção anterior.

Para responder a primeira questão "Quais linguagens e/ou ferramentas utilizam SP-SBRs?", a Tabela 3 lista as linguagens e ferramentas que utilizam SP-SBR encontradas nos trabalhos selecionados.

**Tabela 3. Linguagens e ferramentas que utilizam SBR.**

| ***Shell* / Linguagem** | **Trabalhos** |
|---|---|
| CLIPS[6] | PANESCU, D., PASCAL, C., OLAERU, R. M (2015); HATZILYGEROUDIS, I., KOVAS, K. (2010); WANG, W. et al. (2010) |
| Drools[7] | PARK, N., LEE, H.-K., JANG, J. (2015) |
| JAPE[8] | VLAS, R., ROBINSON, W. N. (2011) |
| JAPLO[9] | ESPÁK, M. (2006) |
| Jess[10] | COHEN, M., RITTER, F., HAYNES, S. (2009); ALHARBI, R. F., BERRI, J., EL-MASRI, S. (2015) |
| LAWRIS[11] | ARAKLIOTIS, S., NIKOLOS, D. G., KALLIGEROS, E. (2016) |

6 http://www.clipsrules.net/

7 https://www.drools.org/

8 https://gate.ac.uk/sale/tao/splitch8.html

9 http://dx.doi.org/10.3217/jucs-012-09-1177

10 http://www.jessrules.com/

| | |
|---|---|
| Netlog[12] | GRUMBACH, S., WANG, F. (2010) |
| REBIT[13] | MACIOŁ, A. et al. (2015) |
| SQL[14] | ABDULLAH, U., SAWAR, M. J., AHMED, A. (2009);<br>SAWAR, M. B., ABDULLAH, U., AHMED, A. (2010) |
| STAR/ESTR[15] | SHIMBUICHI, M., MATSUOKA, K, TAKAMI, K. (2010);<br>YAMASHITA, T., OHTA, T., TAKAMI, K. (2010);<br>SHIMOKURA, M., NAKANISHI, S., OHTA, T. (2007);<br>SHIMOKURA, M., NAKANISHI, S., OHTA, T. (2008) |

Os trabalhos considerados para análise relatam a utilização de diferentes *shells* e linguagens para implementação de SBRs. Algumas destas *shells* são mais conhecidas (CLIPS, Drools e Jess) e outras propostas pelos próprios trabalhos (JAPLO, LAWRIS e Netlog). A seguir será relatado um pouco de cada uma delas.

Conforme Giarratano (2015), CLIPS é um acrônimo para *C Language Integrated Production System* e foi desenvolvido em 1986 pela NASA – USA. CLIPS foi projetada para facilitar o desenvolvimento de software através da modelagem do modelo de conhecimento humano e experiência.

Ainda segundo Giarratano (2015), CLIPS é considerada uma ferramenta para sistemas especialistas pois é um ambiente completo para desenvolvimento destes sistemas, incluindo editor integrado e ferramentas para depuração. Ela contém todos os elementos básicos de um sistema especialista: base de fatos, base de conhecimento e máquina de inferência.

Drools, conforme informações de sua documentação[16], é uma *rule engine* que é baseada em regras e é utilizada para produzir sistemas especialistas, mais corretamente classificado como Sistema de Produção. Drools é um BRMS (*Business Rule Management Systems*), baseado em Java e utiliza o algoritmo Rete.

Conforme Vlas e Robinson (2011) JAPE (*Java Annotation Pattern Engine*) é uma *engine* baseada em regras que suporta a linguagem Java e expressões regulares. JAPE faz parte de uma solução de processamento de texto chamada GATE (*General Architecture for Text Engineering*).

JAPLO, segundo Espák (2006), é uma extensão da linguagem Java que foi criada para permitir a criação de regras, estilo Paradigma Lógico, por programadores familiarizados com Java. JAPLO permite a utilização de regras semelhantes ao Prolog[17] com sintaxe semelhante à da linguagem Java. O nome é uma junção de Java com Prolog.

Conforme Friedman-Hill (2003), Jess é uma *shell*, desenvolvida em Java, baseada em regras desenvolvida pela Sandia National Laboratories no final da década de

12 https://doi.org/10.1007/978-3-642-11503-5_9

11 https://doi.org/10.1109/MOCAST.2016.7495150

13 https://doi.org/10.1007/s00170-015-6860-5

14 https://en.wikipedia.org/wiki/SQL

15 https://doi.org/10.1007/978-1-4471-0287-8_12

16 https://docs.jboss.org/drools/release/5.3.0.Final/drools-expert-docs/html/

17 https://en.wikipedia.org/wiki/Prolog

1990. Embora Jess apresente grandes semelhanças com CLIPS, elas foram desenvolvidas por grupos diferentes de pessoas. Jess assemelha-se em alguns pontos com CLIPS, porém é baseada em Java e permite integração com a linguagem.

Segundo Arakliotis, Nikolos e Kalligeros (2016), LAWRIS (*Learning-Arduino-With-Rules Introductory System*) é uma plataforma de aprendizado baseada na Web que permite desenvolvimento de programas para Arduíno[18] utilizando regras. Esta plataforma foi desenvolvida para permitir que estudantes do ensino fundamental pudessem ter contato com desenvolvimento de Arduíno de uma maneira simples, o desenvolvimento é baseado em regras e utiliza um editor gráfico com blocos estilo *drag and drop*.

Grumbach e Wang (2010) propuseram a linguagem Netlog, uma linguagem declarativa, baseada em regras, para ser utilizada com aplicações distribuídas, como protocolos de comunicação e aplicações *peer-to-peer.*

Macioł et al. (2015) apresentam em seu trabalho a *shell* REBIT (Business and Technological Rules Management System) desenvolvido pela *Faculty of Management* da *AGH University of Science and Technology* na Cracóvia. Conforme os autores, a *shell* REBIT é composta por uma linguagem e um algoritmo de otimização próprios e tem em sua essência uma plataforma para a troca de informações não distorcida entre o sistema baseado em regras e modelos de simulação.

*Structured Query Language* (SQL) é uma linguagem declarativa utilizada diretamente em bancos de dados. Algumas aplicações utilizam SQL baseado em regras, conforme relatam Abdullah, Sawar e Ahmed (2009) e Sawar, Abdullah e Ahmed (2010).

Conforme Yamashita, Ohta e Takami (2010), STAR (*SofTware Architecture using Rule-based language*) foi proposta com o intuito de desenvolver softwares em menor tempo. STAR utiliza uma linguagem baseada em regras chamada ESTR (*Enhanced State Transition Rule*), que apresenta uma especificação simples e um baixo volume de programação.

Com base no que foi descrito anteriormente foi possível verificar algumas linguagens e ferramentas que utilizam SBR. Algumas destas ferramentas são consolidadas e de ampla utilização, como CLIPS, Drools e Jess, e outras foram propostas pelos próprios trabalhos.

Em tempo, o intuito deste trabalho não é classificar a melhor ferramenta para construção de SBR. Ademais, os artigos selecionados não apresentam argumentos suficientes para esta classificação.

Quanto à segunda questão levantada neste trabalho “Em que tipos de soluções pode-se utilizar SP-SBRs?” é possível observar uma diversidade de aplicações de SBR relatadas nos trabalhos encontrados.

Panescu, Pascal e Olaeru (2015) relatam a utilização de SBR para controlar uma aplicação multi robôs em processos de manufatura, na qual os robôs trabalham com áreas de armazenamento individual, porém, utilizando uma área de montagem em comum. Devido a isso, o os robôs devem ser sincronizados para as tarefas de montagem e empilhamento. Comparando a solução construída com SBR e uma solução sequencial, na qual os robôs não trabalham de forma coordenada e paralela, é possível observar um ganho de performance na solução implementada com SBR.

---

18 https://www.arduino.cc/

Uma solução utilizando SBR para controle de robôs também é apresentada por Shimbuichi, Matsuoka e Takami (2010), na qual relatam uma aplicação para controle de mais de um robô, de modelos distintos, coordenados para solucionar uma mesma ação.

Por sua vez, Shimokura, Nakanishi e Ohta (2008) propõe um sistema desenvolvido com os preceitos de SBR para promover uma convivência simbiótica entre humanos e robôs, sendo que, através da utilização de regras é possível facilitar a descrição das tarefas que devem ser executadas pelos robôs de maneira simples e segura.

O controle de robôs pela rede e também uma rede doméstica através da utilização de SBR é apresentada por Shimokura, Nakanishi e Ohta (2007).

Vlas e Robinson (2011) descrevem a utilização de SBR para descobrir requisitos de software *open-source*, os quais normalmente são descritos em meios não tradicionais de documentação de requisitos, como fóruns, chats e e-mails. Estes requisitos estão escritos em linguagem natural e a utilização de SBR facilita a forma como os requisitos são encontrados e classificados.

A utilização de SBRs em software voltado a medicina é comum, principalmente quanto ao auxílio no diagnóstico e melhor tratamento de doenças, um dos pioneiros é o MYCIN[19], criado nos primeiros anos da década de 1970.

Pesquisas recentes também trazem implementações de SBRs voltados para medicina. Wang et al. (2010) descrevem a utilização de SBR para predição e diferenciação de destino para células-tronco mesenquimais, sendo que a base de conhecimento deste sistema para composição das regras foi retirada de estudos anteriores.

No mesmo campo da medicina, agora voltado ao auxílio de diagnóstico, Alharbi, Berri e El-Masri (2015) propuseram um sistema de apoio a decisão clínica para diagnóstico de diabetes utilizando SBR. Este sistema considera informações do paciente, sintomas, sinais, fatores de risco e testes de laboratório para sugerir um tratamento de acordo com o tipo de diabetes do paciente.

Ainda na área médica, mas agora com outra função, Abdullah, Sawar e Ahmed (2009) e Sawar, Abdullah e Ahmed (2010) propuseram a utilização de SBR para processar os documentos necessários para reembolso de procedimentos médicos.

Dentre os artigos selecionados, dois relatam uma breve experiência de utilização de SBR para o ensino de programação, Cohen, Ritter e Haynes (2009) para ensino de Inteligência Artificial e Arakliotis, Nikolos e Kalligeros (2016) para ensino de programação para estudantes do ensino fundamental, utilizando Arduíno e blocos com programação baseada em regras.

Para expressar a amplitude de soluções que podem ser criadas utilizando SBR, dentre os trabalhos selecionados, alguns demonstram outros tipos de soluções. O trabalho escrito por Yamashita, Ohta e Takami (2010) apresenta a utilização de SBR para criar Web Services. Park, Lee e Jang (2015) apresentam a utilização de SBR para uma implementação de *Web of Things* (WoT). Ainda, El-Khayat e Mabrouk (2014) apresentam a utilização de SBRs para seleção/atribuição de candidatos a cursos de ensino superior de acordo com suas habilidades. Finalizando, Macioł et al. (2015) apresentam a utilização de SBR para tomada de decisões em processos de manufatura.

19 https://en.wikipedia.org/wiki/Mycin

Observando os trabalhos é possível notar uma ampla gama de soluções que podem ser construídas com a utilização de SBRs. Neste sentido, Zacharias (2008) descreve algumas áreas nas quais soluções com SBRs estão sendo utilizadas, sendo elas, seguro, finanças, saúde, biologia, jogos para computadores, viagens e engenharia de software.

Embora as informações encontradas nos trabalhos selecionados apresentem valiosa contribuição à utilização de SBR, não foi apresentado material suficiente para responder as duas últimas questões de pesquisa: *QP3*: Quais são as vantagens e desvantagens de utilização de SP-SBRs perante os demais modelos de programação?; *QP4*: Quais elementos podem ser utilizados como métrica para desenvolvimento utilizando SP-SBR?

De qualquer forma, é pertinente citar alguns pontos relacionados à questão de pesquisa 3. No trabalho escrito por Panescu, Pascal e Olaeru (2015) é citado o ganho de performance com a utilização de uma solução SBR em comparação à solução sequencial. Neste experimento a solução com SBR utilizou metade do tempo utilizado pela solução sequencial para realizar a tarefa.

Ainda, os trabalhos escritos por Shimokura, Nakanishi e Ohta (2008) e Arakliotis, Nikolos e Kalligeros (2016) relatam experimentos com facilidade de utilização de SBR como uma vantagem.

Por fim, para a questão de pesquisa 4, entende-se que é necessário um trabalho especificamente voltado para responder esta questão, ampliando um pouco os filtros utilizados para seleção dos artigos.

## 6. Conclusão

Os Sistemas Baseados em Regras (SBR) são uma forma de programação que, conforme Coppin (2010), utilizam regras para fornecer recomendações ou diagnósticos, ou ainda, para determinar uma linha de ação em uma situação particular ou para solucionar um problema específico.

A aplicação de SBRs vêm ganhando o interesse da indústria (Berstel e Lecont, 2010) e conforme Zacharias (2008), existe um renovado e crescente interesse a respeito de SBRs e seu desenvolvimento. Porém, ao mesmo tempo, existem poucos dados e trabalhos descrevendo como os SBRs são utilizados, como é o seu desenvolvimento e os desafios enfrentados por quem os utiliza.

Zacharias (2008) relata algumas áreas nas quais soluções com SBRs estão sendo utilizadas, sendo elas, seguro, finanças, saúde, biologia, jogos para computadores, viagens e engenharia de software, demonstrando a amplitude de áreas nas quais o desenvolvimento com SBR vem sendo utilizado.

O presente trabalho apresentou como objetivo geral identificar e descrever as ferramentas utilizadas para desenvolvimento com SBR, métricas para comparação de desenvolvimento com SBR e vantagens de utilização do desenvolvimento utilizando SBR, descrevendo também tipos de soluções implementadas com utilização de SBR.

Como objetivos específicos: a) Listar ferramentas utilizadas para desenvolvimento com SBR relatadas em trabalhos científicos que descrevam a implementação de uma solução, identificando o tipo de solução implementada; b) Descrever vantagens e desvantagens de utilização de SBR em desenvolvimento de software; c) Identificar métricas para comparação de desenvolvimento com SBR.

