



Pós-Graduação em Ciência da Computação

Viviane Cristina Oliveira Aureliano

Stepwise self-explanation: uma abordagem para o aprendizado de linguagens de programação através da auto-explicação de exemplos em vídeo



Universidade Federal de Pernambuco
posgraduacao@cin.ufpe.br
www.cin.ufpe.br/~posgraduacao

RECIFE
2016

Viviane Cristina Oliveira Aureliano

***Stepwise self-explanation: uma abordagem para o aprendizado de linguagens
de programação através da auto-explicação de exemplos em vídeo***

Este trabalho foi apresentado à Pós-Graduação em
Ciência da Computação do Centro de Informática da
Universidade Federal de Pernambuco como requisito
parcial para obtenção do grau de Doutora em Ciência da
Computação.

ORIENTADORA: Prof. Dra. Patrícia Cabral de Azevedo Restelli Tedesco

RECIFE
2016

Catálogo na fonte
Bibliotecária Monick Raquel Silvestre da S. Portes, CRB4-1217

A927s Aureliano, Viviane Cristina Oliveira
Stepwise self-explanation: uma abordagem para o aprendizado de linguagens de programação através da auto-explicação de exemplos em vídeo / Viviane Cristina Oliveira Aureliano. – 2016.
287 f.:il., fig., tab.

Orientadora: Patrícia Cabral Azevedo Restelli Tedesco.
Tese (Doutorado) – Universidade Federal de Pernambuco. CIn, Ciência da Computação, Recife, 2016.
Inclui referências e apêndices.

1. Inteligência artificial. 2. Educação. 3. Tecnologia educacional. I. Tedesco, Patrícia Cabral Azevedo Restelli (orientadora). II. Título.

006.3 CDD (23. ed.) UFPE- MEI 2017-169

Viviane Cristina Oliveira Aureliano

Stepwise Self-explanation: Uma Abordagem para o Aprendizado de Linguagens de Programação Através da Auto-explicação de Exemplos em Vídeo

Tese de Doutorado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Doutora em Ciência da Computação

Aprovado em: 26/02/2016.

Orientador: Profa. Patricia Cabral de Azevedo Restelli Tedesco

BANCA EXAMINADORA

Profa. Dra. Ana Carolina Brandão Salgado
Centro de Informática / UFPE

Profa. Dra. Carla Taciana Lima Lourenço Silva Schuenemann
Centro de Informática / UFPE

Profa. Dra. Simone Cristiane dos Santos Lima
Centro de Informática / UFPE

Profa. Dra. Lucia Maria Martins Giraffa
Faculdade de Educação / PUC/RS

Prof. Dr. Sergio Crespo Coelho da Silva Pinto
Instituto de Ciência e Tecnologia / UFF

Agradecimentos

A minha mãe Vera Oliveira, pelo amor incondicional. Ao meu marido Ricardo Lins, por todo amor, companheirismo e compreensão nesta jornada. A minha tia Edna Oliveira, pelo apoio constante. Ao meu irmão Vinícius Oliveira e a sua família, pelo entendimento nos momentos de ausência. E a meu cachorro Marley, por simplesmente fazer parte da minha vida.

Aos meus amigos e amigas dos diferentes círculos de amizade, pelas muitas conversas, pela compreensão pelo isolamento e pela partilha de momentos incríveis. A todos os companheiros de doutorado que de alguma forma contribuíram neste trabalho. E aos meus colegas de trabalho do IFPE – Campus Belo Jardim, pelo apoio e empenho para que o meu processo de afastamento fosse efetivado.

A minha orientadora professora Patrícia Tedesco e ao meu orientador do doutorado sanduíche professor Michael Caspersen, pela disposição em me orientar desde o meu primeiro contato com ambos, pelos momentos constantes de aprendizado, pelo apoio nos momentos difíceis e pela confiança depositada. Eu tive muita sorte em tê-los como meus orientadores.

A banca examinadora desta tese, os professores Ana Carolina Salgado, Carla Schuenemann, Simone Lima, Lucia Giraffa e Sérgio Crespo pelas excelentes contribuições fornecidas.

Aos estudantes e professores participantes dos experimentos realizados nesta tese, pelo tempo dedicado, pelas opiniões fornecidas e pelas experiências trocadas.

E finalmente a CAPES pelo suporte financeiro para o período do Doutorado Sanduíche.

Resumo

Começar a programar é considerado difícil pela maioria dos estudantes. Normalmente, as disciplinas introdutórias de programação estão relacionadas a altas taxas de reprovação e de evasão. Ensinar programação é considerado um dos grandes desafios da área de educação em computação. Aliado a isso, os estudantes de programação adotam metodologias de estudo que são ineficientes. Eles decoram soluções para os problemas ao invés de entendê-las, estudam de maneira passiva ou superficial e fazem poucos exercícios. Estes estudantes também possuem um tipo de conhecimento dito frágil, que é aquele conhecimento que eles possuem, mas não conseguem aplicar em novas situações ou problemas encontrados. Para minimizar as dificuldades dos iniciantes em programação, os professores devem apresentar o processo de programação aos estudantes, guiando-os durante a construção dos programas. Uma maneira de fazer isso é através da utilização do *Stepwise Improvement* (FSI), um *framework* conceitual que descreve a atividade de programação como um processo sistemático e incremental composto pelas atividades de extensão, refinamento e reestruturação. A partir do emprego do FSI para a construção de materiais instrucionais, os estudantes podem aprender programação desenvolvendo pequenos pedaços de código sistemática e incrementalmente. Além disso, os professores devem propor atividades de aprendizado que permitam que os estudantes empreguem metodologias de estudo que sejam mais eficazes. Neste caso, uma atividade de aprendizado considerada eficiente para os estudantes é a auto-explicação. A auto-explicação é uma atividade construtiva que auxilia os estudantes a entender o que está sendo estudado e a construir conhecimento a partir deste material. Auto-explicar durante os momentos de estudo mostrou ser eficiente para estudantes estudando a partir de diferentes materiais instrucionais e em diversas disciplinas. A prática da auto-explicação foi pouco explorada na área de Programação, tendo sido aplicada, por exemplo, no estudo de exemplos apresentados estaticamente aos estudantes. Contudo, a literatura não relata a utilização da auto-explicação para o ensino do processo de construção de programas aos estudantes. No intuito de preencher esta lacuna, o objetivo principal desta tese é propor uma abordagem chamada *Stepwise Self-explanation*. Nesta abordagem, lançamos mão do FSI para estruturar exemplos trabalhados em vídeo utilizados como material instrucional de cursos introdutórios de programação. Adicionalmente, utilizamos a técnica da auto-explicação como atividade de

aprendizagem e como forma de orientar o processo de reflexão dos estudantes. Propusemos duas versões da abordagem, uma versão piloto e uma versão refinada. A versão piloto da abordagem foi avaliada por meio de um experimento piloto com estudantes matriculados no ensino médio e técnico. Os resultados deste experimento mostraram que uma abordagem promissora, mas que precisava de melhorias em sua estrutura. A versão refinada da abordagem foi avaliada por meio de três quase-experimentos com estudantes matriculados no ensino médio, técnico e superior. Os resultados obtidos nestes quase-experimentos foram encorajadores. Eles mostraram que os estudantes que fizeram uso da abordagem cometeram menos erros em seus programas do que os outros estudantes, resultando em um melhor desempenho nos pós-testes. A abordagem também obteve um *feedback* positivo dos estudantes e dos professores que a avaliaram.

Palavras-chave: Educação em Ciência da Computação. Ensino e aprendizagem de programação. Iniciantes. Stepwise Improvement. Auto-explicação. Exemplos trabalhados em vídeo.

Abstract

Most beginners find it difficult to start programming. Typically, introductory programming disciplines have high failure and dropout rates. Teaching programming is considered to be one of the greatest challenges in Computer Education. In addition, programming students tend to adopt inefficient study methodologies. They memorize problem solutions rather than understand them, study passively or superficially, and do few exercises. To minimize novices' difficulties, teachers should reveal the programming process to students, guiding them carefully while they are building programs. One way of doing so is through the use of the Stepwise Improvement (FSI), a *framework* that describes programming as a systematic and incremental process consisting of extension, refinement and restructuring activities. By using FSI, novices learn programming by developing small pieces of code in a systematic and incremental way. Besides that, teachers should propose learning activities that allow students to employ more effective study methodologies. In this situation, an evidence-based learning activity is self-explanation. Self-explanation is a constructive activity that helps students understand and build knowledge from instructional material. Self-explaining while studying has proven to be efficient for students studying from different instructional materials in various subjects. Although recommended as one of the seven practices for improving teaching and learning, self-explanation has been little explored in Programming Education. In fact, the literature does not report the use of self-explanation together with the process of building programs. In order to fill in this gap, the main goal of this thesis is to propose an approach called Stepwise Self-explanation. In this approach, we use FSI to structure examples presented in video as instructional material for introductory programming courses. Besides that, we use the self-explanation technique as a learning activity and as a way to guide the students' reflection process. We proposed two versions of the approach, a pilot version and a refined version. The pilot version was evaluated through an experiment with high-school and technical students. The results of this experiment showed that the approach was promising, but needed improvements in its structure. The refined version of the approach was evaluated through three quasi-experiments with high-school, technical and university students. The results were very encouraging. They showed that students who used the approach made fewer mistakes in their programs, and performed better in

the post-tests. The approach also received positive feedback from students and teachers who evaluated it.

Keywords: Computer Science Education. Teaching and learning programming. Novices. Stepwise Improvement. Self-explanation. Worked examples in video.

Lista de figuras

Figura 1.1. Organização geral da tese.....	24
Figura 2.1. Distribuição temporal dos artigos.	42
Figura 2.2. Distribuição geográfica das instituições de pesquisa.	43
Figura 2.3. Níveis de escolaridade para os quais as pesquisas são destinadas.	46
Figura 2.4. Tipo de pesquisa realizada.	47
Figura 3.1. Um exemplo de sequência de desenvolvimento de acordo com o FSI.	55
Figura 3.2. Representação do FSI utilizando a notação BPMN.	56
Figura 3.3. Objetivo principal da Teoria da carga cognitiva.	64
Figura 4.1. Imagem de tela de trecho do exemplo em vídeo <i>The Joy of Code</i>	95
Figura 4.2. Representação diagramática das atividades de extensão e refinamento do FSI.	96
Figura 4.3. Representação diagramática da abordagem <i>Stepwise self-explanation</i>	100
Figura 4.4. Versão refinada da abordagem <i>Stepwise self-explanation</i>	110
Figura 4.5. Trecho de exemplo em vídeo, apresentando passo da implementação do problema (c) juntamente com as questões anterior (a) e posterior (b) à implementação do passo do problema.	116
Figura 4.6. <i>Workflow</i> de utilização da Stepwise Self-explanation.	118
Figura 5.1. Sessão online.....	122
Figura 5.2. Sessão em sala de aula.	124
Figura 5.3. Primeira aula extra (a). Segunda aula extra (b).....	130
Figura 5.4. Primeira aula extra (a). Segunda e terceira aulas extras (b). Quarta aula extra (c).....	134
Figura 5.5. Estrutura geral do terceiro experimento (a). Momentos de aula expositiva seguidos por estudo do exemplo trabalhado em vídeo durante o minicurso (b).	137
Figura 5.6. Etapas de análise dos programas no pré e pós-testes.	139
Figura 6.1. Histograma do GI.....	152
Figura 6.2. Histograma do GC.	152
Figura 6.3. Médias de erros e avisos detectados pelo compilador para GC, GI1 e GI2 nos pré e pós-testes.....	158
Figura 6.4. Médias de erros não detectados pelo compilador para GC, GI1 e GI2 nos pré e pós-testes.....	158
Figura 6.5. Médias de erros não detectados pelo compilador para GC (a), GI1 (b) e GI2 (c) nos pré e pós-testes por categoria de erro.	160
Figura 6.6. Médias de metas certas (a), erradas (b) e omitidas (c) para GC, GI1 e GI2 nos pré e pós-testes.....	161
Figura 6.7. Notas obtidas pelos GC, GI1 e GI2 no pré-teste e no pós-teste.....	164

Figura 6.8. Evolução dos erros não gerados pelo compilador pelo E1 nas 4 aulas extras.	169
Figura 6.9. Evolução dos erros não gerados pelo compilador pelo E2 nas 4 aulas extras.	170
Figura 6.10. Quantidade de metas certas (a), erradas (b) e omitidas (c) para o E1 nos pré e pós-testes ao longo das aulas.	172
Figura 6.11. Quantidade de metas certas (a), erradas (b) e omitidas (c) para o E2 nos pré e pós-testes ao longo das aulas.	173
Figura 6.12. Notas obtidas pelos estudantes nos pré e pós-testes ao longo das aulas extras.....	174
Figura 6.13. Médias dos erros cometidos pelos GC, GI1 e GI2 nos pré e pós-testes...	181
Figura 6.14. Médias de erros cometidos pelos GC (a), GI1 (b) e GI2 (c) no pré e no pós-teste por categoria de erros.	183
Figura 6.15. Médias de metas certas (a), erradas (b) e omitidas (c) para GC, GI1 e GI2 no pré e pós-testes.	184
Figura 6.16. Notas obtidas pelos GC, GI1 e GI2 no terceiro experimento.	186

Lista de tabelas

Tabela 2.1. Critérios de exclusão e de inclusão.....	40
Tabela 2.2. Resultado geral das buscas para o SBIE, WIE, WEI e RBIE.....	41
Tabela 2.3. Quantidade de artigos por instituições de pesquisa.	42
Tabela 2.4. Tópicos abordados nos artigos.....	44
Tabela 2.5. Natureza dos artigos.	44
Tabela 2.6. Escopo dos artigos.	45
Tabela 2.7. Técnicas de coleta de dados apresentadas nos artigos.....	47
Tabela 6.1. Como os alunos do GC e do GI utilizaram os vídeos enquanto estudavam algum assunto.	150
Tabela 6.2. Como os estudantes dos GC, GI1 e GI2 estudam para a disciplina de IP.	155
Tabela 6.3. Como os estudantes dos GC, GI1 e GI2 lembram de ter utilizado vídeos para estudar para a disciplina de IP.	156
Tabela 6.4. Opiniões dos GC, GI1 e GI2 sobre os exemplos em vídeo.	165
Tabela 6.5. Opiniões dos GI1 e GI2 sobre as questões para AE nos exemplos em vídeo.	166
Tabela 6.6. Opiniões dos professores sobre o material instrucional.	167
Tabela 6.7. Como os estudantes do GI estudam para a disciplina de IP.	167
Tabela 6.8. Como um dos estudantes do GI lembra de ter utilizado vídeos para estudar para a disciplina de IP.....	168
Tabela 6.9. Opiniões do GI sobre os exemplos em vídeo.	176
Tabela 6.10. Opiniões do GI sobre as questões para AE nos exemplos em vídeo.	176
Tabela 6.11. Opiniões dos professores a respeito do material instrucional.....	177
Tabela 6.12. Como os estudantes dos GC e GI1 estudam para a disciplina de IP e como os estudantes do GI2 estudam para as disciplinas em geral.	178
Tabela 6.13. Como os estudantes dos GC e GI1 lembram de ter utilizado vídeos para estudar para a disciplina de IP e como os estudantes do GI2 lembram de ter utilizado vídeos para as disciplinas em geral.....	179
Tabela 6.14. Opiniões dos GC, GI1 e GI2 sobre os exemplos em vídeo.	187
Tabela 6.15. Opiniões dos GI1 e GI2 sobre as questões para AE nos exemplos em vídeo.	188
Tabela 6.16. Opiniões dos professores sobre o material instrucional.	188