Para atender estes objetivos uma Revisão Sistemática da Literatura foi realizada, visando responder as questões de pesquisa estabelecidas:

*QP1:* Quais linguagens e/ou ferramentas utilizam SP-SBRs?;

*QP2:* Em que tipos de soluções pode-se utilizar SP-SBRs?;

*QP3:* Quais são as vantagens e desvantagens de utilização de SP-SBRs perante os demais modelos de programação?;

*QP4:* Quais elementos podem ser utilizados como métrica para desenvolvimento utilizando SP-SBR?

As duas primeiras questões foram respondidas, apresentando uma série de ferramentas e linguagens voltadas para desenvolvimento com SBR e uma vasta gama de soluções implementadas com SBR.

Por sua vez, as duas últimas questões permanecem em aberto. Apenas algumas vantagens foram demonstradas pelos artigos selecionados.

Especificamente sobre a questão 4, é necessário aprofundar o estudo buscando métodos e métricas utilizados para comparação de programação. Esta tarefa fica especificada como um trabalho futuro.

Desta forma, os objetivos estabelecidos para este trabalho foram cumpridos parcialmente, pois apenas foram listados as ferramentas e linguagens para SBR e algumas soluções que podem ser desenvolvidas com esta forma de programação.

## Referências


ABDULLAH, U., SAWAR, M. J., AHMED, A. Design of a Rule Based System Using Structured Query Language. 2009 Eighth IEEE International Conference on Dependable, Autonomic and Secure Computing, Chengdu, 2009, pp. 223-228.

ALHARBI, R. F., BERRI, J., EL-MASRI, S. Ontology based clinical decision support system for diabetes diagnostic. 2015 Science and Information Conference (SAI), London, 2015, pp. 597-602.

ARAKLIOTIS, S., NIKOLOS, D. G., KALLIGEROS, E. LAWRIS: A rule-based arduino programming system for young students. 2016 5th International Conference on Modern Circuits and Systems Technologies (MOCAST), Thessaloniki, 2016, pp. 1-4.

BANASZEWSKI, R. F. Paradigma Orientado a Notificações: Avanços e Comparações. Dissertação de Mestrado, CPGEI/UTFPR. Curitiba, 2009.

BERSTEL, B., LECONTE, M. Using Constraints to Verify Properties of Rule Programs. 2010 Third International Conference on Software Testing, Verification, and Validation Workshops, 2010, 349-354.

COHEN, M., RITTER, F., HAYNES, S. Evaluating Design: A Formative Evaluation of Agent Development Environments Used For Teaching Rule-Based Programming. 2009.

COPPIN, Ben. Inteligência artificial. Rio de Janeiro, RJ: LTC, 2010. 636 p.

EL-KHAYAT, G., MABROUK, T. Solving a fit maximization assignment problem using a rule based system and linear programming, The Joint International Symposium on CIE44 and IMSS’14, Istanbul, Turkey, 2014.

ESPÁK, M. Japlo: Rule-based Programming on Java. Journal of Universal Computer Science, 12(9):1177--1189, 2006.

FORGY, C. RETE: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem. Artificial Intelligence , 19 (1), pp. 17-37, 1982.

FRIEDMAN-HILL, E. Jess in Action: Java Rule-Based Systems. Manning Publications Co., Greenwich, CT, USA, 2003.

GIARRATANO, J. C. CLIPS 6.3 User's Guide. Gary Riley: 2015.

GRUMBACH, S., WANG, F. Netlog, a rule-based language for distributed programming. In Proceedings of the 12th international conference on Practical Aspects of Declarative Languages (PADL'10), Manuel Carro and Ricardo Peña (Eds.). Springer-Verlag, Berlin, Heidelberg, 2010, 88-103.

HATZILYGEROUDIS, I., KOVAS, K. A Tool for Automatic Creation of Rule-Based Expert Systems with CFs. IFIP Advances in Information and Communication Technology, Volume 339, Artificial Intelligence Applications and Innovations (AIAI-10), Springer, 195-202, 2010.

KITCHENHAM, B. A., CHARTERS, S. Guidelines for performing systematic literature reviews in software engineering (2007).

KRUG, D. L. Comparativo entre o aprendizado de Programação baseado na abordagem Imperativo-Procedimental e na abordagem de Sistemas Baseados em Regras. Seminário I, PPGCA/DAINF/UTFPR, 2016.

LEE, P.-Y., CHENG, A. M. HAL: A Faster Match Algorithm. IEEE Transactions on Knowledge and Data Engineering, 14 (5), pp. 1047-1058, 2002.

MACIOŁ, A., MACIOŁ, P., JĘDRUSIK, S., LELITO, J. The new hybrid rule-based tool to evaluate processes in manufacturing. International Journal of Advanced Manufacturing Technology 79: 1733-1745. 2015.

MACKERLE, J. A review of expert systems development tools. Engineering Computations. Vol. 6 Iss 1 pp. 2 - 17. 1989.

MIRANKER, D. P. TREAT: A better Match Algorithm for AI Production Systems. Sixth National Conference on Artificial Intelligence - AAAI'87, (pp. 42-47), 1987.

MIRANKER, D. P., BRANT, D. A., LOFASO, B., GADBOIS, D. On the Performance of Lazy Matching in Production Systems. 8th National Conference on Artificial Intelligence AAAI (pp. 685-692). AAAI Press / The MIT Press, 1990.

MOHER, D., LIBERATI, A., TETZLAFF, J., ALTMAN, D.G., The PRISMA Group. Preferred Reporting Items for Systematic Reviews and Meta-Analyses: The PRISMA Statement. PLoS Med. 2009.

NEWELL, A., SIMON, H. A. Human Problem Solving. Englewood Cliffs, NJ, USA: Prentice-Hall, 1972.

PANESCU, D., PASCAL, C., OLAERU, R. M. A rule-based approach for a multi-robot application. 19th International Conference on System Theory, Control and Computing (ICSTCC), Cheile Gradistei, 2015, pp. 75-80.

PARK, N., LEE, H. K., JANG, J. Rule-based modeling tool for web of things applications. 2015 IEEE 5th International Conference on Consumer Electronics - Berlin (ICCE-Berlin), Berlin, 2015, pp. 515-518.

RATTANASAWAD, T., SAIKAEW, K. R., BURANARACH, M., SUPNITHI, T. A review and comparison of rule languages and rule-based inference engines for the Semantic Web 2013 International Computer Science and Engineering Conference (ICSEC), 2013, 1-6.

RICH, Elaine; KNIGHT, Kevin. Artificial intelligence. 2nd. ed. New York: McGraw-Hill, 1991. xvii, 621 p.

RIVOLLI, A., ORLANDO, J., MOREIRA, D. An analysis of rules-based systems to improve SWRL tools ICEIS 2011 - Proceedings of the 13th International Conference on Enterprise Information Systems, 2011, 4 SAIC, 191-194.

ROY, P. V., HARIDI, S. Concepts, Techniques, and Models of Computer Programming. MIT Press, 2004.

ROY, P. V. Programming Paradigms for Dummies: What Every Programmer Should Know. In New Computational Paradigms for Computer Music. p. 9-47. G. Assayag and A. Gerzso (eds.), IRCAM/Delatour France, 2009.

RUSSELL, S., NORVIG, P. Artificial Intelligence: A Modern Approach. Prentice Hall,

2sd ed., 2003.

SAWAR, M. J., ABDULLAH, U., AHMED, A. Enhanced Design of a Rule Based Engine Implemented using Structured Query Language, The 2010 International Conference of Computational Intelligence and Intelligent Systems, at World Congress on Engineering, London, U.K., 30 June - 2 July, 2010. pp67-71.

SHIMBUICHI, M., MATSUOKA, K., TAKAMI, K. Architecture of network robot control server software written in a Rule-based language. IET 3rd International Conference on Wireless, Mobile and Multimedia Networks (ICWMNN 2010), Beijing, 2010, pp. 373-376.

SHIMOKURA, M., NAKANISHI, S., OHTA, T. Home Network Service Programs described in a Rule-based Language. International Conference on Software Engineering Advances (ICSEA 2007), Cap Esterel, 2007, pp. 62-62.

SHIMOKURA, M., NAKANISHI, S., OHTA, T. Network software architecture for a symbiotic human life with robots. 2008 7th Asia-Pacific Symposium on Information and Telecommunication Technologies, Bandos Island, 2008, pp. 1-6.

SIMÃO, J. M., STADZISZ P. C. An Agent-Oriented Inference Engine applied for Supervisory Control of Automated Manufacturing Systems. Frontiers in Artificial Intelligence and Applications ("Advances in Logic, Artificial Intelligence and Robotics" LAPTEC 2002 Edited by J. M. Abe e J. I. da Silva Filho). Vol. 85 (pp 234-241), IOS Press, Amsterdam - The Netherlands. ISSN: 0922-6389.

SIMÃO, J. M. A Contribution to the Development of a HMS Simulation Tool and Proposition of a Meta-Model for Holonic Control. 2005. Tese de Doutorado. CPGEI, CEFET-PR. Curitiba, Brasil, 2005.

VLAS, R., ROBINSON, W. N. A Rule-Based Natural Language Technique for Requirements Discovery and Classification in Open-Source Software Development Projects. In Proceedings of the 2011 44th Hawaii International Conference on System Sciences (HICSS '11). IEEE Computer Society, Washington, DC, USA, 2011, 1-10.

WANG, W., BAÑARES-ALCÁNTARA, R., CUI, Z., WANG, Y. J., COENEN, F. An association rule-based CLIPS program for interactive prediction of MSC

differentiation in vitro. 2010 International Conference on Computer Application and System Modeling (ICCASM 2010), Taiyuan, 2010, pp. V2-406-V2-410.

WATT, D. Programming Language Design Concepts. J. Willey & Sons, 2004.

YAMASHITA, T., OHTA, T., TAKAMI, K. Creating Web Services Using a Rule-Based Language. 2010 Fifth International Conference on Systems and Networks Communications, Nice, 2010, pp. 272-277.

ZACHARIAS, V. Development and Verification of Rule Based Systems - A Survey of Developers. Rule Representation, Interchange and Reasoning on the Web: International Symposium, RuleML 2008, Orlando, FL, USA, October 30-31, 2008.

## SEÇÃO COMPLEMENTAR B – TORRE DE HANÓI COM PON

Artigo escrito para a disciplina de Tópicos Avançados em Sistemas Embarcados 2 - Programação Orientada a Notificações do PPGCA.

# Torre de Hanói com LingPON – Paradigma Orientado a Notificações

**Douglas Lusa Krug[1], Hervé Panetto[2], Jean M. Simão[2]**

[1]Instituto Federal do Paraná – IFPR
Av. Paula Freitas s/n – 84.600-000 – União da Vitória – PR – Brasil

[2]Universidade Tecnológica Federal do Paraná – UTFPR
Av. Sete de Setembro, 3165 – Curitiba – PR – Brasil

***Resumo.*** *Este artigo relata a experiência da implementação de um problema clássico de lógica de programação, a Torre de Hanói, utilizando o Paradigma Orientado a Notificações – PON, e a Linguagem PON – LingPON. Através dele é possível entender um pouco mais sobre este novo paradigma, observando algumas comparações e sugestões de melhoria para o seu desenvolvimento.*

*Palavras chave: Paradigma Orientado a Notificações; Lógica de Programação; Torre de Hanói.*

***Abstract.*** *This article reports the experience of implementing a classical problem in programming logic, Hanoi Tower, using the NOP - Notification Oriented Paradigm, and the NOP language - LingPON. Through it is possible understand a little more about this new paradigm, noting some comparisons and improvement suggestions for its development.*

*Keywords: Notification Oriented Paradigm; Programming Logic; Hanoi Tower.*

## 1. Introdução

O Paradigma Orientado a Notificações – PON, vem sendo desenvolvido por um grupo de pesquisadores da Universidade Tecnológica Federal do Paraná – UTFPR, apresentando-se como uma alternativa para o desenvolvimento de aplicações em plataforma de *software* e *hardware*, ele se propõe a resolver certos problemas existentes nos paradigmas usuais de programação, como o Paradigma Imperativo – PI, e o Paradigma Declarativo – PD [Linhares, Simão e Stadzisz, 2014], [Simão e Stadzisz, 2008].

Desde sua concepção inicial, o PON vem passando por evoluções e sendo validado em diversas aplicações, tanto em *software* como em *hardware*.

Em termos de desenvolvimento de *software*, sua primeira aplicação foi através de um *framework*, desenvolvido em C++, em sua evolução como paradigma, nasceu a Linguagem PON, denominada LingPON.

Como parte de sua evolução, o paradigma e suas aplicações vêm sendo estudados e aprimorados constantemente por docentes e discentes da UTFPR. Durante uma das disciplinas relacionadas ao PON é solicitado aos discentes que escolham um

problema e criem a solução utilizando um paradigma vigente e o PON.

Para validar a facilidade de utilização da LingPON foi escolhido um problema clássico no estudo de lógica de programação, a Torre de Hanói, com o intuito de validar a facilidade de programação da linguagem, e a versatilidade, pois este problema é usualmente resolvido utilizando recursividade.

Inicialmente será abordado um pouco sobre o PON e sua materialização em LingPON, na sequência um breve histórico sobre a Torre de Hanói e sua explicação.

Será também explicado a forma de desenvolvimento utilizando a LingPON e uma comparação de desempenho entre o desenvolvimento em PON e o desenvolvimento no Paradigma Procedimental – PP, finalizando com algumas sugestões de melhoria para a LingPON.

## 2. Paradigma Orientado a Notificações

O Paradigma Orientado a Notificações – PON apresenta melhorias em comparação aos paradigmas vigentes, que por tempo vivem uma inércia de evolução, corrigindo certas deficiências apresentadas nestes, mas também aproveitando-se de pontos fortes que consagraram estes paradigmas.

O PON encontra inspirações no PI, como a flexibilidade algorítmica e a abstração de classes/objetos da Programação Orientada a Objetos – POO, assim como aproveita conceitos próprios do PD, como a facilidade de programação em alto nível e a representação do conhecimento em regras, dos Sistemas Baseados em Regras – SBR [Xavier, 2014].

Os principais elementos do PON são as *Fact Base Elements* – FBE e as *Rules*, as FBEs podem ser associadas a objetos do mundo real, e as *rules* podem ser associadas a regras de relação lógico causal.

O modelo e a lógica de funcionamento do PON podem ser descritos da seguinte forma: As *Rules* são compostas por *Conditions* e *Actions*, as *Conditions* podem se relacionar com uma ou mais *Premisses*, que por sua vez, são responsáveis por verificar os *Attributes* de uma FBE. Cada *Premisse* é composta por uma referência a um *Attribute*, por um operador lógico e um valor. Esta referência de um *Attribute* utilizada na *Premisse* é quem notifica a mudança de seu estado.

Uma *Rule* pode ser composta de uma ou mais *Premisses*, a partir do momento em que todas as *Premisses* são aprovadas, a *Rule* é aprovada e notifica uma *Action*. A *Action* referencia uma ou mais *Instigations*, as quais são associadas a *Methods* da FBE. Sempre que o valor de um *Attribute* é alterado ele mesmo notifica as *Premisses* que são relacionadas a ele. As *Premises*, por sua vez, são reavaliadas e, através de uma operação lógica é comparada ao novo valor do *Attribute* com uma constante ou um valor notificado por outro *Attribute*. Caso o resultado lógico da reavaliação da entidade *Premises* seja alterado, a *Premise* notifica um conjunto de entidades *Conditions* relacionadas a ela. Em seguida, as *Conditions* também têm seus estados lógicos reavaliados de acordo com os resultados lógicos das *Premises*. Assim, quando todas as entidades *Premises* que compõem uma entidade *Condition* apresentam seus valores lógicos verdadeiros, a entidade *Condition* também é satisfeita, aprovando a execução da sua respectiva *Rule*. Com isso, a entidade *Action* agregada a esta *Rule* é executada, invocando os *Methods* necessários através das entidades *Instigations* [Banaszewski, 2009].

A partir deste mecanismo de notificações é possível desenvolver programas com melhor desempenho, menor número de redundâncias estruturais e temporais. Estes programas são mais apropriados para paralelismo e distribuição do que os sistemas computacionais desenvolvidos por meio das soluções baseadas em paradigmas atuais [Pordeus, 2015].

Uma das formas de materialização do PON para software é utilizando a LingPON e o compilador construído para ela.

De modo geral, o código fonte da LingPON segue um padrão de declarações. Primeiramente, o desenvolvedor precisa definir os FBEs de seu programa. Em seguida, o desenvolvedor precisa declarar as instâncias de tais FBEs, bem como definir a estratégia de escalonamento das *Rules*. Subsequentemente, é necessário definir as *Rules* para fins de avaliação lógico causal dos estados do FBEs por meio de notificações. Por fim, é possível adicionar código específico da linguagem alvo escolhida no processo de compilação (e.g. C ou C++) com a utilização do bloco de código *main* [Ferreira, 2015].

## 3. Torre de Hanói

A Torre de Hanói é um problema clássico da matemática, fortemente utilizado como um método lúdico para desenvolver o raciocínio, e também muito utilizado no ensino de lógica e linguagens de programação como um exemplo de algoritmo que utiliza recursividade.

Este problema foi criado em 1883 pelo matemático Francês Édouard Lucas e consiste em um jogo com três hastes e um determinado número de discos, postos inicialmente na haste da esquerda, em ordem decrescente de tamanho, conforme ilustrado na Figura 1.

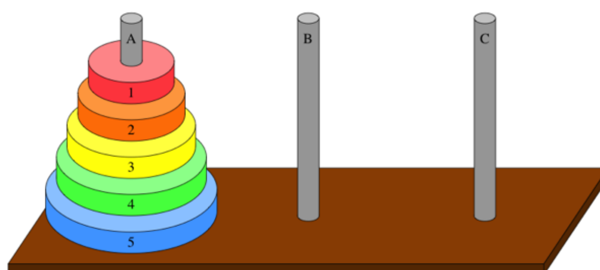



**Figura 1. Torre de Hanói – Configuração Inicial [Torre de Hanói, 2016]**

O objetivo do jogo é mover todos os discos para uma das hastes auxiliares, ficando conforme ilustrado na Figura 2.

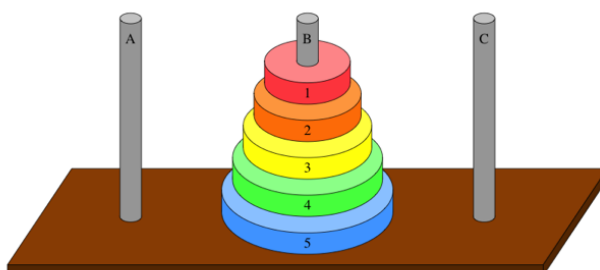



**Figura 2. Torre de Hanói – Final [Torre de Hanói, 2016]**

Embora o objetivo do jogo seja simples, duas regras devem ser seguidas:

1. Apenas um disco pode ser movido por vez;
2. Um disco maior nunca pode ser colocado em cima de um disco menor.

É também esperado que o objetivo seja cumprido com um número mínimo de movimentos, representado pela fórmula matemática ($2^n$ - 1), onde n é igual ao número de discos.

Desta forma, quando temos apenas 3 discos o número mínimo de movimentos necessário é 7, para 5 discos é 31 e assim por diante.

Embora o problema seja antigo ele ainda é muito utilizado no ensino de algoritmos e lógica de programação, principalmente para o ensino de recursividade para demonstrar a elegância do código com este recurso.

A escolha deste problema para validação do paradigma e da LingPON quanto à facilidade de programação e sua versatilidade se pautou principalmente no fato do problema ser utilizado amplamente no ensino de algoritmos e também da popularidade de solução recursiva, a qual não é uma característica do PON.

## 4. Desenvolvimento em LingPON

A materialização escolhida para o desenvolvimento da solução foi LingPON para atender ao objetivo que foi proposto, e compilado para C, C++ 1.0 e C++ 1.1.

Inicialmente foram definidas as FBEs necessárias para solucionar o problema, sendo elas Haste, Disco e Controle.

A FBE Haste conta com atributos que controlam o último disco que está em cada haste, para poder validar se o disco que está sendo movimentado para a haste é menor do que o que já está presente na haste.

Na FBE Disco contém o atributo que identifica em qual haste o disco está posicionado, seja haste A, B ou C.

Para a FBE Controle foram criados quatro atributos, sendo um atributo para controlar o último disco movimentado, um atributo para contar a quantidade de movimentos, um atributo para controlar se um movimento está sendo realizado, evitando paralelismo de movimentos, e um último atributo para armazenar o último movimento realizado, evitando repetição de movimentos.

Foram criadas 3 instâncias da FBE Haste, sendo uma instância para cada haste que compõe o problema. Para a FBE Disco foram criadas 3 instâncias também, sendo uma para cada disco, neste caso, o número de instâncias deve ser ampliado conforme o número de discos é ampliado. Caso seja necessário aumentar o número de discos, também é necessário criar outras regras para controlar os movimentos.

Já para a FBE Controle apenas uma instância foi criada, pois ela serve para controlar os movimentos de todos os discos entre todas as hastes.

Durante o desenvolvimento algumas dificuldades foram encontradas, fato esperado devido ao paradigma e a linguagem ainda estarem em desenvolvimento. Devido a estas dificuldades a implementação em LingPON ficou restrita a apenas 3 discos.

Foram criadas 8 regras, sendo 7 para controlar os movimentos e uma para indicar a finalização do processo. As 7 regras que controlam o movimento baseiam-se nas regras do jogo, onde não é possível movimentar um disco maior para cima de um disco

menor, além disso elas controlam os movimentos realizados, evitando movimentos desnecessários. Também para evitar o paralelismo de movimentos um atributo da FBE Controle foi utilizado para indicar o momento em que o movimento está concluído.

É possível verificar um exemplo de implementação de uma *rule* em LingPON na Figura 3 onde constam as *conditions* e *premisses* que compõe as condições para chamada das *instigations* que disparam os métodos das FBEs.

```
rule rlRegra_A_C_2

    condition
        subcondition scA_C_21
            premise prA_C_211 disco2.atHaste2 == 'A' and
            premise prA_C_212 hasteC.atUltimoDiscoC > 2 and
            premise prA_C_213 controle.atUltimoMovimentado != 2 and
            premise prA_C_214 controle.atLiberado == 1 and
            premise prA_C_216 hasteA.atUltimoDiscoA == 2
        end_subcondition
    end_condition

    action
        instigation inA_C_21 controle.mtLiberado0();
        instigation inA_C_23 hasteC.mtUltimoDisco2C();
        instigation inA_C_24 hasteA.mtUltimoDisco3A();
        instigation inA_C_25 disco2.mtSetaHaste2C();
        instigation inA_C_26 controle.mtUltimoMovimentado2();
        instigation inA_C_27 controle.mtIncMovimentos();
        instigation inA_C_28 controle.mtLiberado1();
    end_action

end_rule
```

**Figura 3. Exemplo de *rule* em LingPON**

## 5. Comparação de Desempenho

Além de desenvolver um programa em LingPON para melhorar o entendimento do paradigma, também foi desenvolvido um algoritmo utilizando um paradigma vigente para solução do problema, visando realizar algumas comparações.

Para comparação deste artigo os critérios a seguir foram selecionados: 1) facilidade de desenvolvimento; 2) legibilidade do código; 3) número mínimo de movimentos; e 4) desempenho.

O desenvolvimento do algoritmo de comparação para solucionar a Torre de Hanói foi desenvolvido em C, utilizando o Paradigma Procedimental – PP, de forma não recursiva.

Quanto à facilidade de desenvolvimento, é possível afirmar que a LingPON foi mais fácil, a partir do momento em que entende-se o funcionamento da mesma, a programação fica intuitiva.

A legibilidade do código gerado em LingPON é mais claro e de mais fácil leitura do que o gerado em C.

Quanto à quantidade de linhas geradas, em C foram geradas 180 linhas de código, já em LingPON foram geradas 286 linhas de código.

Ambos os algoritmos atingiram o objetivo de realizar a tarefa com o número mínimo de movimentos necessário para isso. Embora a flexibilidade e número de discos apenas foi possível utilizando a implementação no paradigma vigente.

Quanto ao desempenho, pode-se afirmar, sem sombra de dúvidas que o código

gerado utilizando o PON é melhor do que o gerado em PP.

Como não foi possível flexibilizar o número de discos em PON, para realizar a comparação de desempenho entre as implementações, foi optado por executar várias vezes o mesmo algoritmo, dessa forma permitindo um elevado número de movimentos para avaliar o desempenho.

Para comparação o código em LingPON foi compilado em C, C++ 1.0 e C++ 1.1. Era desejável gerar o código também para C++ 2.0 porém esta comparação deverá ser realizada em trabalhos futuros.

A Tabela 1 demonstra a comparação de desempenho entre os códigos compilados para o número de repetições que consta na coluna Repetições, o tempo relatado é em segundos. O tempo demonstrado é referente à média de 3 execuções de cada algoritmo para o número de repetições indicado.

**Tabela 1. Comparação de Desempenho**

| Repetições | C | PON – C 1.0 | PON – C++ – 1.0 | PON – C++ – 1.1 |
|---|---|---|---|---|
| 5 | 0,02 | | 0,02 | 0,02 |
| 10 | 0,04 | | 0,02 | 0,02 |
| 20 | 0,07 | | 0,03 | 0,03 |
| 50 | 0,15 | | 0,06 | 0,08 |
| 100 | 0,29 | | 0,12 | 0,14 |
| 500 | 1,45 | | 0,66 | 0,64 |
| 1000 | 2,84 | 1,00 | 1,28 | 1,37 |
| 10000 | 29,29 | 5,00 | 12,63 | 13,25 |
| 100000 | 289,44 | 56,00 | 127,69 | 129,10 |
| 1000000 | 2920,78 | 565,00 | 1250,59 | 1277,90 |

Analisando os dados é possível observar que o código gerado em C (PON) teve um desempenho melhor do que os demais programas.

Também é possível observar que existe uma pequena diferença de desempenho entre os programas compilados em C++ 1.0 e C++ 1.1, ambos PON.

O desempenho do programa desenvolvido em C, utilizando o PP, perdeu para todos os programas utilizando o PON, independente da compilação.

Esta comparação pode ser observada de melhor forma nas Figuras 4, 5, 6 e 7.