Lista de quadros

Quadro 4.1. Objetivos de aprendizagem que são alcançados ao se programar a partir do FSI.	94
Quadro 4.2. Questões para a auto-explicação relacionadas ao vídeo do The Joy of Code.	100
Quadro 4.3. Objetivos de aprendizagem que são alcançados ao se programar a partir do FSI.	103
Quadro 4.4. Script do exemplo de exemplo trabalhado em vídeo.....	104
Quadro 4.5. Script do exemplo de exemplo trabalhado em vídeo contendo os momentos de auto-explicação.	111
Quadro 5.1. Tópicos e objetivos de aprendizagem ministrados nas aulas extras do primeiro experimento.	128
Quadro 5.2. Tópicos e objetivos de aprendizagem ministrados nas aulas extras do segundo experimento.....	133
Quadro 5.3. Tópicos e objetivos de aprendizagem ministrados no minicurso do terceiro experimento.	138
Quadro 5.4. Pós-teste utilizado na segunda aula extra do primeiro quase-experimento.	140
Quadro 5.5. Implementação de um dos participantes do primeiro quase-experimento para o pós-teste do Quadro 5.4.	143

Lista de abreviaturas e siglas

FSI – *Framework Stepwise Improvement*

SSE – *Stepwise Self-explanation*

QP – Questão de pesquisa

GC – Grupo controle

GI – Grupo instrucional

TCLE – Termo de consentimento livre e esclarecido

AE – Auto-explicação

Sumário

1	Introdução	17
1.1	Problema	19
1.2	Hipótese	20
1.3	Objetivos	21
1.4	Metodologia	22
1.5	Contribuições	22
1.6	Estrutura da tese	23
2	Os processos de ensino e de aprendizagem de programação	25
2.1	Aprender a programar é difícil.....	25
2.2	Ensinar a programar também é difícil.....	30
2.3	O longo processo para se passar de iniciante a experiente	36
2.4	Literatura relacionada ao ensino e à aprendizagem de programação para iniciantes no Brasil.....	38
2.4.1	Questões de pesquisa do mapeamento	39
2.4.2	CrITÉrios de inclusão e exclusão.....	39
2.4.3	Extração e síntese dos dados.....	40
2.4.4	Resultados	41
2.4.5	Distribuição temporal.....	41
2.4.6	Distribuição geográfica das instituições de pesquisa	42
2.4.7	Tópicos e escopo dos artigos	43
2.4.8	Nível de escolaridade	45
2.4.9	Tipo de pesquisa e de análise de dados realizados.....	46
2.4.10	Técnica de coleta de dados.....	47
2.4.11	Principais objetivos e resultados obtidos	48
2.5	Implicações	50
3	Referencial teórico	53
3.1	Framework Stepwise Improvement	53
3.2	Teoria da carga cognitiva.....	58
3.2.1	Exemplos trabalhados	65
3.3	Auto-explicação	67
3.4	Aprendizagem cognitiva	73
3.5	Trabalhos relacionados	76

3.5.1	Utilizando o framework stepwise improvement	76
3.5.2	Utilizando vídeos para apoiar o ensino e a aprendizagem de programação ..	77
3.5.3	Utilizando exemplos trabalhados no ensino e na aprendizagem de programação.....	79
3.5.4	Utilizando a auto-explicação no ensino e na aprendizagem de programação	81
3.5.5	Utilizando a aprendizagem cognitiva no ensino e na aprendizagem de programação.....	84
3.6	Implicações	87
4	Stepwise self-explanation.....	90
4.1	Motivação	90
4.2	Descrição da versão piloto da abordagem proposta.....	92
4.2.1	Exemplos trabalhados em vídeo	94
4.2.2	Momentos de auto-explicação	97
4.3	Descrição da versão refinada da abordagem proposta	102
4.3.1	Exemplos trabalhados em vídeo	102
4.3.2	Momentos de auto-explicação	107
4.3.3	Utilização da Stepwise Self-explanation.....	116
4.4	Implicações	118
5	Procedimentos metodológicos	120
5.1	Experimento piloto.....	120
5.1.1	Participantes	121
5.1.2	Materiais instrucionais	121
5.1.3	Execução	122
5.1.4	Análise dos dados	125
5.2	Quase-experimentos.....	125
5.2.1	Primeiro quase-experimento	125
5.2.2	Segundo quase-experimento	130
5.2.3	Terceiro quase-experimento.....	134
5.2.4	Análise dos dados	139
5.3	Restrições dos experimentos.....	147
5.4	Implicações	147
6	Resultados obtidos	150
6.1	Experimento piloto.....	150
6.1.2	Discussão dos resultados.....	153
6.2	Quase-experimentos.....	155
6.2.1	Primeiro quase-experimento	155

6.2.2	Segundo quase-experimento	167
6.2.3	Terceiro quase-experimento.....	177
6.2.4	Discussão dos resultados.....	189
6.3	Implicações	193
7	Considerações finais.....	195
7.1	Contribuições	195
7.2	Trabalhos futuros	198
7.2.1	Refinamento dos procedimentos metodológicos para avaliação da Stepwise Self-explanation	198
7.2.2	Introdução de um outro momento na abordagem para discussão em pares e/ou grupos	199
7.2.3	Auto-explicação de exemplos trabalhados alternativos	199
7.2.4	Auto-explicação de exemplos trabalhados com erros.....	200
7.2.5	Envolvimento dos estudantes no processo de gravação dos exemplos trabalhados em vídeo	200
7.2.6	Sistema tutor inteligente para propor perguntas de acordo com o nível de conhecimento e do contexto da atividade de aprendizado do estudante.....	200
7.2.7	Relatório diagnóstico dos estudantes como forma de auxílio ao processo de avaliação pelo professor.....	201
7.2.8	Material instrucional para utilização da abordagem por outros professores de programação.....	201
7.2.9	Generalizar a abordagem para utilização em outras disciplinas	201
	Referências.....	203
	Apêndice A – Termo de consentimento	215
	Apêndice B – Pré-testes e pós-testes utilizados.....	216
	Primeiro quase-experimento (2 aulas extras).....	216
	Segundo quase-experimento (4 aulas extras).....	220
	Terceiro quase-experimento (minicurso).....	228
	Apêndice C – Artigos publicados.....	230

1 Introdução

Disciplinas introdutórias em linguagem de programação (LP) são intrinsecamente difíceis [ROBINS; ROUNTREE; ROUNTREE, 2003], comumente relacionadas a altos índices de reprovação [BENNEDSEN; CASPERSEN, 2007; WATSON; LI, 2014] e, por estarem nos semestres iniciais de muitos cursos na área de tecnologia da informação, são possivelmente causas dos altos índices de evasão nestes cursos [KINNUNEN; MALMI, 2006; SILVA FILHO et al., 2007]. Ademais, estudantes podem ser aprovados em uma disciplina de programação e mesmo assim possuírem problemas sérios ao aplicar os conceitos de programação [MCCRACKEN et al., 2001]. Em razão disso, ensinar programação é considerado um dos grandes desafios na área de educação em computação [CASPERSEN, 2007; CASPERSEN; KÖLLING, 2009].

De fato, os obstáculos para os estudantes nestas disciplinas são vários. Estudantes possuem dificuldades com as diversas estruturas de linguagem de programação, tais como variáveis [PUTNAM et al., 1989; SAMURÇAY, 1989], estruturas condicionais e de repetição [CHERENKOVA; ZINGARO; PETERSEN, 2014; PUTNAM et al., 1989; SPOHRER; SOLOWAY; POPE, 1985] e arrays [DU BOULAY, 1989]. Conceitos tais como recursão e ponteiros também são considerados difíceis pelos estudantes [LAHTINEN et al., 2005; MILNE; ROWE, 2002]. No entanto, o maior dos problemas vivenciados pelos iniciantes não parece ser o entendimento dos conceitos básicos de programação, mas a combinação e a utilização adequada destes conceitos na construção de um determinado programa [CASPERSEN; KÖLLING, 2009; LAHTINEN et al., 2005; ROBINS et al., 2003; WINSLOW, 1996]. Sendo assim, a maioria dos erros cometidos pelos estudantes não resulta de falhas no conhecimento sobre os construtos de uma linguagem de programação, mas das dificuldades em combiná-las corretamente, ou seja, “*em colocar as peças do programa juntas*” [SPOHRER; SOLOWAY, 1989 p. 401].

Para minimizar as dificuldades encontradas pelos iniciantes em programação, os professores devem apresentar o processo de programação aos estudantes, guiando-os cuidadosamente durante a construção de programas. Uma maneira de fazer isso é por meio da utilização do *Stepwise improvement* (FSI), um *framework* conceitual que descreve a atividade de programação como um processo sistemático e incremental que engloba três atividades diferentes, extensão, refinamento e reestruturação [CASPERSEN,

2007; CASPERSEN; KÖLLING, 2009]. A atividade de extensão ocorre quando a especificação é estendida de maneira a cobrir mais casos (de uso). A atividade de refinamento ocorre quando o código abstrato é modificado de maneira a se construir código executável que implemente a especificação correspondente. A atividade de reestruturação ocorre quando é realizada uma melhoria dos aspectos não funcionais do programa, no entanto essa modificação não envolve uma mudança no comportamento aparente do programa. Por meio da utilização de material instrucional que é estruturado de acordo com este *framework*, os estudantes iniciantes em programação podem aprender programação desenvolvendo pequenos pedaços de código sistemática e incrementalmente.

Dentro deste âmbito, os vídeos possuem o formato ideal para exibição da construção de programas para iniciantes em uma disciplina de programação [BENNEDSEN; CASPERSEN, 2008]. Diferentemente dos livros que são estáticos e apresentam somente os programas resultantes, os vídeos têm a vantagem de poder combinar a apresentação dos programas com todo o seu processo dinâmico de construção, o que é fundamental para a construção incremental proposta pelo FSI.

Aliado a isso, os exemplos trabalhados são uma sequência de passos que demonstram a realização de uma tarefa em particular ou a solução de um problema específico [CLARK; NGUYEN; SWELLER, 2005]. Eles fornecem uma solução para um problema que foi produzida por um profissional experiente e que pode ser utilizada pelos estudantes em seus momentos de estudo [ATKINSON et al., 2000]. Exemplos são considerados, tanto por estudantes quanto por professores, o tipo de material instrucional mais útil para os processos de ensino e de aprendizagem em linguagem de programação [LAHTINEN et al., 2005]. Ademais, os exemplos trabalhados foram recomendados pelo Departamento de Educação dos EUA como uma prática para melhoria dos processos de ensino e aprendizagem [PASHLER et al., 2007].

Entretanto, muitos estudantes de programação estudam de maneira inadequada para as necessidades de uma disciplina de programação [GOMES; MENDES, 2007]. Eles empregam metodologias incorretas enquanto estudam para a disciplina e, por não engajarem-se apropriadamente em seus momentos de estudo, eles acabam por estudar os materiais instrucionais disponíveis de maneira passiva ou superficial [CHIU; CHI, 2014; CLARK et al., 2005]. Em consequência, estes estudantes acabam apresentando resultados insatisfatórios nas disciplinas de programação.

Portanto, a estrutura destes exemplos sozinha não é suficiente para fazer com que os estudantes iniciantes aprendam. Sendo assim, além da orientação fornecida pelo FSI para a estrutura dos exemplos trabalhados, os estudantes iniciantes em programação devem ser orientados enquanto estão aprendendo o processo de programação. Uma prática comprovadamente eficiente para melhorar o aprendizado dos estudantes enquanto eles estudam a partir de diferentes tipos de material instrucional é através da auto-explicação [CHI, 2000; CHI et al., 1989; CHI et al., 1994]. A auto-explicação é uma atividade de aprendizado desempenhada pelos estudantes enquanto estudam um exemplo trabalhado, que os auxilia no processo de entendimento do exemplo e na construção de conhecimento a partir desse exemplo [CLARK et al., 2005]. De acordo com Chiu e Chi (2014), a auto-explicação promove o aprendizado a partir da elaboração da informação sendo estudada. Ademais, a auto-explicação promove a associação desta nova informação ao conhecimento prévio dos estudantes e também permite que eles construam novo conhecimento a partir da conexão de diferentes pedaços da informação fornecida. Similarmente aos exemplos trabalhados, a auto-explicação foi recomendada pelo Departamento de Educação dos EUA como umas das sete práticas para melhoria dos processos de ensino e aprendizagem [PASHLER et al., 2007].

1.1 Problema

Considerando este contexto, dois pontos merecem destaque. Primeiramente vimos que os iniciantes em programação passam por vários problemas, contudo o maior dos seus obstáculos não é o entendimento dos conceitos básicos de programação, mas a combinação e utilização adequada destes conceitos na construção de programas. Sendo assim, a maioria dos erros dos estudantes não resultam de falhas no conhecimento dos construtos de uma linguagem de programação, mas resultam das dificuldades em não conseguir juntar as peças do programa. Segundo, os estudantes de programação estudam de maneira inadequada para as necessidades de uma disciplina de programação, empregando metodologias incorretas de estudo que fazem com eles estudem o material instrucional disponível de maneira passiva ou superficial. Por exemplo, em diversas situações eles se limitam a decorar a solução para um problema ao invés de procurar entender como aquela solução foi construída. Em consequência, tais estudantes acabam passando por disciplinas introdutórias de programação sem adquirir adequadamente os conhecimentos básicos necessários para a sua evolução enquanto programadores.

Dentro deste âmbito, a presente tese objetiva definir uma abordagem para ensino e aprendizagem de programação para estudantes iniciantes que minimize os dois problemas destacados: (i) a dificuldade de combinar as diferentes estruturas de uma linguagem de programação para construir programas sintática, semântica e logicamente corretos; (ii) a maneira inadequada (ou insuficiente) que os estudantes estudam para uma disciplina introdutória de programação. Para fazer isso, propomos a definição de uma abordagem para o aprendizado de programação que estimule e oriente os estudantes iniciantes enquanto eles aprendem o processo de construção de programas descrito pelo FSI a partir de exemplos trabalhados em vídeo. Sendo assim, definimos a questão de pesquisa que norteia esta tese é a seguinte **QP**:

QP: como estimular e orientar o processo de aprendizagem dos estudantes iniciantes em programação enquanto eles estudam o processo de construção de programas a partir de exemplos trabalhados em vídeo?

1.2 Hipótese

Como mencionado anteriormente, a auto-explicação é uma das práticas recomendadas para a melhoria dos processos de ensino e de aprendizagem [PASHLER et al., 2007]. De fato, a atividade de auto-explicar durante os momentos de estudo mostrou ser eficiente para estudantes estudando a partir de diferentes materiais instrucionais (*i.e.*, textos [CHI et al., 1994; MCNAMARA, 2004; WYLIE; KOEDINGER; MITAMURA, 2010] e exemplos estáticos [BERTHOLD; RENKL, 2005a; CRIPPEN; EARL, 2007]) e em diversas disciplinas (*e.g.*, Biologia [CHI, 2000; CHI et al., 1994], Física [CHI et al., 1989] e Matemática [WONG; LAWSON; KEEVES, 2002]). No entanto, a prática da auto-explicação ainda é pouco explorada na área de Programação [SKUDDER; LUXTON-REILLY, 2014], tendo sido aplicada principalmente para o estudo de exemplos apresentados estaticamente aos estudantes [BIELACZYK; PIROLI; BROWN, 1995; PIROLI; RECKER, 1994]. Contudo, a literatura não relata a utilização da auto-explicação para o ensino do processo de construção de programas aos estudantes.

Para preencher esta lacuna, propomos responder a questão de pesquisa definida na Seção 1.1 lançando mão da auto-explicação como atividade de aprendizagem para estimular e orientar os estudantes iniciantes enquanto eles aprendem o processo de

programação descrito pelo FSI a partir de exemplos trabalhados em vídeo. Sendo assim, definimos a seguinte hipótese **Hip**:

Hip: se estimularmos e orientarmos o processo de auto-explicação dos estudantes iniciantes enquanto eles estudam o processo de construção de programas de acordo com o FSI, tais estudantes vão construir programas sintática, semântica e logicamente mais corretos.

Através da reflexão promovida pela auto-explicação é possível que os estudantes iniciantes em programação tenham um desempenho mais satisfatório. Desta forma, eles podem construir programas com menos erros sintáticos, aqueles que não permitem a compilação; erros semânticos, aqueles que permitem a compilação, mas causam término anormal durante a execução; e erros lógicos, aqueles que permitem a compilação e a execução, mas geram resultados incorretos [MCCAULEY et al., 2008 p . 68]. Em consequência, espera-se também que eles apresentem melhores notas em disciplinas de programação.

1.3 Objetivos

Sendo assim, o objetivo principal desta tese é propor a abordagem *Stepwise self-explanation* para o aprendizado de linguagens de programação. Esta abordagem se baseia na auto-explicação do processo de construção de programas de acordo com o FSI apresentado a partir de exemplos trabalhados em vídeo. Para alcançar este objetivo principal, também foram considerados os seguintes objetivos específicos:

- (i) identificar na literatura as dificuldades enfrentadas por estudantes e professores na área de ensino-aprendizagem de programação;
- (ii) definir a abordagem *Stepwise self-explanation* baseada nas dificuldades e práticas de ensino identificadas e como a abordagem será utilizada nos experimentos;
- (iii) definir um conjunto de materiais instrucionais associados para utilização da abordagem proposta; e
- (iv) avaliar a eficácia da abordagem *Stepwise self-explanation* para os processos de ensino e aprendizagem de programação. Os resultados obtidos com a avaliação devem guiar melhorias e ajustes na abordagem.

1.4 Metodologia

Este trabalho foi realizado em 5 etapas. A primeira etapa consistiu em revisão da literatura para identificar as características da área de ensino-aprendizagem de programação. Identificamos os principais problemas pelos quais os estudantes em programação passam e também algumas das decisões que os professores de programação precisam tomar. Verificamos as diferenças entre iniciantes e experientes em programação e, por fim, delineamos um panorama da literatura publicada no Brasil sobre os estudantes iniciantes em programação.

Logo após, iniciou-se a segunda etapa deste trabalho que consistiu no levantamento do referencial teórico, FSI, exemplos trabalhados, auto-explicação e aprendizagem cognitiva, que pudesse minimizar os problemas identificados na etapa anterior. Nesta etapa, também realizamos uma revisão da literatura focada em obter os principais trabalhos relacionados à abordagem proposta.

Na terceira etapa, desenvolvemos uma primeira versão da abordagem *Stepwise self-explanation* que atendia a hipótese definida na Seção 1.2. Desenvolvemos as questões para auto-explicação dos estudantes como parte da implementação da abordagem. Ademais, realizamos um experimento piloto para avaliar a eficácia desta primeira versão da abordagem.

Posteriormente, na quarta etapa, desenvolvemos uma segunda versão da abordagem *Stepwise self-explanation* que atendia a hipótese definida na Seção 1.2. Desenvolvemos o material instrucional que implementou a abordagem. Além disso, realizamos três quase-experimentos para avaliar a eficácia desta primeira versão da abordagem.

Por fim, na quinta e última etapa, avaliamos o material instrucional com os estudantes participantes dos quase-experimentos e com um conjunto de professores da área de programação e de educação.

1.5 Contribuições

As contribuições desta pesquisa estão relacionadas às abordagens para os processos de ensino e aprendizagem de estudantes iniciantes na disciplina de linguagem de programação, com a definição de conceitos que podem ser aplicados por

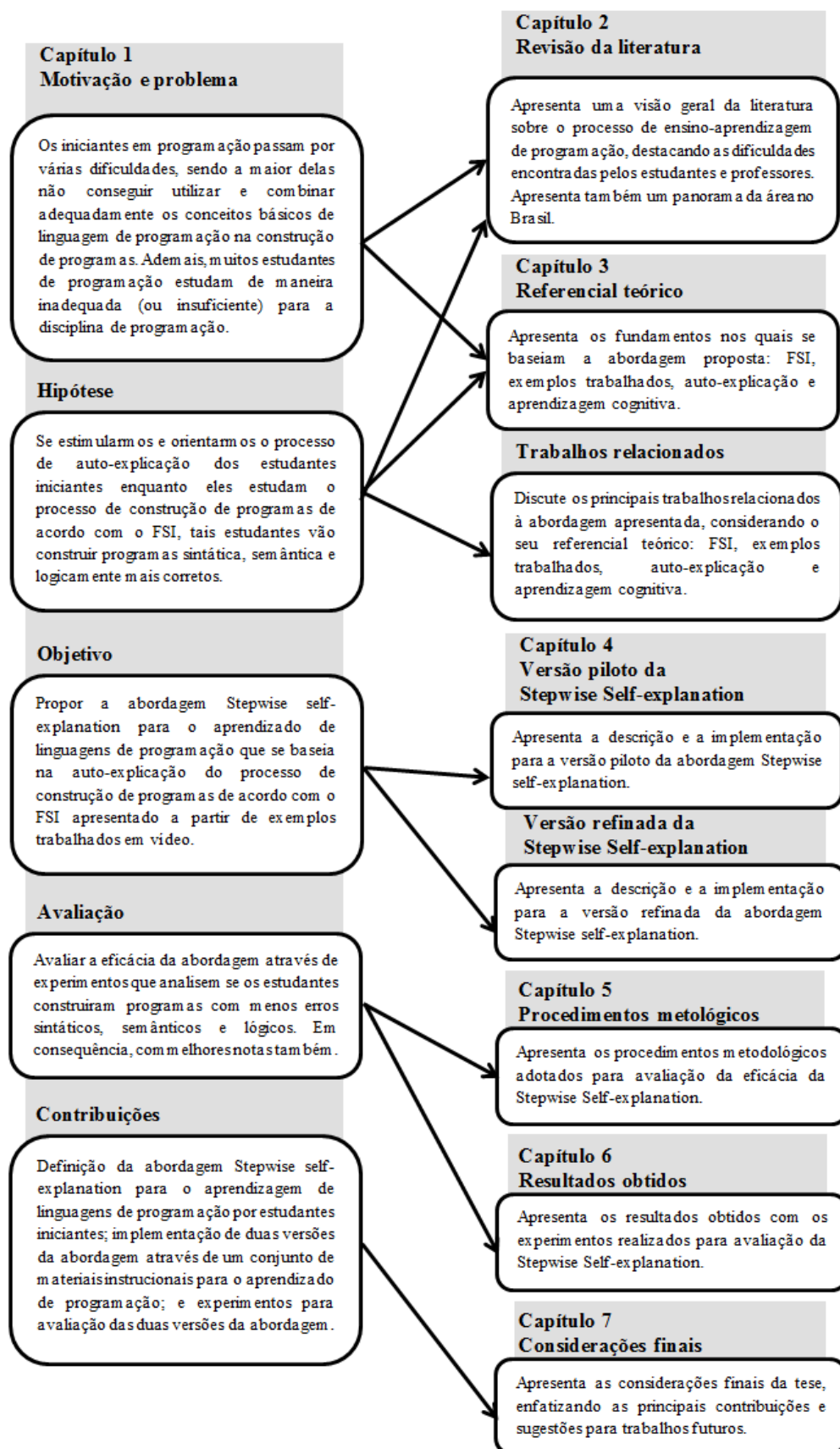
pesquisadores e professores da área de linguagem de programação. Sendo assim, as principais contribuições da presente tese incluem:

- (i) mapeamento sistemático da literatura sobre os processos de ensino e aprendizagem de programação para iniciantes no Brasil;
- (ii) definição da abordagem *Stepwise self-explanation* para o aprendizagem de linguagens de programação por estudantes iniciantes;
- (iii) implementação de duas versões da abordagem através de um conjunto de materiais instrucionais para o aprendizado de programação;
- (iv) experimento piloto para avaliação da versão piloto da abordagem;
- (v) experimentos para avaliação da versão refinada da abordagem; e
- (vi) avaliação do conjunto de materiais instrucionais produzidos para a implementação da abordagem com estudantes iniciantes em programação e professores da área de programação e de educação.

1.6 Estrutura da tese

Este trabalho está organizado em sete capítulos conforme apresentado na Figura 1.1. No Capítulo 2, apresentamos uma revisão da literatura sobre a área de ensino-aprendizagem de programação. Destacamos as dificuldades encontradas pelos estudantes e pelos professores. Além disso, apresentamos um panorama da área de ensino-aprendizagem de programação para iniciantes no Brasil. No Capítulo 3, apresentamos o referencial teórico utilizado nesta tese: o FSI, a teoria da carga cognitiva e os exemplos trabalhados, a teoria da auto-explicação e a aprendizagem cognitiva. Além disso, apresentamos os principais trabalhos relacionados à abordagem apresentada nesta tese. Em seguida, no Capítulo 4, apresentamos a definição da versão piloto e a versão refinada da abordagem *Stepwise Self-explanation*. No Capítulo 5, apresentamos os procedimentos metodológicos para avaliação de eficácia da abordagem. No Capítulo 6 apresentamos os resultados obtidos com os experimentos realizados. Finalmente, no Capítulo 7, descrevemos as contribuições e sugerimos alguns desdobramentos interessantes para esta tese.

Figura 1.1. Organização geral da tese.



Fonte: própria autora.

2 Os processos de ensino e de aprendizagem de programação

Este segundo capítulo tem por objetivo apresentar uma revisão da literatura sobre os processos de ensino e de aprendizagem de programação. Descrevemos na Seção 2.1 a literatura relacionada ao objetivo de pesquisa desta tese que relatam as dificuldades encontradas pelos estudantes em programação. Na Seção 2.2 destacamos as dificuldades que encontram e as decisões que precisam tomar os professores de disciplinas de programação. Na Seção 2.3 abordamos a diferença entre os iniciantes e os experientes em programação. Na Seção 2.4 apresentamos uma análise das pesquisas relacionadas aos processos de ensino e de aprendizagem de programação para iniciantes no Brasil por meio de um mapeamento sistemático da literatura. Finalmente, apresentamos na Seção 2.5 as considerações finais do capítulo.

2.1 Aprender a programar é difícil

A literatura envolvendo pesquisas relacionadas aos processos de ensino e de aprendizagem de programação é unânime ao afirmar que disciplinas introdutórias de programação são consideradas difíceis pela maioria dos estudantes [CASPERSEN, 2007; DU BOULAY, 1989; GOMES; MENDES, 2007; GOMES; SANTOS; MENDES, 2012; MILNE; ROWE, 2002; ROBINS et al., 2003]. Normalmente, estas disciplinas estão relacionadas a altas taxas de reprovação, que segundo dados apresentados por Bennedsen e Caspersen (2007) podem chegar a média de um terço dos estudantes matriculados nestas disciplinas no mundo todo. Mais recentemente, Watson e Li (2014) revisitaram os dados apresentados por Bennedsen e Caspersen (2007) e mostraram que as taxas de reprovação em disciplinas introdutórias de programação permaneceram constantes ao longo dos anos. Disciplinas de introdução à programação também possuem altas taxas de evasão nos cursos de tecnologia, que já chegaram, por exemplo, à metade dos estudantes matriculados em uma universidade na Finlândia [KINNUNEN; MALMI, 2006] ou a quase um terço dos estudantes matriculados nos cursos de Ciência da Computação [SILVA FILHO et al., 2007]. Programar é considerada uma tarefa tão complexa que estudos apontam que mesmo após serem aprovados em uma disciplina introdutória de programação, estudantes ainda apresentam sérios problemas ao aplicar os conceitos da disciplina [KURLAND et al., 1986; MCCRACKEN et al., 2001]. Outra pesquisa traz resultados que mostram que a maioria dos estudantes tem seus conhecimentos de programação consolidados apenas ao final da

segunda disciplina de programação [TEW; MCCRACKEN; GUZDIAL, 2005]. Por este motivo, ensinar programação é considerado um dos sete grandes desafios da área de educação em computação [CASPERSEN 2007].

Robins. Roundtree e Roundtree (2003) publicaram uma extensa revisão da literatura sobre o ensino e a aprendizagem de programação com foco principal nos estudantes iniciantes, que ainda permanece extremamente atual. Os trabalhos analisados por estes autores indicam que aprender a programar envolve adquirir um conjunto de conhecimentos novos e complexos, ao mesmo tempo em que se adquirem estratégias e habilidades práticas relacionadas. De acordo com du Boulay (1989), por exemplo, ao começar a programar, os estudantes vivenciam dificuldades em 5 diferentes áreas, mas que se sobrepõem. São elas: (i) a orientação geral do problema, que envolve descobrir qual a utilidade da programação, que tipos de problemas podem ser resolvidos com os programas e as vantagens de se aprender a programar; (ii) a máquina imaginária¹ que é um modelo da maneira como o computador funciona enquanto um programa está executando; (iii) a notação que está relacionada à sintaxe e à semântica de uma linguagem de programação; (iv) as estruturas que estão relacionadas aos esquemas e aos planos que devem ser aprendidos; e (v) a pragmática que está relacionada às habilidades de especificar, desenvolver, testar e debugar que os estudantes também precisam aprender ao começar a programar. Rogalski e Samurçay (1990) concordam com du Boulay ao afirmar que aprender a programar implica no aprendizado de um novo conjunto de atividades cognitivas e representações mentais relacionadas ao entendimento, planejamento, modificação e retirada de erros de programas.