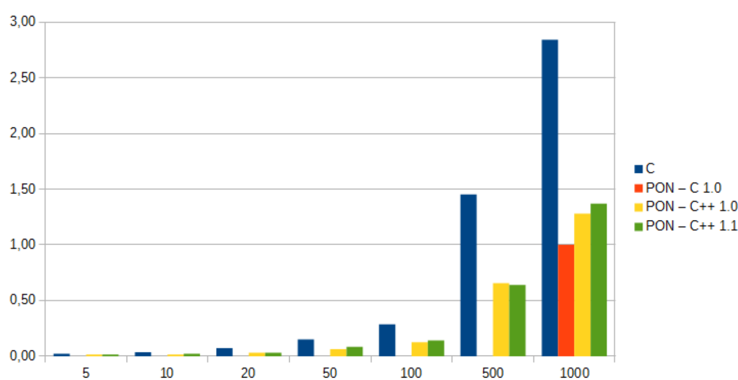



**Figura 4. Gráfico em Coluna – Até 1000**

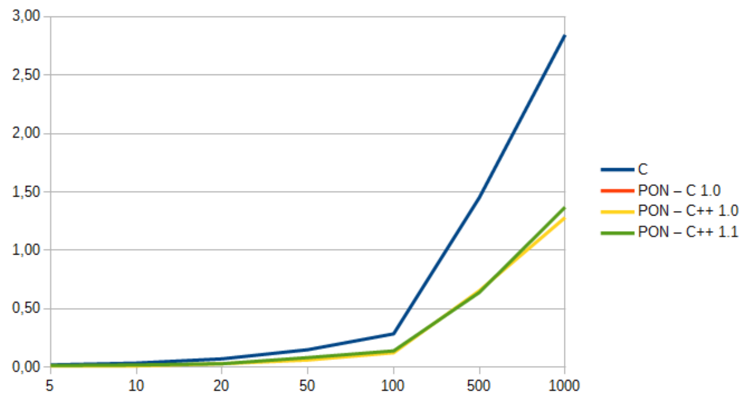


Figura 5. Gráfico em Linhas – Até 1000

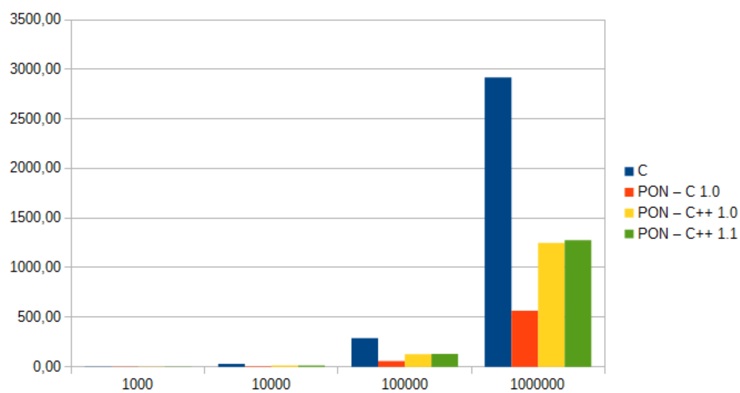


Figura 6. Gráfico em Coluna – A partir de 1000

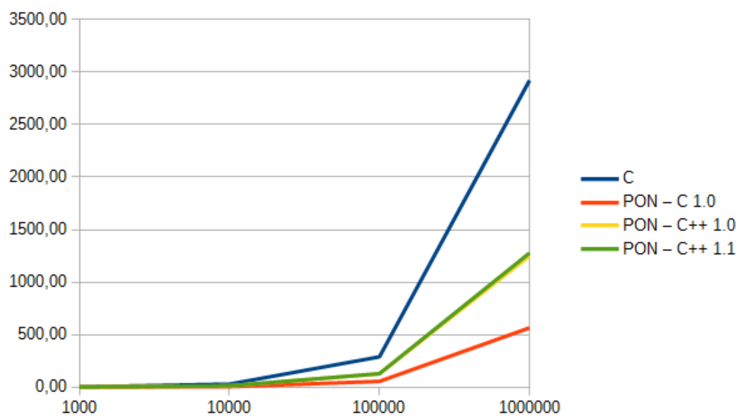


Figura 7. Gráfico em Linhas – A partir de 1000

## 6. Sugestões de Melhorias

Como é de se esperar em qualquer tecnologia que está em desenvolvimento, algumas dificuldades de implementação do algoritmo proposto foram encontradas, estas dificuldades servem de base para algumas sugestões de melhorias para a LingPON:

**1. Utilização de vetores:** Embora seja possível utilizar vetores construídos, um vetor clássico poderia auxiliar o desenvolvimento de vários algoritmos em LingPON, como é o do caso apresentado;

**2. Comparação entre atributos nas premissas:** Atualmente é possível construir as premissas comparando um atributo a uma constante, a comparação de valores entre atributos de diferentes instâncias é de grande utilidade;

**3. Alterar valor de atributos de diferentes instâncias:** Atualmente um método apenas consegue alterar o valor de atributos da própria instância em LingPON, permitir a alteração de atributos de diferentes instâncias através dos métodos das FBEs tornará o a LingPON mais robusta e dinâmica.

## 7. Conclusões

O PON, em várias de suas materializações, vem se apresentando como uma alternativa aos paradigmas vigentes, que vivem uma inércia quanto a sua evolução.

Com base no experimento que foi relatado neste artigo, é possível identificar uma grande facilidade de programação em LingPON, fácil entendimento do código gerado e também um grande ganho de desempenho em comparação com o paradigma imperativo.

Como é esperado de uma tecnologia em desenvolvimento, problemas ainda são encontrados, e através deles surgem sugestões de melhorias que visam contribuir o desenvolvimento do paradigma, principalmente no que tange sua materialização em LingPON.

## References


Linhares, R. R., Simão, J. M. e Stadzisz, P. C. (2014). Arquitetura de Computador Orientada a Notificações – ARQPON. Pedido de Patente. INPI, 2014.

Simão, J. M. e Stadzisz, P. C. (2008). Paradigma Orientado a Notificações (PON) – Uma Técnica de Composição e Execução de Software Orientado a Notificações. Pedido de Patente. INPI, 2008.

Xavier, R. D. (2014). Paradigmas de Desenvolvimento de Software: Comparação entre Abordagens Orientada a Eventos e Orientada a Notificações. Dissertação de Mestrado, PPGCA/UTFPR. Curitiba, 2014. Disponível em: http://repositorio.utfpr.edu.br/jspui/bitstream/1/1006/1/CT_PPGCA_M_Xavier%2c%20Robson%20Duarte_2014.pdf.

Banaszewski, R. F. (2009). Paradigma Orientado a Notificações: Avanços e Comparações. Dissertação de Mestrado, CPGEI/UTFPR. Curitiba, 2009. Disponível em: http://files.dirppg.ct.utfpr.edu.br/cpgei/Ano_2009/dissertacoes/Dissertacao_500_2009.pdf.

Pordeus, L. F. (2015). Notification Oriented Paradigm (NOP): CTA Simulator. Curitiba, 2015.

Ferreira, C. A. (2015). Linguagem e Compilador para o Paradigma Orientado a Notificações (PON): Avanços e Comparações. Dissertação de Mestrado, PPGCA/UTFPR. Curitiba, 2015. Disponível em: http://repositorio.utfpr.edu.br/jspui/bitstream/1/1414/1/CT_PPGCA_M_Ferreira%2c%20Cleverson%20Avelino_2015.pdf.

Torres de Hanói (2016). Disponível em: https://pt.khanacademy.org/computing/computer-science/algorithms/towers-of-hanoi/a/towers-of-hanoi Acessado em Maio de 2016.

## SEÇÃO COMPLEMENTAR C – TERMO DE CONSENTIMENTO LIVRE E ESCLARECIDO

### TERMO DE CONSENTIMENTO LIVRE E ESCLARECIDO

Seu (sua) filho (a) ______________________________________________, código de aluno ____________, está sendo convidado (a) a participar de um estudo científico denominado: “Comparativo entre o aprendizado de Programação baseado na abordagem Imperativo-Procedimental e na abordagem de Sistemas Baseados em Regras”, cujo o objetivo é comparar duas abordagens distintas de programação quanto ao aprendizado e a qualidade do software produzido.

A participação de voluntários é de fundamental importância para o avanço da pesquisa. Caso autorize seu (sua) filho (a) a participar desta pesquisa ele (a), em conjunto com os demais colegas de classe, o mesmo receberá instruções a respeito das duas abordagens de programação envolvidas, solucionará problemas em cada uma destas abordagens e responderá questionários a respeito do desenvolvimento.

As etapas desta pesquisa serão realizadas durante as aulas do componente curricular de Lógica e Linguagem de Programação / Programação I, sendo que o conteúdo que será trabalhado está em acordo com a ementa da disciplina, não apresentando perda ao seu (sua) filho (a) quanto ao aprendizado.

Estou ciente de que a privacidade de meu (minha) filho (a) será respeitada, ou seja, seu nome, ou qualquer outro dado confidencial, será mantido em sigilo. A elaboração final dos dados será feita de maneira codificada, respeitando o imperativo ético da confidencialidade.

A participação de seu (sua) filho (a) nessa pesquisa não é obrigatória e ele (a) pode desistir a qualquer momento, retirando seu consentimento, sem precisar justificar, nem sofrer qualquer dano.

Caso isso ocorra meu (minha) filho (a) continuará participando das aulas com as instruções, porém, os dados por ele (a) gerados não serão utilizados na pesquisa.

A pesquisa está sendo conduzida pelo Professor Douglas Lusa Krug do Instituto Federal do Paraná – IFPR, *campus* União da Vitória, com quem poderei manter contato pelo e-mail douglas.krug@ifpr.edu.br. Estão garantidas todas as informações que eu queira saber antes, durante e depois do estudo.

Li, portanto, este termo, fui orientado quanto ao teor da pesquisa acima mencionada e compreendi a natureza e o objetivo do estudo do qual meu (minha) filho (a) foi convidado a participar. Concordo, que meu (minha) filho (a) participará voluntariamente desta pesquisa, sabendo que não receberá nem pagará nenhum valor econômico pela sua participação.

Nome do responsável: _________________________________

Assinatura do responsável: _________________________________

Assinatura do pesquisador: _________________________________

União da Vitória, _____ de _________________ de 2017.

# SEÇÃO COMPLEMENTAR D – APOSTILA DE PARADIGMA PROCEDIMENTAL

Apostila sobre Paradigma Procedimental preparada para os encontros de instrução utilizados no experimento.

**Instituto Federal do Paraná – IFPR**

Apostila de:

# Introdução ao Paradigma Procedimental

Autor: Prof. Douglas Lusa Krug

Março de 2017

# Sumário

Sumário....................................................................................................2
Prefácio....................................................................................................3
Aula 1 – Introdução....................................................................................................4
Definição de Paradigma Procedimental....................................................................................................4
Introdução à Variáveis....................................................................................................4
Introdução à Estruturas de Decisão....................................................................................................4
Linguagem e Ambiente de Desenvolvimento....................................................................................................5
Primeiro Programa....................................................................................................6
Compilar e Executar....................................................................................................7
Aula 2 – Mais sobre Pascal....................................................................................................8
Estrutura básica de um programa....................................................................................................8
Exibir texto na tela....................................................................................................9
Salvar e editar arquivos....................................................................................................9
Mais estruturas de decisão....................................................................................................10
Múltiplas condições em uma estrutura de decisão....................................................................................................11
Aula 3 – Variáveis e operações....................................................................................................12
Tipos de variáveis....................................................................................................12
Atribuição de valores....................................................................................................12
Comparações e operações aritméticas....................................................................................................13
Referências....................................................................................................15

# Prefácio

Esta apostila foi escrita com o intuito de servir como base para os primeiros passos de programação utilizando o Paradigma Procedimental, um dos paradigmas mais utilizados para ensino de programação.

O Paradigma Procedimental baseia-se na sequência de ordens (comandos) para executar ações. A forma como o programa é executado é sequencial, tanto na atribuição quanto na comparação de valores.

O objetivo desta apostila é introduzir uma base aos princípios deste estilo de programação, aplicando a teoria em conjunto com a prática.

A partir dela será possível resolver problemas simples de programação, mas também, pode ser utilizada como orientação para elaboração de problemas mais complexos.

O público-alvo desta apostila são alunos do Curso Técnico em Informática Integrado ao Nível Médio do Instituto Federal do Paraná – IFPR, campus União da Vitória. Estes alunos estão sendo orientados com base nesta apostila para adquirir o conhecimento necessário para participar do experimento de mestrado intitulado "Comparativo entre o aprendizado de Programação baseado na abordagem Imperativo-Procedimental e na abordagem de Sistemas Baseados em Regras", cujo o objetivo é comparar duas abordagens distintas de programação quanto ao aprendizado e a qualidade do software produzido.

O experimento do mestrado está sendo conduzido pelo autor da apostila. Para mais informações entrar em contato pelo e-mail douglas.krug@ifpr.edu.br.

# Aula 1 – Introdução

## Definição de Paradigma Procedimental

O Paradigma Procedimental (PP) é um paradigma de programação (forma de pensar e escrever programas) que baseia-se na sequência de ordens (comandos) para executar ações.

Dentre estas ações podem existir atribuição de valores à variáveis e também comparação destes valores.

A sequência dos comandos é de extrema importância, pois programas escritos neste paradigma (modelo) são executados na sequência que estão escritos.

Foi um dos paradigmas mais utilizados até a década de 1990, mas ainda é bastante utilizado devido à sua facilidade de modularização, simplicidade e eficiência.

## Introdução à Variáveis

Pode-se definir variáveis como espaços utilizados na memória do computador que podem armazenar valores de forma temporária para a solução de problemas.

Os valores podem ser alterados no decorrer do programa.

Antes de armazenar um valor, a variável deve ser "declarada" no programa, indicando qual é o tipo de dados que ela irá armazenar (número, caracteres, sequência de caracteres, etc).

## Introdução à Estruturas de Decisão

São recursos utilizados para comparar valores, armazenados em variáveis ou não, e "autorizar" a execução de uma ou mais ações.

Estas estruturas servem para determinar se uma ação deve ou não ser executada com base nos valores comparados.

# Linguagem e Ambiente de Desenvolvimento

Para estes primeiros passos no PP será utilizada uma linguagem chamada Pascal, criada na década de 1970 com fins educativos e também para desenvolvimento de software comercial.

O compilador utilizado é o Pascalzim, desenvolvido no Departamento de Ciências da Computação da Universidade de Brasília.

A ferramenta pode ser encontrada no site http://pascalzimbr.blogspot.com.br

Após realizar o download e descompactar, abra o arquivo Pzim.exe. Caso deseje, pode criar um atalho na Área de Trabalho para facilitar o acesso.

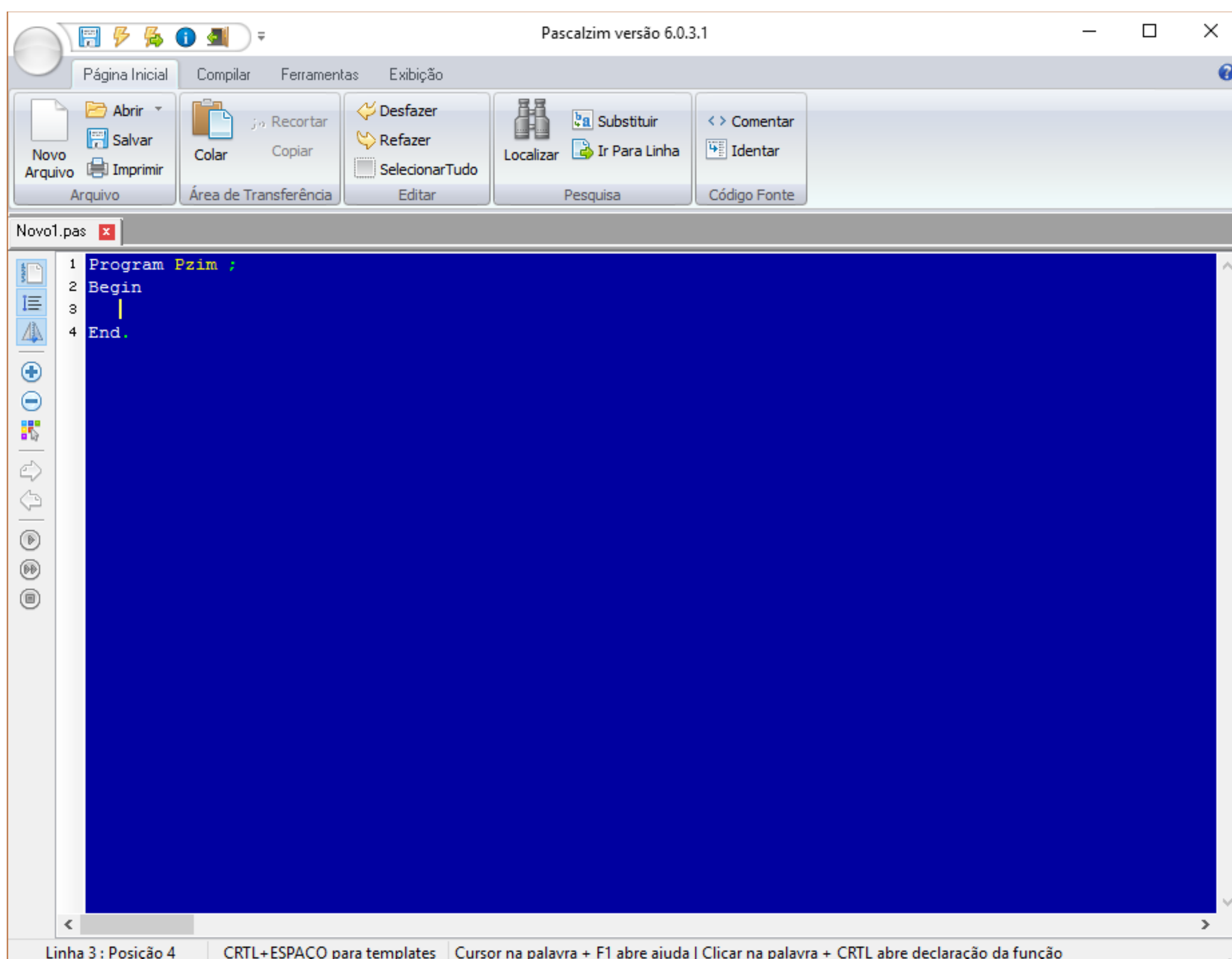



*Figura 1: Ambiente de desenvolvimento - Pascalzim.*

Ao abrir a ferramenta aparecerá uma janela como a ilustrada na Figura 1.

O programa é escrito na tela azul demonstrada acima. É possível salvar arquivos e abrir arquivos já criados para modificação.

# Primeiro Programa

Para entender um pouco dos recursos da linguagem Pascal e do Paradigma Procedimental será escrito um pequeno programa.

Escreva no editor (tela azul) o programa a seguir:

```
Program PrimeiroPrograma ;

Var animal, som: string;

Begin
  writeln ('Informe qual é o animal:');
  read (animal);

  if (animal = 'pato')
  then
          som := 'quack';

  writeln (som);
End.
```

O comando “Program” é utilizado para indicar o nome do programa, neste exemplo PrimeiroPrograma. Exemplo: `Program PrimeiroPrograma ;`

Para declarar variáveis utiliza-se o comando “Var”, seguido pelo identificador destas variáveis e depois o tipo. Exemplo: `Var animal, som: string;`

Para determinar o início e o fim de um programa (escopo do programa), utiliza-se os comandos Begin e End.

O comando `writeln` é utilizado para exibir uma mensagem na tela, ela pode ser um texto escrito diretamente ou ainda o valor armazenado em uma variável. Exemplo: `writeln ('Informe qual é o animal:');`

Para armazenar em uma variável o valor informado pelo usuário, utiliza-se o comando `read` com a variável entre parênteses. Exemplo: `read (animal);`

Outra forma de armazenar um valor em uma variável é a atribuição direta, utilizando o nome da variável seguido pelo valor, separados por := Exemplo: `som := 'quack';`

Para utilizar uma estrutura de decisão, usamos o comando `if` com a comparação entre parênteses. A ação que deve ser executada vem após o comando `then`

Exemplo:

```
if (animal = 'pato')
  then
```

## Compilar e Executar

Após escrever o programa é necessário compilá-lo, ou seja, verificar se está sintaticamente escrito de forma correta transformando ele em uma linguagem legível ao computador.

Para compilar o programa utilize a opção "Compilar Programa" na aba Compilar.

Caso tudo ocorra bem, é possível realizar a execução do programa, acompanhando o resultado. Para isso utilize a opção "Executar Programa".

Ambas as opções estão ilustradas na Figura 2.

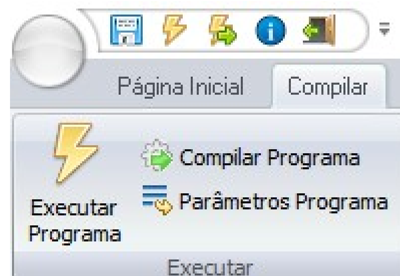



*Figura 2: Compilar e executar programa.*

# Aula 2 – Mais sobre Pascal

## Estrutura básica de um programa

Conforme visto anteriormente um programa em Pascal é executado de forma sequencial, seguindo cada comando, atribuição e comparação. Existe uma estrutura básica para os programas escritos nesta linguagem. Esta estrutura será vista a seguir.

Exemplo de estrutura básica:

| | |
|---|---|
| `program nome_do_programa;` | Cabeçalho do programa, indica o nome dele. |
| `Var` | Seção de declarações de constantes e variáveis. |
| `    nome: tipo;` | Declaração de variável, indica o nome e o tipo. |
| `Begin` | Inicia a seção de comandos. |
| `    comandos` | Ações que o programa executará. |
| `End` | Encerra a seção de comandos. |

Caso queira deixar um comentário em um programa escrito na linguagem Pascal, basta colocar o texto entre chaves { } ou entre parênteses e asterisco (* *). Também é possível comentar uma linha inteira utilizando duas barras //.

Utilizar comentários é interessante para indicar o motivo que levou a utilizar o que foi utilizado, principalmente quando outra pessoa irá ler seu código.

Geralmente colocamos comentários no início de um programa, indicando quem o fez o porque ele foi feito.

A identação (espaços antes dos comandos) serve para hierarquizar o código e deixá-lo de forma mais legível. Veja no exemplo da estrutura que a declaração de variáveis e as ações estão em alinhamento distinto do cabeçalho e do Begin e End do programa. Isto facilita a validação e entendimento do código.

Tanto os comentários quanto a identação fazem parte das boas práticas de programação, e devem ser utilizados independente do paradigma ou modelo de programação que está sendo usado.

Exemplo de utilização de comentários:

```
//Programa de exemplo

//Criado por Douglas Lusa Krug
```

```
Programa PrimeiroPrograma {este é meu primeiro programa}

Var

     animal: string;     (*variável utilizada para salvar o tipo do animal*)
```

## Exibir texto na tela

É possível exibir texto na tela do computador utilizando programas em Pascal, para isso utiliza-se o comando `write` ou o comando `writeln`.

A diferença entre eles é que o comando `writeln` adiciona uma quebra de linha (pula a linha) após exibir o texto.

Através destes comandos é possível exibir um texto diretamente, o valor armazenado em uma variável ou ainda um texto em conjunto com o valor de uma variável.

Exemplos:

Texto direto `writeln('Exibindo um texto diretamente');`

Valor de uma variável `writeln(nome);`

Texto com valor de uma variável `writeln('Valor: ', variavel, '!');`

O que será exibido deve vir entre parênteses após o comando `writeln`, o texto deve estar entre aspas simples. Para separa texto de variável utiliza-se uma vírgula.

### Exercício 1

Crie um novo programa que solicite o seu nome e imprima uma mensagem na tela lhe desejando bom dia. Na mensagem deve aparecer o nome informado, por exemplo, “Bom dia Douglas!”.

## Salvar e editar arquivos

É possível salvar e editar arquivos já criados utilizando o Pascalzim.

Para criar um novo arquivo basta ir na aba “Página Inicial” e escolher a opção “Novo Arquivo”, ou ainda utilizar a tecla de atalho Ctrl + N.

Para salvar o arquivo basta ir na aba “Página Inicial” e escolher a opção “Salvar” ou ainda utilizar o atalho Ctrl + S.

Para abrir um arquivo já criado para edição, basta ir na aba “Página Inicial” e escolher a opção “Abrir” ou ainda utilizar o atalho Ctrl + O.

Estas opções estão ilustradas na Figura 3.

*Figura 3: Opções de arquivo.*

## Mais estruturas de decisão

Uma estrutura de decisão permite ou restringe a execução de determinados comandos. Caso uma condição seja satisfeita os comandos indicados serão executados, caso não seja satisfeita os comandos não são executados.

Geralmente programas complexos contém diversas estruturas de decisão.

Vamos simular o programa de um robô, que deve atender às luzes de um semáforo para andar ou parar. Desta forma vamos construir uma estrutura de decisão para que o robô ande quando a luz do semáforo for verde e outra estrutura de decisão para que o robô pare quando a luz do semáforo for vermelha.

Abra um novo arquivo e escreva o programa ilustrado na Figura 4:

```
1  Program Robo;
2  var
3    cor_luz: string;  {variavel utilizada para receber a cor da luz do semáforo}
4  Begin
5    write ('Informe a cor do semáforo: ');
6    read(cor_luz);     {recebe a cor da luz do semáforo}
7
8    if (cor_luz = 'verde')  {se a luz for verde, andar}
9    then
10     writeln ('Ande!');
11
12   if (cor_luz = 'vermelha')  {se a luz for vermelha, parar}
13   then
14     writeln ('Pare!');
15 End.
```

*Figura 4: Programa exemplo.*

Após finalizar a edição do arquivo salve ele, compile e execute.

Faça o teste informando as cores verde, vermelha e amarelo. Uma em cada execução.

## Múltiplas condições em uma estrutura de decisão

É comum combinar mais de uma validação em uma estrutura de decisão. Para isso podemos utilizar os operadores `and` e `or`.

Eles são equivalentes aos condicionais E e OU da lógica proposicional.

Quando utilizamos o `and` as ações após o `then` apenas serão executadas quando todas as condições forem satisfeitas.

Quando utilizamos o `or` as ações após o `then` apenas serão executadas quando no mínimo uma das condições forem satisfeitas.

Por exemplo, podemos indicar que para que o robô pare é necessário que o sinal esteja vermelho e que o robô esteja andando. Para isso basta adicionar mais uma condição na estrutura de decisão, como uma variável chamada `status` com o valor `andando`.

As condições devem estar entre parênteses de forma separada, sendo que o and estará entre elas.

Exemplo:

```
if (cor_luz = 'vermelha') and (status = 'andando')
then
     writeln ('Pare!');
```

# Aula 3 – Variáveis e operações

## Tipos de variáveis

Na linguagem Pascal existem alguns tipos de dados predefinidos pelo compilador, estes tipos de dados são utilizados nas definições das variáveis e determinam o que pode ser armazenado em cada variável.

Os tipos de dados predefinidos são:

- Boolean: Permite dois tipos de valores TRUE e FALSE;
- Char: Permite armazenar um caractere, seja ele uma letra, um número ou um símbolo;
- Integer: Permite armazenar números inteiros, sendo eles negativos ou positivos, no intervalo de -2.147.483.648 até 2.147.483.647;
- Real: Permite armazenar números reais, sendo eles negativos ou positivos;
- String: Permite armazenar um conjunto ou cadeia de caracteres, o tamanho máximo permitido é de 255 caracteres.

Relembrando que a declaração de variáveis é realizada antes da instrução `Begin`, sendo iniciada pela instrução `Var`, conforme exemplo ilustrado na Figura 5.

```
1 Program variaveis ;
2 Var
3   teste: boolean;
4   genero: char;
5   idade: integer;
6   valor: real;
7   nome: string;
8 Begin
```

*Figura 5: Declaração de variáveis.*

## Atribuição de valores

A atribuição de um valor a uma variável pode ocorrer de forma direta no programa ou através da entrada fornecida por um usuário.

A forma direta ocorre informando o nome da variável seguida por `:=` e o valor que deseja atribuir, por exemplo: `nome := 'Douglas';`

Para variáveis do tipo char e string os valores devem ser informados entre aspas simples, para os demais sem aspas. Para variáveis do tipo real, a separação do decimal deve ser feita utilizando um ponto.

A atribuição de valores a partir da entrada fornecida pelo usuário ocorre com o comando `read(nome_variavel)` sendo que entre os parênteses é informada a variável que receberá o valor informado pelo usuário. Por exemplo: `read (nome);`

## Comparações e operações aritméticas

É possível realizar operações aritméticas com a linguagem Pascal, para isso utiliza-se os operadores a seguir:

- + para operação de soma, exemplo: `valor := 2 + 3;`
- - para operação de subtração, exemplo: `valor := 6 – 3;`
- * para operação de multiplicação, exemplo: `valor := 5 * 2;`
- / para operação de divisão, exemplo: `valor := 18 / 3;`

Caso necessário é possível combinar várias operações, por exemplo, o comando a seguir está fazendo a média aritmética de 3 números e armazenando o resultado na variável chamada media: `media := (6 + 7 + 9) / 3;`

Crie um novo programa Pascal e copie as atribuições acima e verifique o resultado.