Vários são os trabalhos que relatam as dificuldades de entendimento que os estudantes iniciantes em programação possuem sobre os assuntos da disciplina. O livro *Studying the novice programmer* editado por Soloway e Spohrer (1989) apresenta um conjunto de artigos que falam sobre as dificuldades que os programadores iniciantes possuem e os erros que eles cometem ao aplicar alguns dos construtos das linguagens de programação, tais como variáveis, estruturas condicionais e de repetição. Neste livro, Samurçay (1989) trata das dificuldades vivenciadas pelos estudantes iniciantes, examinando como eles aplicam o conceito de variáveis. Os resultados demonstram que a inicialização de variáveis é uma atividade mais difícil do que a atualização e a

¹ Do termo em inglês *notional machine*.

utilização de variáveis em condições de parada de laços de repetição. A inicialização de variáveis é melhor entendida quando é realizada a partir de um comando de leitura de dados do que a partir de uma atribuição de valor. Além disso, a atualização e a utilização de variáveis em condições de parada de laços de repetição parecem ter um nível similar de dificuldade. du Boulay (1989) relata uma série de obstáculos enfrentados por estudantes iniciantes ao estudar programação em Pascal. De maneira similar aos achados de Samurçay, du Boulay encontra que os estudantes têm dificuldades em entender as operações de inicialização de variáveis e de atribuição de valores às mesmas. Ele também relata que os estudantes encontram dificuldades ao aprender sobre *arrays*, confundindo principalmente os valores armazenados no *array* com os seus índices. Outra dificuldade descrita está relacionada à mudança do fluxo de controle do programa ocasionada pela utilização de estruturas condicionais e de repetição no código. Por último, ele também comenta a dificuldade de utilização do ponto e vírgula pelos estudantes, principalmente quando os erros não são indicados pelo compilador, mas acabam por criar estruturas condicionais e de repetição cujos corpos não são executados. Spohrer et al. (1989) analisaram os erros encontrados nos primeiros programas sintaticamente corretos submetidos pelos estudantes para um determinado problema. Os resultados mostraram que os erros mais frequentes nas implementações construídas pelos estudantes foram aqueles relacionados aos testes nas estruturas condicionais e de repetição, resultado que se assemelha ao encontrado por du Boulay. Em seguida a estes erros, os mais comuns foram os erros relacionados aos comandos de entrada e saída, à delimitação de escopo de blocos de código, à atualização e à inicialização de variáveis. Putnam et al. (1989) apresentaram os erros cometidos por estudantes iniciantes na linguagem de programação Basic. Os seus resultados mostraram que os estudantes com um semestre ou mais de experiência em Basic ainda cometiam erros básicos relacionados aos diferentes construtos aprendidos, tais como em comandos de atribuição de valores a variáveis, de entrada e saída, e relacionados às estruturas condicionais e de repetição.

Milne e Rowe (2002) conduziram um questionário com o objetivo de coletar os conceitos e tópicos relacionados ao paradigma de orientação a objetos (utilizando a linguagem de programação C++) que são considerados mais difíceis pelos iniciantes em programação. Os resultados mostraram que os estudantes possuem diversos problemas ao aprender conceitos mais abstratos, especialmente os relacionados à utilização de

ponteiros e à manipulação de memória. Lahtinen, Ala-Mutka e Järvinen (2005) também realizaram um questionário com o objetivo de coletar opiniões sobre as dificuldades percebidas pelos estudantes e pelos professores ao aprender e ensinar programação, respectivamente. Os resultados mostraram que as maiores dificuldades são relacionadas principalmente ao aprendizado de conceitos que necessitam de um entendimento mais geral do programa. As questões consideradas mais difíceis pelos estudantes que responderam ao questionário foram entender como projetar um programa para resolver uma determinada tarefa, dividir o programa em procedimentos e encontrar erros nos seus programas. Além disso, os conceitos considerados mais difíceis pelos estudantes foram recursão, ponteiros e referências, tipos abstratos de dados, manipulação de erros e utilizar bibliotecas da linguagem, alguns deles já citados anteriormente por Milne e Rowe (2002).

Gomes e Mendes (2007) afirmam que muitos estudantes estudam de maneira inadequada para uma disciplina de programação. Eles empregam metodologias incorretas de estudo, o que inclui, por exemplo, decorar a solução de problemas ao invés de entender os conceitos relacionados à construção de programas. Além disso, os iniciantes em programação frequentemente não estudam o suficiente às necessidades da disciplina de programação, através da resolução de listas de exercícios. Eles também afirmam que os alunos não possuem o nível de abstração necessário para resolução de problemas genéricos nem possuem raciocínio lógico-matemático suficiente e que, por isso, eles não conseguem construir algoritmos. Mais recentemente, Cherenkova, Zingaro e Petersen (2014) analisaram mais de 260 mil programas construídos por estudantes matriculados em uma disciplina introdutória de programação ao longo de um semestre. Os resultados desta análise indicaram que os estudantes possuem sérios problemas na aplicação das estruturas condicionais e de repetição, sendo estas últimas as que representam o maior desafio para eles. Além disso, segundo os autores, embora os estudantes tenham melhorado no decorrer do semestre, as dificuldades relacionadas a estes assuntos persistiram durante todo o período.

Contudo, Robins, Rountree e Rountree. (2003) sugerem que as grandes dificuldades dos iniciantes em programação estão relacionadas a questões de resolução de problemas, ao projeto e a codificar uma solução como um programa. Nesse sentido, o maior dos problemas vivenciados pelos iniciantes não parece ser o entendimento dos conceitos básicos de programação, mas a combinação e a utilização adequada destes

conceitos na construção de um determinado programa. Dessa forma, os iniciantes possuem o que se chama o conhecimento em programação, ou seja, o conhecimento de programação de natureza declarativa [DAVIES, 1993], mas não conseguem aplicar esse conhecimento através das estratégias de programação, ou seja, a maneira como o conhecimento de programação é usado e aplicado [DAVIES, 1993]. É a esta mesma conclusão à qual chegam Spohrer e Soloway (1989) após a análise dos dados coletados em um curso introdutório de programação na linguagem Pascal: que a maioria dos erros cometidos pelos estudantes não resulta de falhas no conhecimento sobre os construtos da linguagem de programação, mas resultam de dificuldades em combiná-los corretamente, ou seja, “*em colocar as peças do programa juntas*” [SPOHRER; SOLOWAY, 1989 p. 401]. Opinião que é compartilhada por diversos outros autores presentes na literatura [LAHTINEN et al., 2005; ROBINS et al., 2003; WINSLOW, 1996].

Robins, Rountree e Rountree (2003) afirmam ainda que os problemas enfrentados pelos estudantes parecem ser agravados quando o conhecimento adquirido por eles é frágil ou não consolidado. Esse conceito de conhecimento frágil foi introduzido por Perkins e Martin (1986) quando eles conduziram um estudo com alunos do ensino médio matriculados em um curso introdutório de programação na linguagem BASIC. Os autores classificaram como conhecimento frágil aquele conhecimento que é parcial, difícil de usar ou é utilizado incorretamente, em razão de sua não consolidação no momento em que é utilizado. Assim, o conhecimento frágil pode ser classificado como (i) ausente, para aquele conhecimento que foi esquecido; (ii) inerte, para aquele conhecimento que foi aprendido mas não é utilizado; ou (iii) incorreto, para aquele conhecimento que foi aprendido, mas é utilizado incorretamente [ROBINS et al., 2003]. Além disso, Robins, Rountree e Roundtree afirmam que o conhecimento procedural também pode ser frágil. Quando isto acontece, os estudantes iniciantes não conseguem analisar o código escrito para a retirada de erros, mesmo quando eles têm consciência do fluxo de execução do programa.

Outro tema bastante pesquisado sobre os estudantes iniciantes de programação envolve os fatores que influenciam o sucesso em programação, ou seja, as habilidades que predizem o bom desempenho destes estudantes em disciplinas de programação. Neste âmbito, pesquisas mostram que abordagens de ensino que fazem com que os estudantes tenham um aprendizado mais profundo apresentam uma correlação positiva

com as notas obtidas por estes estudantes em disciplinas introdutórias de programação [DE RAADT et al., 2005; SIMON et al., 2006]. Outras pesquisas mostram que possuir experiência prévia em programação também é considerado um fator que influencia o sucesso de estudantes iniciantes em programação [BERGIN; REILLY, 2005; WIEDENBECK, 2005].

Em uma revisão da literatura acerca dos processos de ensino e de aprendizagem de programação em um período de quatro anos (2005-2008) em seis importantes conferências na área, Sheard e co-autores (2009) mencionam como fatores de sucesso algumas características, tais como as notas na disciplina de Matemática e a experiência prévia em disciplinas relacionadas a Ciências; a auto-percepção do estudante sobre o seu entendimento; estratégias de aprendizagem profundas e metacognição, entre outras [SHEARD et al., 2009 p. 99]. Mais recentemente o trabalho apresentado por Watson, Li e Godwin (2014) introduziu fatores dinâmicos para previsão do sucesso de estudantes em disciplinas introdutórias de programação. Segundo os autores, estes fatores baseados na frequência de eventos (sequências de erros e sucessos dos estudantes enquanto estão programando) e no tempo que os estudantes passaram programando em aulas de laboratório estão moderada ou fortemente correlacionados com o desempenho dos estudantes na disciplina.

2.2 Ensinar a programar também é difícil

Ser professor de uma disciplina introdutória de programação é uma tarefa tão complexa quanto ser estudante em uma disciplina como essa. Embora existam muitos trabalhos que relatam as diferentes dificuldades vivenciadas pelos estudantes ao iniciarem suas vidas como programadores, os anos de pesquisa intensa na área trouxeram efeitos limitados para a atuação da prática do professor em sala de aula [PEARS et al., 2007]. Com o objetivo de melhorar esse cenário, Pears et al. (2007) identificaram e classificaram trabalhos importantes da literatura de programação introdutória que podem ajudar o trabalho dos professores ao planejar esta disciplina. Sendo assim, a partir da referida pesquisa, os autores esperam auxiliar os professores de programação a tomar decisões tais como a escolha da linguagem de programação a ser adotada, das ferramentas e ambientes de programação, além de questões como os tipos de práticas pedagógicas que obtiveram resultados positivos ao serem aplicadas e como

essa disciplina pode ser incorporada ao currículo de um curso da qual ela precisa fazer parte.

Segundo Pears et al. (2007), uma das primeiras decisões que um professor planejando uma disciplina introdutória de programação precisa tomar diz respeito à ementa daquela disciplina, ou seja, o que vai ser ensinado durante o período em que a disciplina será lecionada. De acordo com eles, o planejamento de uma disciplina de programação introdutória envolve um conjunto de definições sob as quais o professor deve ter ciência das opções que possui. As opções são as seguintes, entre outras: (i) do paradigma de programação que serão ensinados, (ii) do tema norteador da disciplina e (iii) do público alvo ao qual ela será direcionada. A definição do paradigma de programação possui como algumas opções o paradigma de orientação a objeto, o procedural e o funcional, entre outros. Em relação à definição do tema norteador da disciplina, Pears et al. (2007) citam entre outras opções, a programação orientada a componentes. Por fim, a definição do público alvo vai depender do tipo de curso no qual a disciplina está inserida. Algumas opções de público alvo, por exemplo, podem ser alunos do ensino fundamental ou médio cursando uma disciplina optativa, alunos do ensino médio matriculados em um curso técnico e alunos matriculados em diferentes cursos em uma universidade que possuem uma disciplina introdutória de programação como disciplina obrigatória, entre outros.

Para auxiliar a equipe de professores na decisão do que deve ser ensinado aos estudantes em uma disciplina introdutória de programação, a ACM e a IEEE publicam recomendações para o currículo de diversos cursos da área de computação. O mais recente currículo da ACM/IEEE para a área de Ciência da Computação foi publicado no ano de 2013 [J. T. F. ON COMPUTING CURRICULA, 2013]. Apesar de não indicar um paradigma de programação ou um tema norteador para a disciplina introdutória de programação em específico, o currículo sugere os tópicos que devem ser abordados, bem como os objetivos de aprendizagem que devem ser alcançados pelos estudantes ao cursar a disciplina.

A segunda decisão que o professor planejando uma disciplina tem que tomar está relacionada às práticas pedagógicas adotadas na disciplina. Segundo Pears et al. (2007), enquanto a ementa define o que deve ser ensinado, as práticas pedagógicas ensinam a maneira com que o processo de ensino e aprendizagem é conduzido de maneira a facilitar os resultados de aprendizado desejados. Dentre as técnicas que se destacam

estão a resolução de problemas, aprendizado de uma linguagem em particular e a produção de código e sistemas durante o curso.

A habilidade de resolução de problemas é uma das características fundamentais da programação. Assim, vários professores acreditam que essa abordagem deve ser privilegiada durante o ensino de programação. Alguns autores são favoráveis à adoção do ensino de programação através da resolução de problemas. Barnes et al. (1997) propuseram uma abordagem para o ensino de programação baseada na abordagem de resolução de problemas para a educação matemática de Pólya (1945). Os autores desta abordagem para o ensino de programação nomearam-na *How to program it*, em alusão ao livro *How to solve it* onde a abordagem de Pólya é apresentada. Contudo, Winslow (1996) alerta para o fato de que ensinar resolução de problemas é necessário, mas não suficiente para o ensino de programação. Além disso, é importante ensinar também como expressar as resoluções dos problemas como programas. Assim, ensinar as estruturas de uma linguagem e como combiná-las e usá-las apropriadamente deve ser uma preocupação de todo professor de programação [ROBINS et al., 2003].

A segunda prática pedagógica a qual Pears et al. (2007) se referem é a mais tradicional delas e está relacionada ao ensino principalmente das estruturas e da sintaxe de uma linguagem de programação em particular ou de uma família de linguagens, tais como as linguagens orientadas a objeto. Segundo Robins, Rountree e Rountree (2003), este é o caso da maioria dos livros-textos em linguagem de programação, que estão preocupados apenas com a apresentação do conhecimento sobre uma linguagem de programação em particular. Estes livros negligenciam as habilidades estratégicas necessárias para o ensino de uma linguagem de programação, assim como o processo de criação dos exemplos apresentados. Em resumo, os livros focam principalmente na apresentação de construtos, de conceitos de programação e de exemplos de programas estáticos, sem endereçar a questão da trajetória percorrida pelo programador para se chegar a uma determinada solução [CASPERSEN; KÖLLING, 2009]. O foco somente em aspectos da sintaxe da linguagem de programação traz prejuízos aos estudantes, uma vez que as suas habilidades de programação acabam não sendo desenvolvidas [GOMES; MENDES, 2007].

Por fim, a última prática pedagógica para o ensino de programação à qual Pears et al. (2007) se referem é aquela que enfatiza o desenvolvimento de código e sistemas durante o curso. Neste âmbito, é citada a pesquisa de Soloway (1986) que enfatiza o

ensino de programação através de esquemas e planos. Segundo este autor, o ensino de programação a partir somente da sintaxe e da semântica de uma linguagem de programação não é suficiente para que os estudantes aprendam a programar. Ao invés disso, os estudantes precisam aprender sobre o conhecimento que os mais experientes em programação possuem, isto é, sobre mecanismos, explicações, metas, planos e composição de planos, entre outros assuntos [SOLOWAY, 1986].

Aliado ao conhecimento de esquemas e planos para as estruturas e as estratégias de utilização de uma linguagem de programação, uma recomendação importante feita por Robins, Rountree e Rountree (2003) é que a prática pedagógica dos professores deve privilegiar principalmente a combinação e o uso adequado das estruturas para a construção dos programas. Esta afirmação é reforçada por Spohrer e Soloway (1989) que afirmam que é necessário utilizar estratégias de ensino que ressaltem a integração de metas e planos durante a construção de programas, e, além disso, enfatize a combinação dos construtos da linguagem de programação para alcançar determinados objetivos.

O terceiro conjunto de pesquisas analisadas por Pears et al. (2007) fornece um conjunto de informações para a escolha da linguagem de programação que será adotada em uma disciplina introdutória de programação. Os trabalhos citados nessa categoria trazem uma série de conclusões, tais como: as linguagens Java, C++ e C continuam entre as mais comumente utilizadas para o ensino; muitos professores utilizam uma linguagem orientada a objetos para ensinar o paradigma procedimental; poucos são os professores que ensinam programação funcional; há uma insatisfação com a utilização de Java, C++ e C para o ensino e um debate que sugerem a sua substituição por outras linguagens (e.g., Python, Logo, Eiffel, Pascal) que foram desenvolvidas objetivando-se o ensino; uma disciplina introdutória de programação deve ter como objetivo principal a resolução de problemas e o desenvolvimento de algoritmos; a escolha da linguagem de programação pode ser feita pelo demanda de mercado e dos alunos. Atualmente, as linguagens Java, C e C++ continuam sendo as linguagens mais populares de acordo com o índice da comunidade de programação TIOBE².