Ao utilizar estruturas de decisão no Pascal é possível realizar comparação entre valores, para estas comparações os operadores relacionais a seguir podem ser utilizados:

- = Compara se ambos os valores são iguais, caso sim, o valor da comparação é verdadeiro;
- <> Compara se os valores são diferentes, caso sejam, o valor da comparação é verdadeiro;
- < Verdadeiro caso o valor da esquerda seja menor do que o valor da direita;
- <= Verdadeiro caso o valor da esquerda seja menor ou igual ao valor da direita;
- > Verdadeiro caso o valor da esquerda seja maior do que o valor da direita;
- >= Verdadeiro caso o valor da esquerda seja maior ou igual ao valor da direita;

Exemplo de utilização: `if (valor1 >= 10) then`

## Exercício 2

Abra um novo arquivo no Pascalzim e copie o programa da Figura 6. Este programa solicita 2 números ao usuário, retorna a soma destes 2 números e informa se a soma é maior do que 100.

```
1  // Programa exemplo de soma
2  // Criado por Douglas Lusa Krug
3
4  Program soma ;
5  Var
6    valor1 : real;
7    valor2 : real;
8    soma : real;
9  Begin
10   {solicita o primeiro valor}
11   writeln ('Informe o primeiro valor:');
12   read (valor1);
13
14   {solicita o segundo valor}
15   writeln ('Informe o segundo valor:');
16   read (valor2);
17
18   {realiza a soma dos 2 valores}
19   soma := valor1 + valor2;
20
21   writeln ('A soma dos valores é: ', soma);
22
23   {verifica se a soma é maior do que 100}
24   if (soma >= 100)
25   then
26     writeln ('Soma maior que 100!');
27
28 End.
```

*Figura 6: Exemplo soma.*

## Exercício 3

Crie um programa em Pascal que solicite o ano de nascimento de uma pessoa, calcule a idade dela com base no ano atual, e informe se a mesma tem obrigação de votar.

Utilize identação e comentários no programa.

# Referências

KRUG, D. L., SIMÃO, J. M., BASTOS, L. C. Comparativo entre o aprendizado de Programação baseado na abordagem Imperativo-Procedimental e na abordagem de Sistemas Baseados em Regras. Seminário I, PPGCA/DAINF/UTFPR, 2016.

Documentação do compilador Pascalzim, disponível em http://pascalzimbr.blogspot.com.br/

## SEÇÃO COMPLEMENTAR E – APOSTILA DE SISTEMAS BASEADOS EM REGRAS

Apostila sobre Sistemas Baseados em Regras preparada para os encontros de instrução utilizados no experimento.

**Instituto Federal do Paraná – IFPR**

Apostila de:

# Sistemas Baseados em Regras

Autor: Prof. Douglas Lusa Krug

Março de 2017

# Sumário

Sumário....................2
Prefácio....................3
Aula 1 – Introdução....................4
Definição de Sistemas Baseados em Regras....................4
Composição dos SBRs....................4
Ferramenta CLIPS....................5
Base de Fatos....................6
Base de Regras....................8
Aula 2 – Mais Sobre Regras....................10
Estrutura de uma regra....................10
Exibir texto na tela....................11
Arquivo com o programa....................11
Mais regras....................12
Múltiplas condições em uma regra....................14
Aula 3 – Programas dinâmicos....................16
Solicitando dados ao usuário....................16
Verificando valores na base de fatos....................16
Operações aritméticas....................17
Aula 4 – Repetições....................21
Variáveis globais....................21
Referência para exclusão de um fato....................21
Repetindo ações com CLIPS....................21
Referências....................23

# Prefácio

Esta apostila foi escrita com o intuito de servir como base para os primeiros passos de programação utilizando Sistemas Baseados em Regras (SBRs). SBRs são uma forma de programação pertencente ao Paradigma Lógico e representam uma forma de programação declarativa, de forma não sequencial.

Os SBRs utilizam fatos e regras para resolução de problemas e originalmente foram escritos para simular a forma como um ser humano resolve os problemas.

O objetivo desta apostila é introduzir uma base aos princípios deste estilo de programação, aplicando a teoria em conjunto com a prática.

A partir dela será possível resolver problemas simples de programação, mas também, pode ser utilizada como orientação para elaboração de problemas mais complexos.

O público-alvo desta apostila são alunos do Curso Técnico em Informática Integrado ao Nível Médio do Instituto Federal do Paraná – IFPR, campus União da Vitória. Estes alunos estão sendo orientados com base nesta apostila para adquirir o conhecimento necessário para participar do experimento de mestrado intitulado “Comparativo entre o aprendizado de Programação baseado na abordagem Imperativo-Procedimental e na abordagem de Sistemas Baseados em Regras”, cujo o objetivo é comparar duas abordagens distintas de programação quanto ao aprendizado e a qualidade do software produzido.

O experimento do mestrado está sendo conduzido pelo autor da apostila. Para mais informações entrar em contato pelo e-mail douglas.krug@ifpr.edu.br.

# Aula 1 – Introdução

## Definição de Sistemas Baseados em Regras

Sistemas Baseados em Regras (SBRs) são uma forma de programação pertencente ao Paradigma Lógico. O Paradigma Lógico é um paradigma de programação (forma de pensar e escrever programas) que enfatiza a descrição declarativa do problema em vez de decompor o problema em uma sequência ordenada de comandos.

O SBR foi criado com o intuito de simular o raciocínio humano, relacionando fatos com regras.

## Composição dos SBRs

Os SBRs trabalham essencialmente com duas bases de elementos, a Base de Fatos e a Base de Regras.

Na Base de Fatos são guardados temporariamente alguns valores que podem ser utilizados para resolução de problemas.

Na Base de Regras são guardadas condições que relacionam fatos com valores para determinar a execução de uma ação.

Os fatos ativam as regras a fim de produzir uma resposta para um problema, desta forma executando uma determinada ação.

Além da Base de Fatos e da Base de Regras, existe um mecanismo chamado de Máquina de Inferência (MI), que é o responsável por processar as bases e encontrar as partes correspondentes, determinando a execução das ações.

A ilustração da arquitetura de um SBR é apresentada na Figura 1.

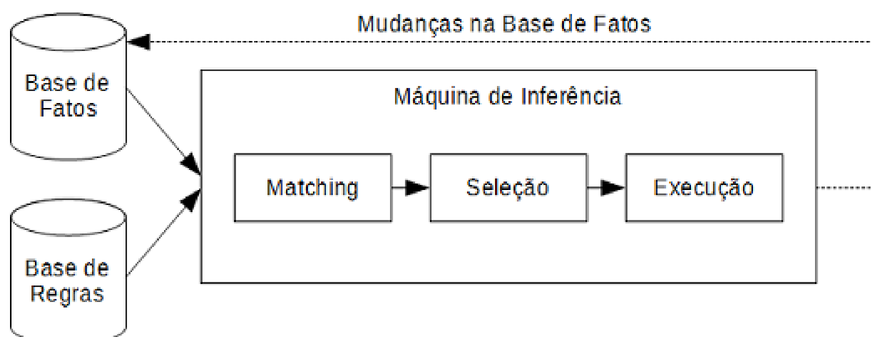



*Figura 1: Arquitetura de um SBR – adaptado de FRIEDMAN-HILL, 2003.*

Conforme apresentado na Figura 1 a MI é composta pelas fases de *Matching* (casamento), Seleção e Execução.

**Casamento:** fase responsável por relacionar os valores dos elementos armazenados na **Base de Fatos** com as regras da **Base de Regras** buscando encontrar correspondência.

Caso encontre correspondência, esta regra é ativada e armazenada em uma área chamada **Conjunto de Conflito**.

**Seleção:** pega as regras ativadas que estão na área de **Conjunto de Conflito** e as ordena com base em regras de resolução de conflito, que podem ser devido a prioridades definidas ou ainda devido à recentidade.

Estas regras ordenadas formam uma **Agenda**.

**Execução:** executa as ações indicadas pelas regras que estão na **Agenda**. Durante a execução das ações determinadas pelas regras valores dos elementos da **Base de Fatos** podem ser alterados, ativando novas regras.

# Ferramenta CLIPS

Para desenvolver programas utilizando SBRs é possível utilizar diversas linguagens de programação e ambientes de desenvolvimento. Esta apostila apresenta a linguagem (e ferramenta) CLIPS.

C Language Integrated Production System – CLIPS é uma linguagem de programação (e ferramenta) desenvolvida em 1986 por Software Technology Branch (STB) e NASA/Lyndon B. Johnson Space Center.

CLIPS é uma ferramenta completa para SBR e Sistemas Especialistas.

A ferramenta pode ser encontrada no site http://www.clipsrules.net/

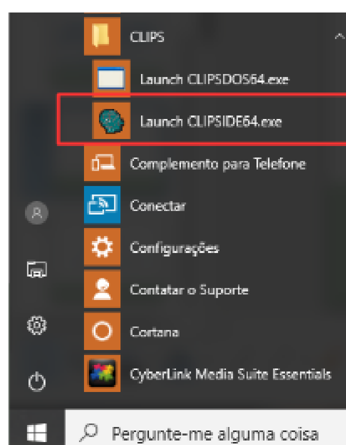



*Figura 2: Ferramenta CLIPS.*

Após instalada, para abrir a ferramenta basta ir até o menu Iniciar e encontrar a pasta CLIPS. Escolha o programa “Launch CLIPSIDE64.exe”, conforme ilustrado na Figura 2.

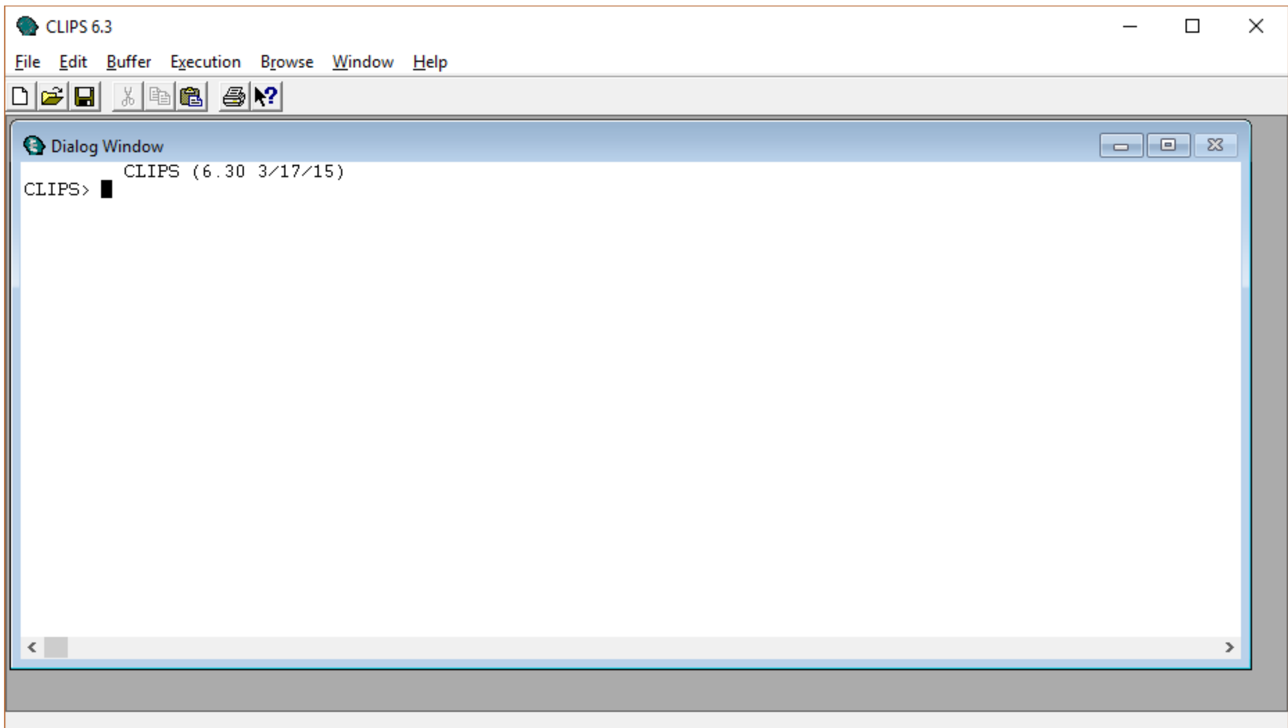



*Figura 3: Ambiente CLIPS.*

Ao abrir a ferramenta aparecerá uma janela como a ilustrada na Figura 3.

Os comandos podem ser escritos ao lado do indicador “CLIPS>” onde o cursor estará piscando.

Os comandos utilizados na linguagem CLIPS devem utilizar parênteses como delimitador, ou seja, todo comando inicia abrindo parênteses e encerra fechando parênteses.

# Base de Fatos

Para verificar a Base de Fatos no CLIPS utiliza-se o comando `(facts)` digitando ele diretamente ao lado do CLIPS> seguido pelo comando “Enter”.

Após utilizar este comando será listado todos os fatos que estão armazenados na Base de Fatos, conforme ilustrado na Figura 4.

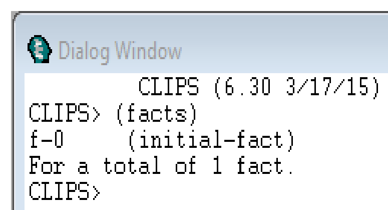



*Figura 4: Base de Fatos.*

Inicialmente a Base de Fatos contém apenas o elemento (initial-fact) que pode ser utilizado para carregar fatos iniciais em alguns programas.

Para adicionar um novo fato à Base de Fatos utilizamos o comando `assert`, informando também o nome do fato e o valor do mesmo. Conforme exemplo ilustrado na Figura 5.

Exemplo: `(assert (nome-aluno Douglas))`

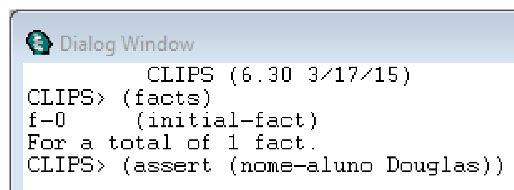



*Figura 5: Comando assert.*

Após o comando `assert` é informando, entre parênteses, o nome do elemento “nome-aluno” seguido pelo valor do mesmo “Douglas”.

Caso tudo ocorra bem, aparecerá a mensagem “<Fact-1>” onde o número após o hífen é a ordem do fato adicionado, conforme ilustrado na Figura 6.

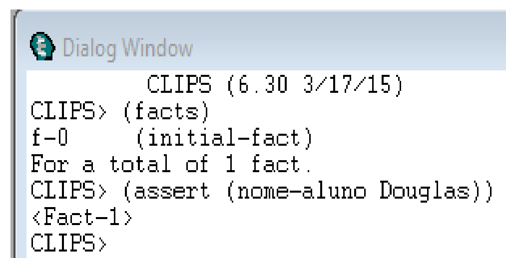



*Figura 6: Fato adicionado.*

Após adicionar o fato é possível verificar o mesmo na base de fatos, utilizando o comando `(facts)`, conforme ilustrado na Figura 7.

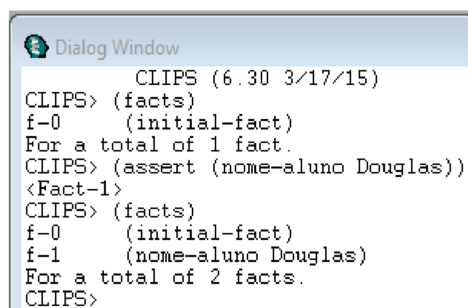



*Figura 7: Consultando Base de Fatos.*

Importante observar os parênteses utilizados, sempre é necessário utilizar o parêntese para o comando `assert` e para os valores deste comando.

Caso queira excluir um valor da Base de Fatos é possível fazê-lo com o comando `retract`, indicando a ordem do fato armazenado na Base de Fatos (número após o Fact-).

Exemplo: `(retract 1)`

Caso queira limpar toda a Base de Fatos basta utilizar o comando `clear`, também entre parênteses.

Exemplo: `(clear)`

Ao inserir um novo fato após utilizar o comando `retract`, a numeração do fato excluído não é reaproveitada. Ao utilizar o comando `clear` a numeração é zerada e reiniciada, incluindo novamente o (initial-fact).

## Base de Regras

Para adicionar uma regra no CLIPS utiliza-se o comando `defrule`.

Exemplo:

```
(defrule duck
   (animal-is duck)
=>
   (assert (sound-is quack))
)
```

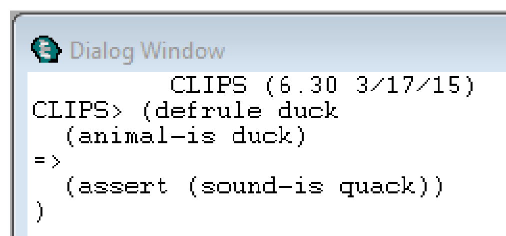



*Figura 8: Criando uma Regra.*

A regra pode ser criada diretamente ao lado do CLIPS, conforme ilustrado na Figura 8, após fechar o parêntese do comando `defrule` basta dar um Enter para a regra estar criada.

`Defrule` é o comando do CLIPS para criar uma regra e `duck` é o nome da regra.

A linha `(animal-is duck)` verifica se existe na Base de Fatos um elemento chamado `animal-is` com valor igual a `duck`. Caso isso seja verdade, a regra é ativada e adicionada na Agenda.

O conjunto com os sinais `=>` separa as condições das ações. Antes dela está o conjunto de condições que deve ser satisfeita, após ela estão as ações que serão executadas caso a regra seja ativada.

O comando `(assert (sound-is quack))` adiciona um elemento chamado `sound-is` com valor `quack` na Base de Fatos. Como está após o `=>` ele somente é executado quando a regra é ativada.

Desta forma pode-se dizer que uma regra constitui da parte com as condições para que ela seja executada e das ações que serão executadas caso as condições sejam atendidas. Isto é separado pelos sinais `=>`

Após criar a regra é necessário executar o programa, fazendo com que as regras sejam ativadas caso as condições sejam atendidas. Para isso utiliza-se o comando `(run)`.

Adicione um elemento na base de fatos para ativar a regra criada. Comando `(assert (animal-is duck))`. Após isso execute o comando `(run)`.

Caso tudo ocorra conforme o esperado, um novo elemento chamado `sound-is` com o valor `quack` será adicionado à Base de Fatos. Para verificar liste os fatos com o comando `(facts)`.

# Aula 2 – Mais Sobre Regras

## Estrutura de uma regra

Conforme visto anteriormente uma regra é constituída basicamente da condição e das ações. A parte com as condições é conhecida como *Left-Hand Side* (LHS) e são os padrões antes do `=>` . As ações que são executadas caso os padrões do LHS sejam ativados está no que é conhecido como *Right-Hand Side* (RHS) e estão após o `=>` .

Para cada regra é possível adicionar um comentário que descreve para que ela serve. Este comentário deve ser colocado depois do nome e antes dos padrões (condições).

Exemplo de estrutura:

| Código | Descrição |
|---|---|
| `(defrule nome_da_regra` | Indica a criação de uma regra e o seu nome. |
| `    "Exemplo de estrutura"` | Comentário de cabeçalho, para que serve a regra. |
| `    (padrão 1)` | Inicia os padrões, ou seja, as condições que |
| `    (padrão 2)` | devem ser atendidas para executar as ações. |
| `    (padrão n)` | Uma regra pode ter de 0 até várias condições. |
| `=>` | Separa as condições das ações. |
| `    (ação 1)` | Ações que serão executadas caso todas as |
| `    (ação 2)` | condições (padrões) sejam atendidas. |
| `    (ação n)` | Uma regra pode ter 0 ou mais ações. |
| `)` | Indica o final da regra. |

Caso queira utilizar mais comentários na criação de uma regra, por exemplo, para indicar o motivo de utilizar uma condição ou uma ação, basta adicionar o comentário precedido por ponto e vírgula ";".

Utilizar comentários é interessante para indicar o motivo que levou a utilizar o que foi utilizado, principalmente quando outra pessoa irá ler seu código.

A identação (espaços antes dos comandos) serve para hierarquizar o código e deixá-lo de forma mais legível. Veja no exemplo da estrutura que os padrões e as ações estão em alinhamento distinto do início e fim da regra, também da separação das partes. Isto facilita a validação e entendimento do código.

Tanto os comentários quanto a identação fazem parte das boas práticas de programação, e devem ser utilizados independente do paradigma ou modelo de programação que está sendo usado.

Exemplo de utilização de comentários:

```
(defrule duck
   (animal-is duck)      ; verifica se existe o animal é um duck
=>
   (assert (sound-is quack))  ; adiciona o elemento sound-is
)
```

O nome de regras é único, ou seja, apenas uma regra pode existir com um determinado nome. Caso seja utilizado o nome de uma regra já existente, o conteúdo da anterior será substituído pelo conteúdo da nova.

## Exibir texto na tela

É possível exibir texto na tela do CLIPS, para isso utiliza-se o comando `printout`.

Exemplo: `(printout t "texto" crlf))`

O comando `printout` deve ser utilizado entre parênteses. A letra `t` indica que a mensagem será exibida na tela. A mensagem que será exibida deve estar entre aspas duplas. O `crlf` serve para melhor disposição do texto ao ser exibido.

### Exercício 1

Crie uma nova regra chamada `duck`, substituindo a adição do elemento `sound-is` pela exibição do quack na tela.

## Arquivo com o programa

É possível criar um arquivo contendo as regras no CLIPS, para que seja de melhor visualização e edição, e também para utilização futura de forma rápida e fácil.

Para isso basta ir ao menu File – New, ou ainda utilizar a tecla de atalho Ctrl + N.

Neste editor, conforme ilustrado na Figura 9, é possível colocar as regras e alterá-las de forma mais simples. Para salvar o arquivo basta ir ao menu File – Save ou ainda utilizar o atalho Ctrl + S.

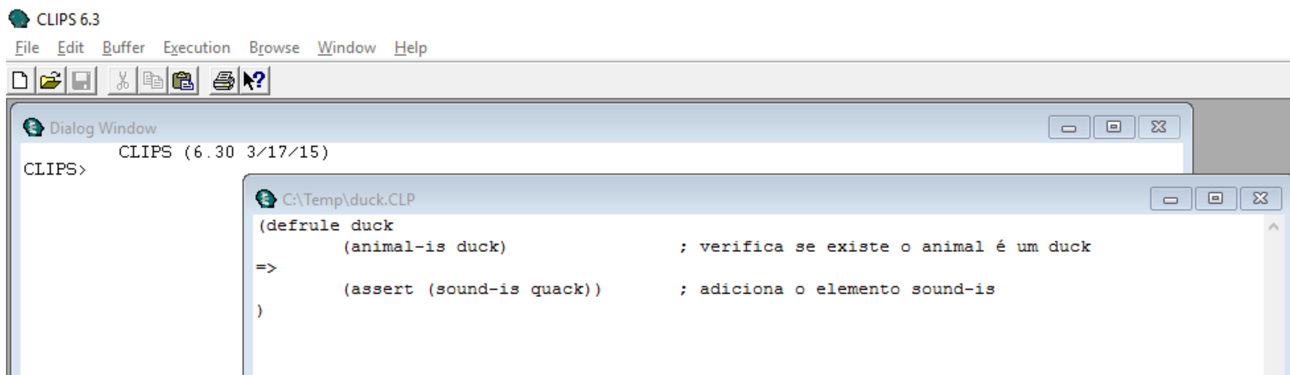


*Figura 9: Editor de programa.*

Para abrir um arquivo já criado para edição, basta ir ao menu File – Open.

Para carregar as regras no CLIPS a partir de um arquivo criado, basta ir até o menu File – Load e escolher o arquivo. As regras serão criadas como se estivesse utilizando a interface diretamente, conforme ilustrado pela Figura 10.

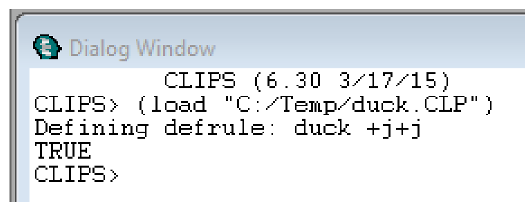


*Figura 10: Arquivo carregado.*

É possível carregar novamente as regras após editar o arquivo, as anteriores serão substituídas pelas regras alteradas.

## Mais regras

A utilização de SBRs faz mais sentido quando utilizamos mais de uma regra. Sistemas complexos contém milhares de regras.

Criar mais de uma regra em um programa auxilia também a entender o funcionamento de um SBR.

Vamos simular o programa de um robô, que deve atender às luzes de um semáforo para

andar ou parar. Desta forma vamos construir uma regra para que o robô ande quando a luz do semáforo for verde e outra regra para que o robô pare quando a luz do semáforo for vermelha.

Abra um novo arquivo para que as regras sejam criadas e adicione as regras conforme a seguir (não seja preguiçoso copiando e colando o texto, digite ele para que possa compreender melhor o seu funcionamento):

```
(defrule sinal-verde
     "Andar quando a luz do sinal for verde"
     (cor-luz verde)
=>
     (printout t "Ande!" crlf)     ;exiba ande na tela
)

(defrule sinal-vermelho
     "Parar quando a luz do sinal for vermelha"
     (cor-luz vermelha)
=>
     (printout t "Pare!" crlf)     ;exiba pare na tela
)
```

Após finalizar a edição do arquivo salve ele. Para carregar as regras utilize a opção Load no menu File e escolha o arquivo salvo. Caso tudo ocorra bem aparecerá o que está na Figura 11.

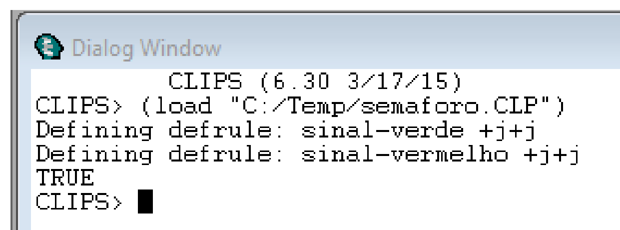



*Figura 11: Arquivo de regras carregado.*

Agora adicione um elemento na Base de Fatos com o nome cor-luz com valor verde.

Execute o comando `(run)` e verifique o resultado.

Depois adicione um elemento na Base de Fatos com o nome cor-luz com valor vermelha. Execute o comando `(run)` e verifique o resultado.

Importante mencionar que o fato de existir um elemento cor-luz com valor verde não faz com que a regra fique ativada diretamente, exibindo "Ande!" na tela. Após executar as ações da regra uma vez ela somente será executada novamente caso um novo elemento cor-luz com valor verde seja adicionado na Base de Fatos. Os elementos utilizados anteriormente não são considerados.

Para adicionar este novo elemento, o anterior deve ser excluído com o comando `retract`.

### Exercício 2

Exclua o elemento da Base de Fatos chamado cor-luz com valor verde. Adicione um novo fato chamado cor-luz com valor verde e execute o programa.

## Múltiplas condições em uma regra

É comum termos situações onde devemos combinar condições para que ações sejam executadas. Para isso basta adicionar quantas condições forem necessárias antes do `=>` , ou seja, na parte do LHS.

Isto é equivalente ao condicional E da lógica proposicional.

Por exemplo, podemos indicar que para que o robô pare é necessário que o sinal esteja vermelho e que o robô esteja andando. Para isso basta adicionar mais uma condição na regra, como um elemento chamado `andando` com o valor `sim`.

Este novo elemento pode ser adicionado quando a regra sinal-verde for executada.

As regras ficariam desta forma:

```
(defrule sinal-verde

      (cor-luz verde)

=>

      (printout t "Ande!" crlf)

      (assert (andando sim))

)
```

```
(defrule sinal-vermelho
	(cor-luz vermelha)
	(andando sim)
=>
	(printout t "Pare!" crlf)
)
```

# Aula 3 – Programas dinâmicos

## Solicitando dados ao usuário

Quando é necessário realizar iteração com o usuário, normalmente, um programa solicita ao usuário que ele informe alguns dados.

Em CLIPS, para ler a informação dada por um usuário utiliza-se o comando `read`. Para armazenar esta informação em um fato, utiliza-se o comando `read` em conjunto com o comando `assert`.

Por exemplo, para armazenar um valor informado pelo usuário em um fato chamado nome, utilizamos o comando: `(assert (nome (read)))`

## Verificando valores na base de fatos

Em algumas situações é necessário verificar se há valor atribuído a um fato, ou ainda, verificar qual valor está armazenado neste fato.

Para isso podemos utilizar alguns recursos em CLIPS:

a) Para verificar se um fato está sem valor, ou não existe na Base de Fatos, utiliza-se o `not`, exemplo: `(not (nome))`. Este comando pode ser colocado no LHS, ou seja, como uma condição, e significa que se não houver valor no fato nome a condição é verdadeira;

b) Para verificar se existe um valor qualquer armazenado em um fato, pode-se utilizar uma variável temporária, utilizando um ponto de interrogação antes do nome dela, exemplo `(nome ?n)`. Utilizando este comando como condição significa que a condição será verdadeira caso tenha qualquer valor armazenado no fato `nome`. Ainda, o valor que existir neste fato será armazenado na variável `?n` para utilização no bloco de ações (RHS);

c) Para comparar valores na base de fatos pode-se utilizar o comando `test`, por exemplo, `(test (>= 2 1))`. Utilizando este comando como condição significa que a condição será verdadeira caso o número 2 seja maior ou igual ao número 1. Estes números podem ser valores armazenados na Base de Fatos.