Em seu trabalho, Sheard et al. (2009) também encontraram pesquisas que foram categorizadas como experiência de ensino dos professores e o que eles fazem para

² TIOBE software. Disponível em: < <http://bit.ly/19fhRoL> >. Acesso em: 04 de Jan. 2016.

auxiliar seus alunos no aprendizado da disciplina. Alguns resultados relatados pelos autores foram: nem a linguagem de programação nem o paradigma de programação parecem afetar a habilidade de projetar programas dos alunos; que os alunos preferem construir jogos a criar histórias ou tarefas tradicionais, e que preferem também tarefas mais específicas às aquelas abertas; ensinar estratégias de programação de maneira explícita tem potencial para melhorar os resultados dos alunos; programação em par é uma técnica efetiva para o aprendizado do projeto de programa e da atividade de programação em si; desenvolvimento dirigido a testes pode melhorar o teste de código realizado pelo aluno, bem como o desempenho de programação; muitos estudos defendem a utilização de linguagens de programação visuais, no entanto um estudo relatou a dificuldade que os alunos possuem ao transferir o conhecimento adquirido para a linguagem textual e outro estudo relatou que os estudantes estão focados mais em dicas textuais na tela e menos na animação. Neste âmbito, Aureliano e Tedesco (2012a) compararam a utilização do Scratch com a utilização da linguagem de programação C para o ensino de *arrays* e não encontraram uma diferença estatisticamente significativa nos resultados obtidos pelos estudantes, o que é consistente com a literatura apresentada [PEARS et al., 2007].

Por fim, Sheard et al. (2007) analisaram a pesquisa desenvolvida na área de ferramentas que podem ser utilizadas para auxiliar professores e estudantes nos processos de ensino e de aprendizagem de linguagem de programação. Segundo estes autores, há um grande número de pesquisas relacionadas ao desenvolvimento de ferramentas com este objetivo. Uma situação semelhante ocorre no Brasil [AURELIANO; TEDESCO, 2012b]. No entanto, a maioria destas ferramentas acabam sendo propostas para resolver problemas locais das instituições nas quais foram desenvolvidas ou como projetos de pesquisa que acabam sendo descontinuados e, por isso, dificilmente estas ferramentas acabam sendo utilizadas por outros pesquisadores, além dos que as desenvolveram. Embora um grande número de ferramentas desenvolvidas não seja utilizado amplamente, há alguns exemplos que são mais populares entre os professores de programação. Para o ensino de estudantes iniciantes em programação, algumas das ferramentas mais conhecidas e adotadas são Alice, Greenfoot, Scratch e BlueJ.

O Alice é um ambiente 3D para programação gráfica que possibilita a criação de mundos animados e interativos [COOPER; DANN; PAUSCH, 2000]. Com esta

ferramenta, programadores iniciantes podem construir mundos e escrever programas para animar os objetos desse mundo à medida que aprendem conceitos introdutórios de programação orientada a objetos. Ao invés de escrever o código linha a linha, para criar um programa usando Alice, basta que os estudantes arrastem as instruções e expressões para a janela de código. Desta maneira, os erros de sintaxe são inexistentes e os estudantes se preocupam apenas com o programa que estão criando. Originalmente, Alice foi desenvolvida para melhorar o desempenho de estudantes em disciplinas introdutórias de programação em cursos de Ciência da Computação [MOSKAL; LURIE; COOPER, 2004]. No entanto, ela também vem sendo utilizada no ensino introdutório de programação de estudantes do ensino médio [ADAMS, 2007; WANG et al., 2009].

O Greenfoot [HENRIKSEN; KÖLLING, 2004] fornece uma plataforma que permite uma abordagem interativa, visual e envolvente para ensinar programação orientada a objetos em Java para iniciantes. Assim como Alice, Greenfoot permite que a programação seja realizada através da criação e simulação de mundos e objetos desse mundo. A grande diferença, no entanto, é que os gráficos animados no Greenfoot são criados em 2D, enquanto Alice é um ambiente para gráficos 3D. Segundo Kölling (2008), Greenfoot pode ser utilizada em diferentes níveis de educação, por exemplo, com adolescentes do ensino médio visando motivá-los a seguirem uma carreira na área de computação ou com estudantes matriculados em disciplinas de programação em Java.

O BlueJ, a partir do qual o Greenfoot foi criado, também é um ambiente de desenvolvimento para a linguagem de programação Java criado especialmente para disciplinas introdutórias de programação nesta linguagem [KÖLLING et al., 2003]. Este ambiente de desenvolvimento foi pensado para simplificar o ensino de conceitos de orientação a objeto em uma abordagem de ensino chamada pelos autores de *Object first* (Objetos primeiro). Nesta abordagem, os estudantes criam as classes, os objetos e os métodos desde o seu primeiro contato com a linguagem e podem interagir com as classes criadas a partir de um diagrama UML simplificado que fica disponível na interface da ferramenta.

Por fim, o Scratch foi criado pelo grupo de pesquisa *Lifelong Kindergarten* do MIT *Media Lab* com o propósito de introduzir a programação de maneira fácil e rápida para aqueles que não possuem nenhum tipo de experiência no assunto [MALONEY et

al., 2010]. Para tanto, o processo de aprendizagem, com a utilização do Scratch, ocorre apoiado por uma linguagem de programação visual que permite a manipulação de mídias, tais como imagens e sons, para a criação de jogos ou de animações interativas, enfatizando o aprendizado auto-dirigido e a colaboração em pares [MALONEY et al., 2010; MALONEY et al., 2008].

2.3 O longo processo para se passar de iniciante a experiente

Diversas pesquisas focam na diferença entre pessoas iniciantes e experientes em diferentes assuntos. De acordo com Bransford, Brown e Cocking (2000), estas pesquisas demonstram que o extenso conhecimento adquirido pelos mais experientes influencia a forma como eles percebem e organizam, interpretam e representam a informação que a eles é disponibilizada. Consequentemente, eles conseguem lembrar, raciocinar e resolver problemas mais rápida e precisamente do que os menos experientes. Algumas das características importantes do conhecimento dos mais experientes que influenciam os processos de ensino e aprendizagem são as seguintes [BRANSFORD; BROWN; COCKING, 2000 p. 31]:

1. Experientes notam características e padrões significativos da informação que não são notados pelos estudantes;
2. Experientes adquiriram uma grande quantidade de conhecimento que é organizada de maneira que reflete um entendimento profundo do assunto;
3. O conhecimento dos experientes não pode ser reduzido a um conjunto de proposições ou fatos isolados, mas reflete situações de aplicabilidade, ou seja, o conhecimento é condicionado a um conjunto de circunstâncias;
4. Experientes recuperam de maneira ágil aspectos importantes do seu conhecimento sem muito esforço ou atenção;
5. Embora experientes saibam suas disciplinas completamente, isso não garante que eles sejam capazes de ensinar outras pessoas; e
6. Experientes possuem diferentes níveis de flexibilidade na maneira como lidam com novas situações.

Mais especificamente na área de programação, os pesquisadores são unânimes em afirmar que os iniciantes em programação não possuem o conhecimento específico e as habilidades de resolução de problemas que uma pessoa mais experiente possui. Winslow (1996) cita um conjunto de estudos concluindo que os iniciantes em

programação: não possuem um modelo mental da área detalhado; possui conhecimento superficial sobre o assunto; possuem o conhecimento frágil e não dão a devida importância às estratégias; utilizam estratégias gerais de resolução de problemas ao invés de específicas a um problema em questão; e utilizam uma abordagem mais “linha-a-linha” para a solução do problema [WINSLOW, 1996 p. 18]. Semelhantemente, Robins, Rountree e Rountree (2003) também citam outro conjunto de estudos que complementa os problemas que os iniciantes em programação possuem: quando comparado aos experientes, os iniciantes passam pouco tempo planejando e testando código; quando encontram um problema, os iniciantes corrigem o programa localmente ao invés de reformulá-lo por completo; têm pouca habilidade no rastreamento de código; não compreendem a natureza sequencial da execução de um programa; possuem conhecimento específico ao contexto ao invés de um conhecimento mais geral [ROBINS et al., 2003 p. 151].

De maneira semelhante, Winslow (1996) e Robins, Rountree e Rountree (2003) apresentam várias características dos experientes em programação. Winslow (1996) cita um conjunto de estudos que concluem que os experientes em programação: possuem vários modelos mentais, escolhem e combinam estes modelos de uma maneira oportunista; têm um conhecimento profundo sobre o assunto, que é organizado de maneira hierárquica e em camadas com mapas explícitos entre essas camadas; aplicam todo conhecimento que possuem; quando dada uma tarefa em uma área familiar, resolvem essa tarefa a partir dos dados fornecidos e desenvolvem sub-metas em uma maneira hierárquica; quando o problema não é familiar, retrocedem em técnicas gerais da resolução do problema; reconhecem problemas que possuem uma solução similar com maior facilidade (pelo uso de padrões ou esquemas); tendem a construir um programa a partir de suas estruturas de dados ou objetos; usam algoritmos ao invés de uma sintaxe específica, isto é, abstraem de uma linguagem particular para um conceito mais geral; possuem um conhecimento sintático e semântico melhor, além de possuir habilidades táticas e estratégicas melhores também [WINSLOW, 1996 p. 18]. Robins, Rountree e Rountree (2003), por sua vez, resumem os trabalhos sobre os experientes em programação afirmando que eles: possuem representações sofisticadas do conhecimento e das estratégias de resolução de problemas que empregam; possuem esquemas de conhecimento eficientemente organizados e especializados; organizam o conhecimento de acordo com características funcionais, tais como os algoritmos ao invés de focar em

detalhes superficiais como a sintaxe da linguagem; utilizam estratégias gerais de resolução de problemas assim como estratégias especializadas; são bons em reconhecer, usar e adaptar padrões e esquemas; são mais rápidos, mais precisos e são capazes de utilizar um grande conjunto de exemplos, fontes de conhecimento e estratégias de maneira efetiva [ROBINS et al., 2003 p. 139-140].

Em sua comparação sobre os iniciantes e os experientes em programação, Winslow (1996) destaca também algumas características que são independentes do nível de habilidade do programador. São elas: dada uma nova linguagem não familiar, sua sintaxe não é um problema, mas sim como usar e combinar as instruções para alcançar os efeitos desejados; aprender os conceitos e as técnicas de uma nova linguagem exige escrever programas nessa linguagem, estudar sintaxe e semântica não é suficiente para entender e aplicar apropriadamente a nova linguagem; resolver problemas por analogia é comum em todos os níveis, no entanto escolher a analogia adequada pode ser difícil; em todos os níveis, as pessoas avançam para o próximo nível praticando [WINSLOW, 1996 p. 18].

Leva-se cerca de 10 anos para que um programador iniciante se torne um programador experiente [WINSLOW, 1996]. Desta maneira, não é de se surpreender que os estudantes finalizem disciplinas de programação com sérias dificuldades para construir programas [MCCRACKEN et al., 2001] ou tenham seus conhecimentos básicos de programação consolidados apenas após a conclusão da segunda disciplina de programação [TEW et al., 2005]. No entanto, sabemos que não há outra maneira de percorrer esse caminho de iniciante a experiente a não ser a partir da prática e do estímulo ao aprendizado profundo dos estudantes. Veremos mais adiante como isso foi implementado neste trabalho.

2.4 Literatura relacionada ao ensino e à aprendizagem de programação para iniciantes no Brasil

Com o objetivo de analisar as publicações científicas realizadas no Brasil relacionadas ao ensino e à aprendizagem de programação para iniciantes, Aureliano e Tedesco (2012b) realizaram um mapeamento sistemático da literatura nos anos de 2002-2011 nos anais do Simpósio Brasileiro Informática na Educação (SBIE) e Workshop de Informática na Escola (WIE). Este trabalho foi estendido com os seguintes objetivos:

- incluir os artigos publicados nos anos de 2012-2014 do SBIE e WIE;

- incluir os artigos publicados nos anos de 2002-2014 no Workshop sobre educação em computação (WEI) e na Revista Brasileira de Informática na Educação (RBIE); e
- responder às novas questões de pesquisa apresentadas na Seção 2.4.1.

A seguir são apresentadas as questões de pesquisa que foram respondidas pelo mapeamento, os seus critérios de inclusão e de exclusão, a extração e a síntese dos dados e os seus resultados obtidos para a versão estendida deste mapeamento.

2.4.1 Questões de pesquisa do mapeamento

O mapeamento sistemático realizado teve como questão central de pesquisa a seguinte pergunta: *Qual o panorama atual das publicações científicas no Brasil relacionado ao ensino e à aprendizagem de programação para iniciantes?* Para responder a essa questão, foram definidas as seguintes questões específicas de pesquisa:

- QP1: Quais são os tópicos e os escopos dos artigos publicados no Brasil na área de ensino-aprendizagem de programação para iniciantes?
- QP2: Quais são os níveis de escolaridade para os quais as pesquisas na área de ensino-aprendizagem de programação para iniciantes no Brasil estão sendo destinadas?
- QP3: Quais são os tipos de pesquisa e como são realizadas as análises dos dados nas pesquisas realizadas na área de ensino-aprendizagem de programação para iniciantes no Brasil?
- QP4: Quais são as técnicas de coleta de dados utilizadas nas pesquisas realizadas na área de ensino-aprendizagem de programação para iniciantes no Brasil?
- QP5: Quais são os objetivos e resultados obtidos pelas pesquisas realizadas na área de ensino-aprendizagem de programação para iniciantes no Brasil?

2.4.2 Critérios de inclusão e exclusão

A pesquisa foi realizada através de busca manual nos anais dos eventos SBIE, WIE e WEI e na RBIE nos anos de 2002-2014. Esta busca foi executada em duas etapas. A pré-seleção dos artigos consistiu em verificar os sites ou CDs contendo os anais dos referidos eventos e acessar manualmente todos os artigos completos, lendo os seus títulos, resumos e palavras-chave. Depois da pré-seleção, em uma segunda etapa, todos os artigos pré-selecionados foram analisados pela primeira autora deste trabalho e a cada um deles foram aplicados os critérios de inclusão e exclusão que são

apresentados na Tabela 2.1. A aplicação de apenas um dos critérios de exclusão ou inclusão determinou se o artigo deveria ser excluído ou incluído, respectivamente. Posteriormente, os estudos selecionados de acordo com os critérios de inclusão tiveram seus dados extraídos para fornecer as respostas das perguntas de pesquisa. Os estudos semelhantes que apresentavam a evolução de uma mesma pesquisa e que continham os mesmos autores foram removidos para evitar duplicações. Em um último momento, os dados extraídos dos artigos selecionados para inclusão foram sintetizados para apresentação dos resultados.

Tabela 2.1. Critérios de exclusão e de inclusão.