Para exemplificar a solicitação de dados ao usuário e a validação de valores, será

utilizado um programa simples para solicitar o nome de uma pessoa e expressar uma saudação.

Abra o CLIPS e crie um novo arquivo para edição, copie as regras demonstradas na Figura 12:

```
; Programa criado para apresentar uma saudação
; Criado por: Douglas Lusa Krug

(defrule bom-dia
        ; regra para saudar a pessoa
        (nome ?n)
=>
        (printout t "Bom dia " ?n "!" crlf)
)

(defrule i-nome
        ; regra para solicitar o nome da pessoa
        (not (nome))
=>
        (printout t "Informe o nome: " crlf)
        (assert (nome (read)))
)
```

*Figura 12: Programa saudação.*

Salve o arquivo e vá até a janela de diálogo (Dialog Window) do CLIPS, antes de carregar o programa utilize os comandos `(clear)` e `(reset)`. A utilização destes comandos antes de carregar o programa faz-se necessária para limpar o CLIPS.

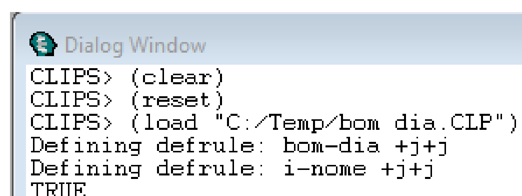



*Figura 13: Carregamento do programa.*

Caso tudo esteja ok aparecerá a mensagem `TRUE`. Após isso execute o comando `(run)`.

## Operações aritméticas

É possível realizar operações aritméticas no CLIPS, para isso utiliza-se os comandos a seguir:

- + para operação de soma, exemplo: `(+ 2 3)`
- - para operação de subtração, exemplo: `(- 6 3)`
- * para operação de multiplicação, exemplo: `(* 5 2)`

- / para operação de divisão, exemplo: `(/ 18 3)`

Caso necessário é possível combinar várias operações e também armazenar o valor em um fato, por exemplo, o comando a seguir está fazendo a média aritmética de 3 números e armazenando no fato chamado media: `(assert (media (/ (+ 6 7 9) 3)))`

Execute estes comandos diretamente no CLIPS para praticar e verifique o resultado.

Ao utilizar condições no CLIPS é possível realizar comparação entre valores com o comando `(test)`, para estas comparações os operadores relacionais a seguir podem ser utilizados:

- eq Compara se ambos os valores são iguais, caso sim, o valor da comparação é verdadeiro (todos os tipos);
- neq Compara os valores são distintos, caso sim, o valor da comparação é verdadeiro (todos os tipos);
- = Compara se ambos os valores são iguais, caso sim, o valor da comparação é verdadeiro (apenas números);
- <> Compara se os valores são diferentes, caso sejam, o valor da comparação é verdadeiro (apenas números);
- < Verdadeiro caso o primeiro valor seja menor do que o segundo valor;
- <= Verdadeiro caso o primeiro valor seja menor ou igual ao segundo valor;
- > Verdadeiro caso o primeiro valor seja maior do que o segundo valor;
- >= Verdadeiro caso o primeiro valor seja maior ou igual ao segundo valor;

Exemplo de utilização: `(test (>= valor1 10))`

## Exercício 3

Abra um novo arquivo no CLIPS e copie o programa da Figura 14. Este programa solicita 2 números ao usuário, retorna a soma destes 2 números e informa se a soma é maior do que 100.

```
; Programa exemplo de soma
; Criado por Douglas Lusa Krug

(defrule soma
        ; Caso existam os dois valores realiza a soma
        (valor1 ?v1)
        (valor2 ?v2)
=>
        (assert (soma (+ ?v1 ?v2)))
        (printout t "A soma é: " (+ ?v1 ?v2) crlf)
)

(defrule recebe-valores
        ; Solicita os valores
        (not (valor1))
        (not (valor2))
=>
        (printout t "Informe o valor 1:" crlf)
        (assert (valor1 (read)))
        (printout t "Informe o valor 2:" crlf)
        (assert (valor2 (read)))
)

(defrule maior-100
        ; Verifica se a soma é maior do que 100
        (soma ?s)
        (test (>= ?s 100))
=>
        (printout t "Soma maior que 100!" crlf)
)
```

*Figura 14: Exemplo soma.*

Note que a ordem das regras não é sequencial, pois são executadas a partir do momento que as condições (LHS) são satisfeitas.

Salve o arquivo e vá até a janela de diálogo (Dialog Window) do CLIPS, antes de carregar o programa utilize os comandos `(clear)` e `(reset)`.

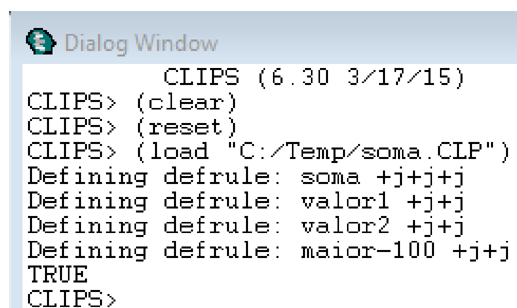



*Figura 15: Exemplo soma.*

Caso tudo esteja ok aparecerá a mensagem `TRUE`. Após isso execute o comando `(run)`.

### Exercício 4

Crie um programa em CLIPS que solicite o ano de nascimento de uma pessoa, calcule a idade dela com base no ano atual, e informe se a mesma tem obrigação de votar.

Utilize identação e comentários no programa.

# Aula 4 – Repetições

## Variáveis globais

Podemos utilizar variáveis globais em CLIPS. Elas são declaradas no início do programa com o comando `defglobal`. O nome da variável é precedido de `?*` e seguido de `*`. Exemplo: `?*numeros*`

Utiliza-se variáveis globais quando é necessário armazenar valores que mudam de valor no decorrer do programa e não devem ser utilizados para disparar regras.

As variáveis globais devem ser utilizadas quando é necessário esta alteração constante de valores, o que não poderia ser feito de forma fácil com fatos.

Para armazenar um valor a uma variável utiliza-se o comando `bind` seguido pelo nome da variável e pelo valor que será armazenado a ela. Exemplo `(bind ?*numeros* 10).`

## Referência para exclusão de um fato

Quando necessário é possível excluir um fato com o comando `retract`. Também é possível realizar o vínculo de um fato na parte de condições da regra, inclusive associando à existência de um valor ou a um valor específico.

Para isso utiliza-se uma variável para o endereço do fato, por exemplo,

```
?fn <- (num ?n)
```

Neste caso a regra está verificando se existe um elemento chamado `num` com qualquer valor. Está também armazenando o valor na varável `?n` e armazenando o endereço do fato na variável `?fn`, permitindo o a exclusão do fato com o comando `(retract ?fn).`

## Repetindo ações com CLIPS

Para executar ações de forma repetida em CLIPS não utilizamos uma estrutura de repetição "convencional" como, por exemplo, um o *for*, o *while* e o *do while*.

Através das regras e fatos é possível criar repetições de ações de forma direta.

Veja o exemplo a seguir, nele estamos solicitando ao usuário a quantidade de vezes que

uma determinada ação deve ser realizada (regra `inicializa`) e a partir do momento que temos a quantidade de vezes a regra `executa-loop` é acionada.

Como condições da regra `executa-loop` temos que deve existir um fato chamado `cont`, com qualquer valor, e o valor deste fato deve ser menor ou igual ao valor da variável `vezes`.

A cada execução das ações da regra `executa-loop` é realizada a remoção do fato `cont`, com a ação `retract`. Também é realizado a adição do fato `cont` com o incremento do valor através do comando `assert`.

O exemplo apresentado na Figura 16 demonstra a utilização de repetição com CLIPS, em conjunto com a utilização de variáveis globais e exclusão de fatos pelo endereço.

```
1  (defglobal
2       ?*vezes* = 0
3  )
4
5  (defrule inicializa
6       (not (cont))
7  =>
8       (printout t "Informe o numero de repeticoes: " crlf)
9       (bind ?*vezes* (read))
10      (assert (cont 1))
11 )
12
13 (defrule executa-loop
14      ?fc <- (cont ?c)
15      (test (<= ?c ?*vezes*))
16 =>
17      (printout t "Repeticao num.: " ?c crlf)
18      (retract ?fc)
19      (assert (cont (+ ?c 1)))
20 )
```

*Figura 16: Exemplo de repetições.*

Também é possível fazer a estrutura de repetição sem o `retract`. Desta forma os fatos com valores anteriores continuarão em memória. Para isso basta retirar o `?fc` ← na linha 14 e excluir a linha 18.

## Exercício 5

Crie um programa em CLIPS que solicite a quantidade de pessoas em uma família e na sequência solicite a idade de cada uma delas. Ao final o programa deve informar a média de idade delas.

# Referências

FRIEDMAN-HILL, E. Jess in Action: Rule Based System in Java. Greenwich, CT, USA: Manning Publications Co, 2003.

GIARRATANO, J. C. CLIPS 6.3 User's Guide. Gary Riley: 2015.

KRUG, D. L., SIMÃO, J. M., BASTOS, L. C. Comparativo entre o aprendizado de Programação baseado na abordagem Imperativo-Procedimental e na abordagem de Sistemas Baseados em Regras. Seminário I, PPGCA/DAINF/UTFPR, 2016.

# SEÇÃO COMPLEMENTAR F – QUESTIONÁRIOS APLICADOS AOS ALUNOS

## Questionário de Avaliação Completo

Questionário aplicado para as turmas da 1ª série para avaliar o experimento com PP e SBR e também para a 2ª série avaliando o experimento com SBR.

A sua colaboração no preenchimento correto do questionário é fundamental para esta pesquisa!

1) Idade:

2) Gênero: (   ) Masculino   (   ) Feminino

3) Você tinha conhecimento anterior de programação? (   ) Sim  (   ) Não

4) Você dedicou tempo fora dos momentos de aula para estudo da forma de programação utilizando o Paradigma Procedimental/Sistemas Baseados em Regras e da linguagem Pascal/CLIPS?
(   ) Sim  (   ) Não

5) Classifique a sua motivação de aprendizagem durante as aulas utilizadas para este experimento. A menor classificação é 1 e a maior é 5.
(   ) 1  (   ) 2  (   ) 3  (   ) 4  (   ) 5

6) Classifique a facilidade de utilização da ferramenta Pascal/CLIPS utilizada durante as aulas deste experimento. A menor classificação é 1 e a maior é 5.
(   ) 1  (   ) 2  (   ) 3  (   ) 4  (   ) 5

7) Classifique a clareza das explicações feitas pelo professor durante as aulas deste experimento. A menor classificação é 1 e a maior é 5.

( ) 1 ( ) 2 ( ) 3 ( ) 4 ( ) 5

8) Classifique a clareza do material de apoio (apostila) utilizado durante as aulas deste experimento. A menor classificação é 1 e a maior é 5.
( ) 1 ( ) 2 ( ) 3 ( ) 4 ( ) 5

9) Classifique a utilidade dos exemplos aplicados durante as aulas deste experimento para o aprendizado da linguagem de programação Pascal/CLIPS. A menor classificação é 1 e a maior é 5.
( ) 1 ( ) 2 ( ) 3 ( ) 4 ( ) 5

10) Classifique a facilidade dos exercícios utilizados durante as aulas deste experimento. A menor classificação é 1 e a maior é 5.
( ) 1 ( ) 2 ( ) 3 ( ) 4 ( ) 5

11) Classifique a facilidade do exercício final utilizado neste experimento. A menor classificação é 1 e a maior é 5.
( ) 1 ( ) 2 ( ) 3 ( ) 4 ( ) 5

12) Classifique a forma de programação utilizando o Paradigma Procedimental/Sistemas Baseados em Regras através da linguagem Pascal/CLIPS quanto ao entendimento do código gerado. A menor classificação é 1 e a maior é 5.
( ) 1 ( ) 2 ( ) 3 ( ) 4 ( ) 5

13) Classifique a forma de programação utilizando o Paradigma Procedimental/Sistemas Baseados em Regras através da linguagem Pascal/CLIPS quanto a facilidade de programação. A menor classificação é 1 e a maior é 5.
( ) 1 ( ) 2 ( ) 3 ( ) 4 ( ) 5

14) Classifique a sua experiência no geral quanto ao experimento durante a realização deste. A menor classificação é 1 e a maior é 5.
( ) 1 ( ) 2 ( ) 3 ( ) 4 ( ) 5

**Questionário de Avaliação Adaptado**

Questionário aplicado para a turma da 2ª série avaliando o experimento com PP.

A sua colaboração no preenchimento correto do questionário é fundamental para esta pesquisa!

1) Idade:

2) Gênero: (   ) Masculino   (   ) Feminino

3) Você tinha conhecimento de programação antes de iniciar o curso? (   ) Sim (   ) Não

4) Classifique a facilidade de utilização da ferramenta (Code Blocks) utilizada durante as aulas. A menor classificação é 1 e a maior é 5.
(   ) 1  (   ) 2  (   ) 3  (   ) 4  (   ) 5

5) Classifique a facilidade do exercício utilizado neste experimento. A menor classificação é 1 e a maior é 5.
(   ) 1  (   ) 2  (   ) 3  (   ) 4  (   ) 5

6) Classifique a forma de programação utilizando o Paradigma Procedimental através da linguagem C quanto ao entendimento do código gerado. A menor classificação é 1 e a maior é 5.
(   ) 1  (   ) 2  (   ) 3  (   ) 4  (   ) 5

7) Classifique a forma de programação utilizando o Paradigma Procedimental através da linguagem C quanto à facilidade de programação. A menor classificação é 1 e a maior é 5.
(   ) 1  (   ) 2  (   ) 3  (   ) 4  (   ) 5

8) Classifique a sua experiência no geral quanto ao experimento durante a realização deste. A menor classificação é 1 e a maior é 5.

( ) 1 ( ) 2 ( ) 3 ( ) 4 ( ) 5