Critérios de exclusão	Critérios de inclusão
• Artigos que abordam os processos de ensino e aprendizagem de programação como ferramenta para alcançar objetivos diversos; • Artigos que abordam os processos de ensino e aprendizagem de programação para estudantes que não são iniciantes; • Artigos que abordam os processos de ensino e aprendizagem de programação em disciplinas diversas àquelas disciplinas introdutórias em programação; • Artigos que não sejam claramente sobre os processos de ensino e aprendizagem de programação para iniciantes; • Artigos aos quais não tivemos acesso aos anais ou aos arquivos.	• Artigos completos que abordam os processos de ensino e aprendizagem de programação para iniciantes.

2.4.3 Extração e síntese dos dados

Nesta fase, foram extraídos os seguintes dados gerais: título, autores, evento, ano, instituições de pesquisa, unidades federativas destas instituições, nível de escolaridade dos sujeitos pesquisados, disciplina e curso onde a pesquisa foi realizada. Além disso, foram extraídos trechos dos artigos que responderam as perguntas de pesquisa desta revisão: tópicos, escopos e natureza dos artigos. Por fim, para os artigos considerados de pesquisa [SIMON, 2007], extraímos trechos que responderam as demais questões de pesquisa desta revisão: escopo dos artigos, nível de escolaridade investigado, tipo de pesquisa e de análise dos dados, técnica de coleta de dados e principais objetivos e resultados.

2.4.4 Resultados

Nos anos de 2002 a 2014 foram publicados um total de 2024 artigos completos nos eventos e na revista analisados, dos quais 89 foram selecionados para ser incluídos para a extração dos dados³. A Tabela 2.2 apresenta os resultados gerais dos artigos incluídos para os eventos SBIE, WIE e WEI e a RBIE.

Tabela 2.2. Resultado geral das buscas para o SBIE, WIE, WEI e RBIE.

Evento	Artigos incluídos	Total de artigos completos	Artigos incluídos / Total de artigos completos (%)
SBIE	42	1009	4.16%
WIE	9	572	1.57%
WEI	36	205	17.56%
RBIE	2	238	0.84%
Total	89	2024	4.40%

Durante o processo de busca, não conseguimos acesso aos anais do WIE do ano de 2004 nem aos anais do WEI dos anos de 2004 e 2006. Por esta razão, artigos completos que abordam o tema sobre os processos de ensino e aprendizagem de programação para iniciantes nestes eventos realizados nos referidos anos não foram considerados em nossa análise.

2.4.5 Distribuição temporal

O gráfico apresentado na Figura 2.1 mostra a distribuição temporal dos artigos incluídos nesta revisão. Apesar de não haver nenhum padrão ou tendência clara para a distribuição dos artigos ao longo dos anos, observamos que quase 70% deles foram publicados nos anos de 2008 a 2014. Para este período, tivemos um mínimo de 8 artigos completos publicados em 2011 e o máximo de 13 artigos completos publicados em 2014. Sendo assim, podemos dizer que, a partir de 2009, houve um aumento no interesse dos pesquisadores brasileiros sobre a questão dos processos de ensino e aprendizagem de programação para iniciantes e que esse interesse vem crescendo ao longo dos anos. Esse interesse se mostrou ainda mais evidente na última edição do Congresso Brasileiro de Informática na Educação (CBIE) em 2015, onde foi realizada a primeira edição do Workshop de Ensino em Pensamento Computacional, Algoritmos e Programação⁴ (WAlgProg). Ao contrário dos demais workshops do CBIE que tiveram apenas um turno para apresentações (manhã ou tarde), o WAlgProg teve dois turnos de apresentações devido à grande quantidade de artigos submetidos e selecionados.

³ A lista completa dos artigos incluídos está disponível em < <http://bit.ly/1JXNoE0>>.

⁴ WAlgProg. Disponível em: < <http://walgprog.gp.utfpr.edu.br/>>. Acesso em 15 jan. 2016.

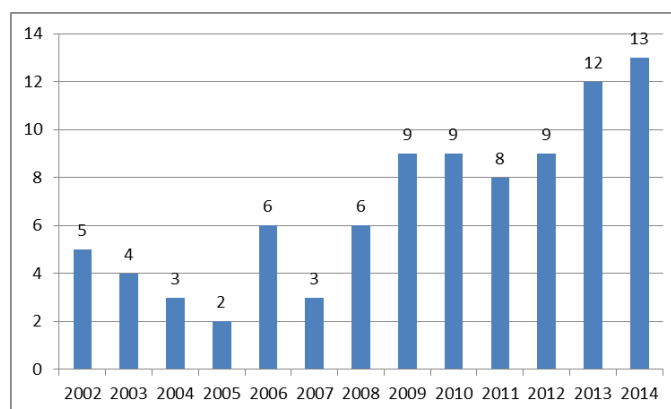


Figura 2.1. Distribuição temporal dos artigos.

2.4.6 Distribuição geográfica das instituições de pesquisa

Na Tabela 2.3 é apresentada a quantidade de artigos por instituições de pesquisa. As 4 instituições que mais publicaram ao longo destes 13 anos de pesquisa foram a Universidade do Vale do Itajaí (UNIVALI), a Universidade Federal do Espírito Santo (UFES), a Universidade Federal da Paraíba (UFPB) e a Universidade Federal de Campina Grande (UFCG) em conjunto com a Universidade Federal de Alagoas (UFAL), respectivamente com 9, 5, 3 e 3 artigos. Outras instituições publicaram 2 ou menos artigos completos, sendo que a grande maioria publicou apenas 1 artigo completo nos eventos e na revista estudados.

Tabela 2.3. Quantidade de artigos por instituições de pesquisa.

Instituição	Artigos	Instituição	Artigos	Instituição	Artigos
UNIVALI	9	UNISINOS	1	UFAL / UFCG	1
UFES	5	UNIRIO	1	UFABC	1
UFPB	3	UNIPLAC	1	UESC / UFES / UFRGS	1
UFCG / UFAL	3	UNIPAMPA / UFPEL	1	UERN / UFERSA	1
USP	2	Unifesp / Mackenzie / UFSCar	1	UEM / CTI Renato Archer	1
UFRJ	2	UnB	1	UEFS	1
UFRGS / Faculdade de Tecnologia Senac Pelotas	2	ULBRA	1	UEA	1
UFPE / IFPE	2	UFSC / Católica UNISANTOS	1	UCS / UFRGS	1
UFPA	2	UFRPE	1	UCB / UnB	1
UFG	2	UFRN	1	SERPRO	1
UFCG	2	UFRJ / UFLA	1	PUCRS	1
UFAL	2	UFRA	1	PUCPR	1
ITA	2	UFPR / UTFPR	1	PUC-Campinas	1
FATEC-ID / PUC-Campinas	2	UFPR	1	ITA / IMES / Mackenzie	1

CEETEPS	1	UFPA / UFRA / UFES	1	IMES / Mackenzie	1
USP / FACENS	1	UFF	1	IFAL / UFAL / UFCG	1
UNOESTE / UFRGS	1	UFCG / UFPB	1	FURB / Universiteit Van Amsterdam	1
UNIVALI / UNISUL	1	UFC	1	FAGOC / UNIFAL / UFRJ	1
UNIVALI / PUCRS	1	UFAM / UFES	1	FAESA	1
UNIVALE	1	UFAM	1	FACCAMP / CTI	1
UNISINOS / UNILASALLE	1	UFSCar / UENF	1	CEFET-ES / UFES	1

Na Figura 2.2 é apresentada a distribuição geográfica das instituições que realizaram as pesquisas apresentadas nos artigos. Das pesquisas apresentadas, 37% foram realizadas por instituições localizadas na regiões Sudeste, 29% na região Sul, 21% na região Nordeste, 7% na região Norte e 5% na região Centro-oeste. Aproximadamente 17% das pesquisas foram realizadas em colaboração de uma ou mais instituições de pesquisa brasileiras e um único artigo foi realizado em colaboração com uma instituição de pesquisa internacional.

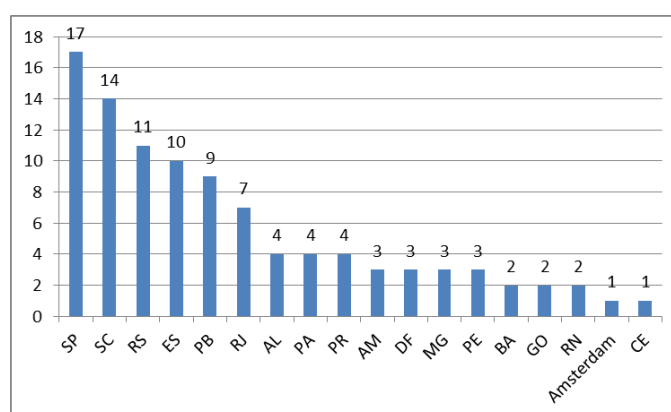


Figura 2.2. Distribuição geográfica das instituições de pesquisa.

2.4.7 Tópicos e escopo dos artigos

Para fornecer um quadro geral dos tópicos abordados nos artigos, lançamos mão da classificação proposta por Simon (2007) e utilizada anteriormente para classificar os artigos apresentados por Sheard et al. (2009). Aplicamos primeiro aos artigos a dimensão de tópicos [SIMON, 2007]. Nos dados apresentados, percebemos que a grande maioria dos artigos, cerca de 63%, compreende ferramentas que foram desenvolvidas para auxiliar o processo de ensino e aprendizagem de programação para iniciantes. O segundo tópico mais frequentemente abordado, cerca de 24% dos artigos, são as técnicas criadas ou aplicadas ao processo de ensino e aprendizagem de programação para iniciantes. Já o terceiro tópico mais frequentemente abordado, cerca

de 8% dos artigos selecionados, trata de ferramentas que foram desenvolvidas para auxiliar o processo de avaliação dos estudantes iniciantes em programação. Os outros tópicos correspondem aos 5% restantes. Sendo assim, notamos que há uma nítida preferência da comunidade brasileira pela proposição de novas ferramentas de software, o que está em desacordo com a revisão apresentada por Sheard et al. (2009), onde a proposição de ferramentas ocupa a terceira posição no ranking dos artigos publicados.

Tabela 2.4. Tópicos abordados nos artigos.

Tópico	Descrição do tópico (adaptado de [SIMON, 2007])	#Artigos
Ferramentas para os processos de ensino e aprendizagem	Ferramentas que foram desenvolvidas para auxiliar o processo de ensino e aprendizagem.	56
Técnicas de ensino-aprendizagem	Técnicas que foram criadas ou aplicadas ao processo de ensino e/ou aprendizagem (incluindo a aplicação de ferramentas existentes).	21
Ferramentas de avaliação	Ferramentas que foram criadas ou aplicadas ao processo de avaliação dos estudantes.	7
Habilidades/aptidão	O que faz os estudantes serem bons em programação; como eles podem se preparar melhor para a disciplina; quais os processos que eles aplicam ao estudar para a disciplina.	3
Currículo	Conteúdo da disciplina de programação.	1
Técnicas de avaliação	Técnicas que foram criadas ou aplicadas ao processo de avaliação dos estudantes	1

A Tabela 2.5 mostra a aplicação da dimensão natureza que diferencia artigos considerados mais práticos, tais como um relato de experiência, dos artigos considerados de pesquisa, tais como aqueles nos quais são conduzidos um experimento [SIMON, 2007]. Nesta tabela, podemos perceber que a grande maioria dos artigos, cerca de 65%, pertence à categoria dos artigos mais práticos, os relatórios. Os outros artigos classificados como experimentos ou análises são os considerados arquivos de pesquisa e respondem pelos 35% restantes dos artigos selecionados. Com este resultado, percebemos que a grande maioria dos artigos sobre a área de ensino e aprendizagem de programação para iniciantes que são publicados no Brasil não são considerados artigos de pesquisa.

Tabela 2.5. Natureza dos artigos.

Natureza	Descrição da categoria de natureza (adaptado de [SIMON, 2007])	#Artigos
Relatório	Relato de experiência sobre algo que foi realizado, talvez contendo um pesquisa de satisfação dos estudantes.	58
Experimento	Responde a uma pergunta de pesquisa através do projeto e realização de um estudo.	20
Análise	Responde a uma pergunta de pesquisa através da análise de dados (e.g., taxa de aprovação).	11

Aos 31 artigos considerados de pesquisa, aplicamos a categoria escopo que considera o alcance da pesquisa realizada [SIMON, 2007]. A Tabela 2.6 mostra que a grande maioria dos artigos, cerca de 71%, envolveu pesquisa apenas da disciplina de programação introdutória em apenas uma instituição. Apenas 10% dos artigos envolveram estudantes em mais de uma instituição. Por fim, 19% das pesquisas envolveram a disciplina introdutória de programação com outras disciplinas.

Tabela 2.6. Escopo dos artigos.

Escopo	Descrição da categoria de escopo (adaptado de [SIMON, 2007])	#Artigos
Disciplina	Quando a pesquisa envolveu apenas a disciplina de programação introdutória em uma única instituição.	22
Programa/ Departamento	Quando a pesquisa envolveu a disciplina de programação em pelo menos dois cursos de um mesmo programa ou departamento ou envolveu a disciplina de programação com outras disciplinas de um mesmo curso.	6
Várias instituições	Quando a pesquisa envolve estudantes de pelo menos duas instituições.	3

2.4.8 Nível de escolaridade

Os artigos foram classificados de acordo com o nível de escolaridade para os quais as pesquisas se destinam. Os níveis de escolaridade indicados são apresentados no gráfico da Figura 2.3. Verificamos que aproximadamente 81% dos artigos abordaram estudos no contexto da educação superior. Já 7% dos artigos apresentaram estudos no contexto do nível técnico e 10% no contexto do ensino médio. Os restantes 2% não informaram o nível de escolaridade no qual a pesquisa foi realizada. Desta forma, estes dados mostram claramente que as pesquisas no Brasil focam os processos de ensino e aprendizagem de programação para estudantes iniciantes matriculados no ensino superior.

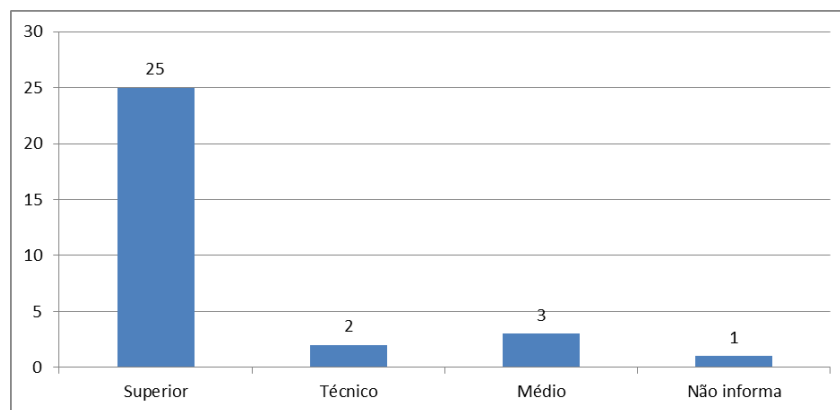


Figura 2.3. Níveis de escolaridade para os quais as pesquisas são destinadas.

2.4.9 Tipo de pesquisa e de análise de dados realizados

Os artigos também foram classificados de acordo com o tipo de pesquisa, quantitativa, qualitativa ou mista, que foi apresentada como parte do estudo [SIMON, 2007]. A grande maioria dos artigos, cerca de 71%, apresenta uma abordagem quantitativa de pesquisa. Apenas 7% dos artigos reportaram uma abordagem qualitativa, enquanto 22% reportaram uma abordagem mista. Nos artigos com a abordagem quantitativa, há predominância da apresentação dos dados através de estatísticas descritivas. Dentre estes, vemos que aproximadamente um terço deles também adota um teste estatístico, tais como Mann-Whitney para análise da diferença entre grupos e Spearman para verificação do grau de relacionamento entre variáveis. Os artigos que adotaram uma abordagem qualitativa, não informaram explicitamente o tipo de análise adotado. Os artigos que adotaram uma abordagem mista, em sua maioria, utilizaram uma abordagem qualitativa para formar categorias associada a frequência com que a categoria definida ocorreu.

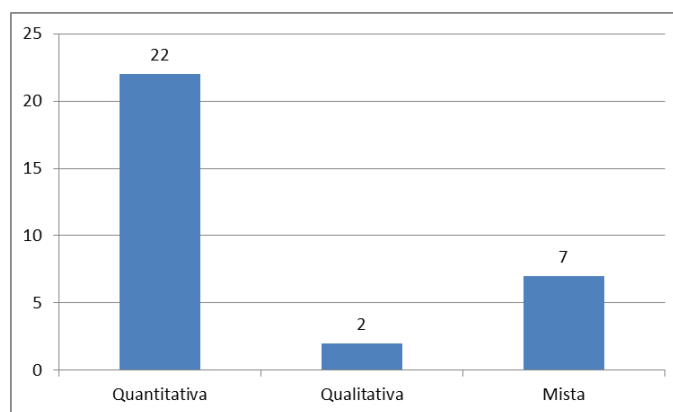


Figura 2.4. Tipo de pesquisa realizada.

2.4.10 Técnica de coleta de dados

Um total de 48 técnicas de coleta de dados foram mencionadas nos 31 artigos de pesquisa selecionados. As técnicas mencionadas são apresentadas na Tabela 2.7. A maioria dos artigos, cerca de 35%, utilizou questionários como técnica, por exemplo, para coletar informações dos estudantes ao longo da disciplina ou para avaliar o nível de satisfação dos estudantes ao utilizar determinada ferramenta. Programas construídos pelos estudantes ou outros tipos de exercícios (e.g., exercícios de múltipla escolha) também foram utilizados por cerca de 35% dos artigos. As avaliações realizadas pelos estudantes durante as disciplinas foram utilizadas por cerca de 29% dos artigos. Também foram utilizados os registros de uso de ambientes e ferramentas, tais como blogs, sistemas de administração de cursos (e.g., Moodle) ou as ferramentas produzidas nos artigos (23%), e pré e pós-testes específicos para os artigos em questão (16%). A técnica de coleta de dados menos mencionada utiliza os materiais utilizados na disciplina (6%), tais como os livros didáticos adotados na disciplina.

Tabela 2.7. Técnicas de coleta de dados apresentadas nos artigos.

Técnica de coleta de dados	#Artigos
Questionários	11
Programas construídos e/ou outros exercícios realizados pelos estudantes	11
Avaliação da disciplina	9
Registros de utilização dos ambientes e ferramentas	7
Pré e/ou pós-testes	5
Material utilizado na disciplina	2
Outros (notas em outras disciplinas diferentes de programação; auto-avaliação; debate com os estudantes; ocorrência de eventos)	4

2.4.11 Principais objetivos e resultados obtidos

Nesta seção apresentamos os principais objetivos e resultados obtidos pelos artigos selecionados.

2.4.11.1 Ferramentas para os processos de ensino e aprendizagem e de avaliação

Os artigos de pesquisa que foram classificados nessa categoria apresentam ferramentas que foram desenvolvidas para auxiliar os processos de ensino e aprendizagem de programação e também que foram criadas ou aplicadas ao processo de avaliação dos estudantes [SIMON, 2007]. Dos 31 artigos selecionados, 16 foram classificados como ferramentas para os processos de ensino e aprendizagem e 3 artigos como ferramentas de avaliação.

A grande maioria dos artigos tinha como objetivo apresentar uma ferramenta (ou uma melhoria dessa ferramenta) e avaliar a sua eficácia através da realização de exercícios, avaliações ou testes com os estudantes. Além disso, muitas vezes, a ferramenta também era avaliada de acordo com um questionário de opinião com os estudantes.

Alguns resultados exibidos nos artigos são os seguintes:

- utilizar uma ferramenta para escrever textos sobre tópicos básicos de programação está relacionada a melhoria no aprendizado dos estudantes [LOPES; DE MENEZES, 2006];
- utilizar um sistema tutor inteligente parece influenciar positivamente o desempenho de estudantes iniciantes em programação [RAABE; GIRAFFA, 2006; RISSOLI; SANTOS, 2013];
- utilizar um jogo de computador para o ensino de programação não promoveu melhores resultados na aprendizagem dos estudantes, porém aqueles considerados com maior dificuldade beneficiaram-se com a utilização do jogo [DE JESUS; RAABE, 2010];
- estudantes de programação sentem-se frustrados quando compilam inúmeras vezes um programa sem sucesso, quando encontram muitos erros nos programas ou quando passam muito tempo na resolução de um mesmo problema [IEPSEN; BERCHT; REATEGUI, 2011]; e

- a análise automática de programas dos estudantes iniciantes mostrou que eles possuem problemas sintáticos, não conseguem utilizar os construtos da linguagem corretamente e constroem programas que estão distantes de uma solução considerada adequada [SILVA et al., 2014].

2.4.11.2 Técnicas de ensino e de aprendizagem

Os artigos de pesquisa que foram classificados nessa categoria apresentam técnicas que foram criadas ou aplicadas ao processo de ensino e/ou aprendizagem de programação, incluindo a aplicação de ferramentas já existentes [SIMON, 2007]. Dos 31 artigos selecionados, 9 foram classificados como técnicas de ensino-aprendizagem.

Os nossos resultados mostram que a grande maioria dos artigos tinha como objetivo utilizar técnicas, tais como Programação por demonstração ou Aprendizado baseado em problemas, ou utilizar ambientes de desenvolvimento já existentes, tais como o Scratch e o Alice. Em alguns artigos, também encontramos a combinação de técnicas e ferramentas para os processos de ensino e aprendizagem de programação para iniciantes.

Destacamos alguns resultados obtidos na análise dos artigos selecionados:

- há indícios de que a técnica da programação por demonstração contribui para o desenvolvimento do raciocínio lógico dos estudantes [FERREIRA; GONZAGA; SANTOS, 2009];
- combinar a abordagem de ensino baseado em problemas com o ambiente Alice contribuiu de forma significativa para tornar os estudantes mais ativos e independentes e a trabalharem de forma colaborativa [GONDIM; AMBRÓSIO; COSTA, 2009];
- há uma influência positiva da utilização do Scratch, já que os processos de ensino e aprendizagem de programação parece ser facilitado em alguns aspectos, tais como tempo reduzido de aula expositiva, fácil combinação de comandos na construção de um programa, ausência de erros sintáticos e linguagem de programação em português [AURELIANO; TEDESCO, 2012a]; e
- os estudantes de uma disciplina ministrada na EaD tiveram opiniões divergentes em relação à utilização do Scratch, enquanto metade da turma

avaliou positivamente a utilização da ferramenta, a outra metade avaliou negativamente a ferramenta [AURELIANO; TEDESCO, 2013].

2.4.11.3 *Habilidade/aptidão*

A categoria de habilidade/aptidões envolve os artigos que relatam as características que fazem os estudantes serem bons em programação, como eles podem se preparar melhor e quais os processos que eles aplicam ao estudar para a disciplina [SIMON, 2007]. Dos 31 artigos selecionados, 3 foram classificados nesta categoria.

O primeiro artigo tem por objetivo avaliar o desempenho dos estudantes para explicar os altos índices de reprovação e desistências. Os resultados encontrados mostram que embora os estudantes novatos e repetentes na disciplina introdutória de programação tenham estudado a mesma quantidade de horas, os hábitos de estudo dos repetentes foram ineficazes. Estes hábitos envolveram por exemplo estudar passivamente exercícios resolvidos aliado à pouca ou nenhuma resolução de problemas, o que indica a necessidade de estratégias de ensino que engajem mais os estudantes de programação em seus processos de estudo [GUEDES, 2014].

O segundo artigo tem por objetivo verificar correlações entre a disciplina de programação e outras disciplinas do curso. Os resultados deste artigo mostram que há uma correlação positiva moderada entre a disciplina introdutória de Programação e a disciplina de Laboratório de Programação [SILVA; CAVALCANTE; DE BARROS COSTA, 2013].

O último artigo tem como objetivo examinar os hábitos de estudos dos estudantes iniciantes em programação. Os resultados encontrados mostram que a realização de exercícios e o estudo diário são os fatores mais relevantes para o bom desempenho destes estudantes [DE ARAUJO et al., 2013].

2.5 Implicações

Neste segundo capítulo apresentamos uma revisão da literatura relacionada aos processos de ensino e de aprendizagem de programação. Vimos que as disciplinas de programação são intrinsecamente difíceis, que estão comumente relacionadas a altas taxas de reprovação e de evasão. Os problemas encontrados pelos estudantes incluem, por exemplo, a utilização de variáveis, estruturas condicionais e de repetição e *arrays*. Contudo, a maior obstáculo encontrado pelos estudantes iniciantes em programação não

é o entendimento dos conceitos básicos de programação, mas a combinação e utilização adequada destes conceitos na construção de programas. Sendo assim, a maioria dos erros dos estudantes não resultam de falha no conhecimento dos construtos de uma linguagem de programação, mas resultam das dificuldades “*em colocar as peças do programa juntas*” [SPOHRER; SOLOWAY, 1989 p. 401]. Além disso, os estudantes em programação nem sempre sabem como estudar adequadamente para as necessidades de uma disciplina de programação. Eles empregam metodologias inadequadas enquanto estudam para a disciplina e, por não engajarem-se apropriadamente em seus momentos de estudo, eles acabam por estudar os materiais instrucionais disponíveis de maneira passiva ou superficial. Em consequência, tais estudantes tem resultados nas disciplinas de programação que estão longe dos resultados considerados satisfatórios.

Outro tópico abordado pela literatura está relacionado às habilidades que predizem o bom desempenho dos estudantes em programação. Entre outras coisas, a literatura mostra que estratégias de aprendizagem que fazem com que os estudantes estudem mais profundamente o material, as notas em disciplinas de matemática e ciências, a experiência prévia em programação e o tempo que os estudantes passaram programando em aulas de laboratório estão relacionadas ao desempenho dos estudantes em programação.

Ensinar programação também não é uma atividade simples e envolve que o professor tome uma série de decisões ao planejar uma disciplina de programação para um curso. Entre as decisões estão o paradigma e a linguagem de programação, a ferramenta de desenvolvimento e as práticas pedagógicas que devem ser utilizadas no decorrer da disciplina. Entre os paradigmas mais utilizados estão o procedural e orientado a objetos e entre as linguagens de programação as mais utilizadas continuam sendo Java, C e C++. Em relação às ferramentas utilizadas para o ensino de programação para iniciantes, foram citadas Alice, BlueJ, Greenfoot e Scratch. No que diz respeito às práticas pedagógicas utilizadas, em função das dificuldades vivenciadas pelos estudantes, é indicado que os professores privilegiem a combinação e o uso adequado das estruturas dos programas através da resolução de problemas.

Além disso, os iniciantes em programação não possuem o conhecimento específico e as habilidades de resolução de problemas que os profissionais mais experientes possuem. Entre as características citadas estão, por exemplo, que os mais experientes possuem esquemas de conhecimento eficientemente organizados e

especializados; organizam o conhecimento de acordo com a funcionalidade e não com a sintaxe da linguagem; reconhecem e adaptam facilmente esquemas; e são mais precisos e capazes de aplicar um grande conjunto de exemplos na resolução de problemas, entre outras coisas. Para adquirir todas estas habilidades, sabemos que não há outra maneira de percorrer esse caminho de iniciante a experiente a não ser a partir de muito estudo, dedicação e prática em suas vidas enquanto programadores.

Por fim, vimos que a literatura no Brasil aponta que as dificuldades encontradas nas disciplinas introdutórias de programação são as mesmas. Diante destas dificuldades, o interesse da comunidade pela área de ensino de programação vem crescendo ao longo dos anos. Além disso, os dados revelaram uma nítida preferência da comunidade pela criação de ferramentas para auxiliar nos processos de ensino e aprendizagem e de avaliação. Dentre estes trabalhos, vimos que utilizar uma ferramenta para escrever textos sobre tópicos básicos de programação e um sistema tutor inteligente estão associados a melhoria no aprendizado dos estudantes [LOPES; DE MENEZES, 2006; RAABE; GIRAFFA, 2006; RISSOLI; SANTOS, 2013]; e que estudantes de programação sentem-se frustrados ao compilar inúmeras vezes um programa sem sucesso, ao encontrar muitos erros nos programas ou ao passar muito tempo na resolução de um mesmo problema [IEPSEN et al., 2011]. Em segundo lugar, ficaram os trabalhos que propõem abordagens para ensino-aprendizagem. Dentre eles, vimos que combinar a abordagem de ensino baseado em problemas com o Alice [GONDIM; AMBRÓSIO; COSTA, 2009] e utilizar o Scratch trazem resultados positivos para os processos de ensino e aprendizagem de programação [AURELIANO; TEDESCO, 2012a]. Em terceiro lugar, ficaram os trabalhos na categoria habilidade e aptidão, que relatam características dos estudantes em programação, como eles podem se preparar melhor e quais os processos que eles aplicam ao estudar para a disciplina. Dentre eles, vimos que os hábitos de estudo dos estudantes repetentes em uma disciplina de programação são ineficazes [GUEDES, 2014]; e que a realização de exercícios e o estudo diário são os fatores mais relevantes para o bom desempenho dos estudantes de programação [DE ARAUJO et al., 2013].

Dentro deste âmbito, veremos no próximo capítulo o referencial teórico que embasa o trabalho desta tese.

3 Referencial teórico

Apresentamos neste terceiro capítulo a fundamentação teórica deste trabalho. Na Seção 3.1, descrevemos o processo de programação definido pelo *framework Stepwise Improvement* (FSI) utilizado para a construção de um conjunto de exemplos utilizados como parte do material instrucional produzido nesta tese. Na Seção 3.2, apresentamos a Teoria da Carga Cognitiva, juntamente com os exemplos trabalhados, técnica utilizada nos momentos iniciais de aprendizado dos estudantes de programação. Na Seção 3.3, apresentamos a teoria das auto-explicações (AE) que foi utilizada para auxiliar os estudantes em seus processos de estudo. Descrevemos na Seção 3.4, a teoria da Aprendizagem Cognitiva juntamente com os métodos de ensino que foram utilizados na definição da abordagem apresentada nesta tese. Na Seção 3.5, apresentamos um conjunto de trabalhos relacionados a esta tese. Por fim, na Seção 3.6 estão as considerações finais do capítulo.

3.1 *Framework Stepwise Improvement*

O processo de programação definido pelo FSI [CASPERSEN, 2007; CASPERSEN; KÖLLING, 2009] unifica ideias do *Stepwise Refinement* [WIRTH, 1971] e dos métodos ágeis de desenvolvimento de software⁵. O *Stepwise Refinement* caracteriza o processo de programação como uma atividade unidimensional na qual a construção de programas concretos acontece a partir de programas abstratos através de transformações sucessivas do programa. Assim, a implementação de código executável é realizada a partir de especificações por meio de uma sequência de passos de refinamento. No entanto, de acordo com Caspersen (2007), o *Stepwise Refinement* segue uma abordagem estritamente *top-down* e, por isso, para que ele seja colocado em prática é necessário que os requisitos e a especificação do sistema sejam conhecidos desde o início do processo de desenvolvimento.

Para resolver esta limitação, o FSI define que a construção de código executável seja feita a partir de uma abordagem que está alinhada com os métodos ágeis de desenvolvimento de software, batizada por seus proponentes de ilhas crescentes de funcionalidade. Diferentemente das tradicionais abordagens *top-down* e *bottom-up*, a abordagem das ilhas crescentes de funcionalidade define que o processo de

⁵ Agile Alliance. Disponível em: <<http://bit.ly/1JU8jaZ>>. Acesso em: 06 out. 2015.

desenvolvimento de um sistema deve se iniciar a partir de um pequeno conjunto de funcionalidades que vai crescendo gradualmente.

O FSI surgiu da necessidade de se trazer para a educação em programação, essa noção de processo de desenvolvimento de programas através das ilhas crescentes de funcionalidade, muitas vezes negligenciada. Segundo os seus proponentes, os livros e cursos introdutórios de programação focam principalmente na apresentação de construtos, de conceitos de programação e de exemplos de programas estáticos, sem endereçar a questão da trajetória percorrida pelo programador para se chegar a uma determinada solução. Eles afirmam que:

“[o] fato de que todos iniciamos com programas incorretos e incompletos quando começamos a endereçar um problema que vai sendo lentamente modificado para melhorar e estender nossa implementação até que cheguemos em uma solução aceitável, parece ser jogada para baixo do tapete como se fosse um segredo embaraçoso que não devesse ser mencionado.”

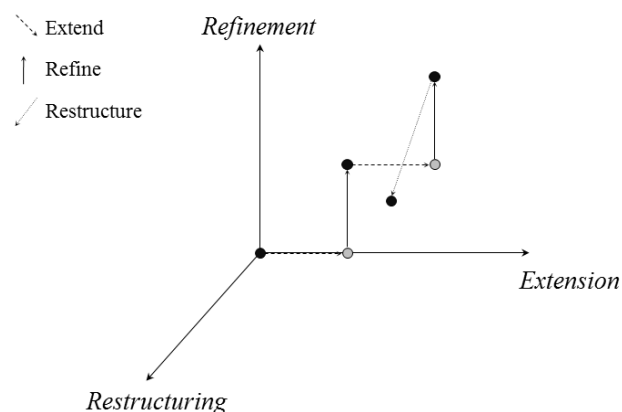
[CASPERSEN; KÖLLING, 2009 p. 3]

Na tentativa de preencher esta lacuna, o FSI foi definido por seus autores como um *framework* conceitual que descreve a atividade de programação como um processo de construção iterativo e incremental que engloba as três atividades definidas a seguir:

- **Extensão** é a atividade que ocorre quando a especificação é estendida de maneira a cobrir mais casos (de uso);
- **Refinamento** é a atividade que ocorre quando o código abstrato é modificado de maneira a se construir código executável que implemente a especificação corrente; e
- **Reestruturação** é a atividade que ocorre quando é realizada uma melhoria dos aspectos não funcionais do programa, no entanto essa modificação não envolve uma mudança no comportamento aparente do programa.

Estas atividades são organizadas em um espaço tridimensional que é percorrido pelos programadores a cada novo passo na trajetória do processo de desenvolvimento de um programa. A Figura 3.1 ilustra uma sequência de desenvolvimento de acordo com o FSI que consiste em cinco passos ordenados: extensão, refinamento, extensão, refinamento e reestruturação.

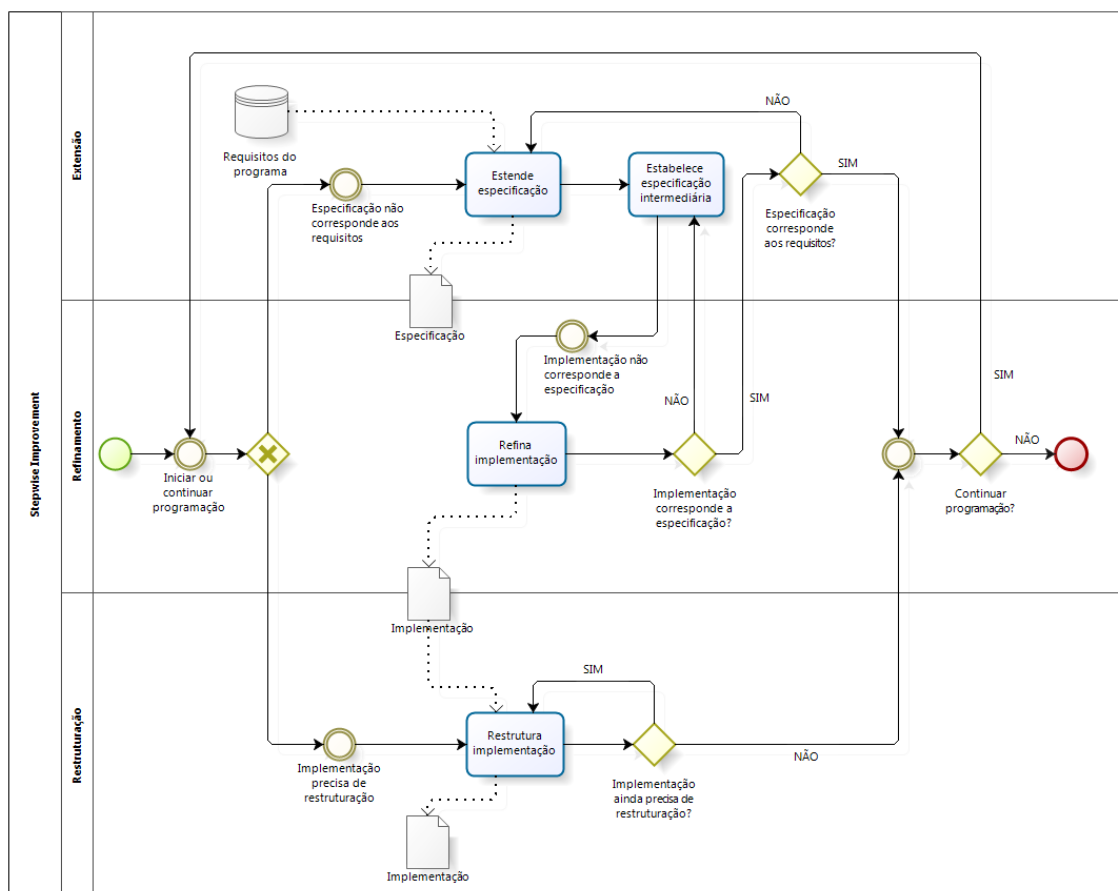
Figura 3.1. Um exemplo de sequência de desenvolvimento de acordo com o FSI.



Fonte: própria autora.

Para que entendamos adequadamente como ocorre o processo de programação iterativo e incremental de acordo com o FSI, faz-se necessário apresentar um conjunto de definições [CASPERSEN, 2007 p. 92-104] que foi utilizado posteriormente na estruturação deste trabalho. Na Figura 3.2 uma representação diagramática do FSI em notação BPMN [OMG, 2011]. Nesta figura, o FSI é representado como um processo onde cada uma de suas três atividades é representada em uma faixa, juntamente com as suas (sub-)atividades associadas. Em poucas palavras, escrever um programa de acordo com o FSI significa que o programador terá que partir dos requisitos do programa e construir a especificação do programa a partir destes requisitos. Na sequência, a partir da especificação do programa são estabelecidas especificações intermediárias que serão utilizadas para o refinamento da implementação. A especificação vai sendo estendida até que corresponda aos requisitos do programa. Da mesma forma, enquanto a implementação não corresponder à especificação estendida, especificações intermediárias serão estabelecidas e a implementação vai sendo refinada. Ademais, durante este processo, uma reestruturação pode precisar ser realizada. Neste caso, a implementação é reestruturada até que a execução desta atividade não seja mais necessária. A seguir, descreveremos estas atividades em detalhes.

Figura 3.2. Representação do FSI utilizando a notação BPMN.



Fonte: própria autora.

O *framework* apresentado na Figura 3.2 é composto por duas unidades atômicas, os mecanismos e suas especificações. Um mecanismo possui um conjunto de requisitos, uma especificação e uma implementação. O conjunto de requisitos descreve o conjunto de requisitos final do mecanismo. A **especificação** descreve o comportamento desejado da implementação atual do mecanismo. O FSI não estabelece uma regra para a representação de uma especificação e, por isso, ela pode ser representada de diferentes maneiras, através de um conjunto de casos de teste, de alguma linguagem natural estruturada ou implicitamente através de convenções de nomes, por exemplo. A implementação do mecanismo, por sua vez, é o seu código executável. De acordo com o FSI, o programa é um conjunto de mecanismos.

No que diz respeito à **extensão**, um mecanismo pode ser classificado como total ou parcial. Ele é total se a especificação da sua implementação corresponde ao seu conjunto de requisitos. Um mecanismo é parcial se não é total. O primeiro objetivo do programador no processo de desenvolvimento incremental descrito pelo FSI é aprimorar

a especificação do mecanismo até que esta especificação corresponda adequadamente ao seu conjunto de requisitos. Para atingir este objetivo, o programador estabelece especificações intermediárias que vão sendo estendidas incrementalmente, até que o mecanismo parcial se torne total.

Considerando que cada especificação intermediária do mecanismo possui uma implementação correspondente, ao longo de um determinado intervalo de tempo, o mecanismo pode possuir diferentes implementações. Desta maneira, uma implementação que está mais próxima do início deste intervalo de tempo corresponde a uma parcela menor do conjunto de requisitos do mecanismo, enquanto que uma implementação mais próxima do final do mesmo intervalo corresponde a uma porção maior deste conjunto de requisitos. Quando esta noção de desenvolvimento incremental é realizada através de uma sequência de extensões intermediárias, ela é representada através de uma trajetória de desenvolvimento uniforme. A trajetória de desenvolvimento uniforme de um mecanismo é uma sequência de extensões intermediárias s_i , com $0 \leq i \leq n$, onde s_0 representa a extensão inicial quando a especificação é vazia e o mecanismo ainda não faz nada e s_n representa a extensão final quando a especificação corresponde ao conjunto de requisitos e, por conseguinte, o mecanismo é total. Dentro desta perspectiva, o objetivo de qualquer processo de desenvolvimento é possibilitar que programadores percorram o caminho de s_0 até s_n através de sucessivas extensões intermediárias, de forma que, a cada incremento na trajetória de desenvolvimento do mecanismo, s_{i+1} atende a mais requisitos do que s_i .

Quando as especificações intermediárias ainda não possuem uma implementação correspondente, o mecanismo pode ser classificado como consistente ou inconsistente. Ele é consistente quando a sua implementação corresponde a sua especificação, ou seja, se a sua implementação se comporta de acordo com o que está definido em sua especificação. Sendo assim, o segundo objetivo do programador no processo de desenvolvimento incremental descrito pelo FSI é aprimorar a implementação do mecanismo de maneira que ela corresponda a sua especificação. Para atingir esse objetivo, o programador vai refinando a implementação do mecanismo de maneira que ele passe de inconsistente a consistente.

Além dos aprimoramentos da especificação e da implementação do mecanismo descritas anteriormente, o FSI prevê a **reestruturação** como um outro tipo de aprimoramento da implementação do mecanismo, desta vez que interfere na sua

estrutura interna, porém sem alterar o seu comportamento externo. A reestruturação, ou refatoramento como é mais conhecido, é segundo Fowler e Beck (1999), “*uma maneira disciplinada de limpar código que minimiza as chances de se introduzir erros. Em essência, quando você refatora você está aprimorando o projeto de código depois que ele [o código] já foi escrito.*” [FOWLER; BECK, 1999 p. 9]. Ela ocorre quando, por exemplo, percebe-se “*cheiros ruins*” na implementação, tais como um trecho de código que é repetido diversas vezes e que pode ser substituído por uma nova função (ou um novo método, no caso de uma linguagem orientada a objeto).

Para que o FSI seja utilizado adequadamente para a educação em programação, Caspersen e Kölling (2009) defendem a sua aplicação utilizando passeios guiados ao invés de deixar os estudantes realizarem caminhadas aleatórias. Para conduzir os passos dos estudantes no espaço de programação oferecido pelo *framework*, os autores afirmam que os professores devem estar preocupados com o nível de orientação adequado que fornecerão durante a instrução. Isso garante que aspectos importantes da educação em programação sejam considerados: treinar os alunos nas habilidades de programação e, ao mesmo tempo, manter a carga cognitiva enquanto os estudantes estão estudando sob controle.

Neste sentido, o FSI traz uma contribuição significativa à educação de programação, fornecendo orientação nas três atividades que engloba. No que diz respeito às atividades de extensão, refinamento e reestruturação, ele fornece orientação na maneira como o material instrucional para a disciplina de programação pode ser estruturado. Por exemplo, os vídeos instrucionais *The Joy of Code*⁶ que ensinam a linguagem de programação Java utilizando a ferramenta *Greenfoot* e o livro *Introduction to Programming with Greenfoot* [KÖLLING, 2010] seguem as três atividades do FSI na maneira como foram estruturados⁷. Além disso, Caspersen (2007) também utilizou o *framework* para definir um processo de programação orientado a objetos para iniciantes que provê orientação sobre a atividade de refinamento.

3.2 Teoria da carga cognitiva

A teoria da carga cognitiva está fundamentada na arquitetura do sistema cognitivo humano e na maneira que o processo de aprendizado humano acontece [CLARK et al.,

⁶ *The Joy of Code*. Disponível em < <http://bit.ly/1jbTbHq> >. Acesso em: 03 nov. 2015.

⁷ Informação fornecida pelo autor principal do FSI, Michael Caspersen.

2005; SWELLER, 1994; SWELLER; VAN MERRIENBOER; PAAS, 1998]. O conhecimento humano é organizado em padrões significativos de informação [BRANSFORD et al., 2000] ou elementos do conhecimento [SWELLER et al., 1998] denominados esquemas. Os esquemas são estruturas de memória que permitem que uma grande quantidade de informação seja tratada como se fosse um único elemento [CLARK et al., 2005]. Segundo Caspersen (2007), possuímos dois tipos principais de esquemas, os esquemas de conhecimento e os esquemas de habilidades. Os esquemas de conhecimento correspondem ao conhecimento declarativo, ou seja, ao conhecimento sobre as coisas [BIGGS; TANG, 2011]. Os esquemas de habilidades, por sua vez, correspondem ao conhecimento procedural (ou do funcionamento), ou seja, ao conhecimento que permite a utilização ou a aplicação apoiada pela compreensão do conhecimento declarativo [BIGGS; TANG, 2011]. Um programador, por exemplo, inclui em seus esquemas de conhecimento informações sobre os construtos da linguagem, tal como saber que existe uma estrutura de repetição na linguagem de programação C que é chamada `for`, que serve para ser utilizada quando queremos repetir um trecho da implementação de um programa por um determinado número de vezes e que a sua forma (sintaxe) é `for(inicialização; teste; incremento) [corpo do for]`, onde os valores `inicialização`, `teste` e `incremento` vão depender da sua utilização no programa. Aliado a esse conhecimento declarativo, um programador possui também os esquemas de habilidade que incluem informações que permitem que ele desenvolva a estrutura de repetição `for` a partir de sua especificação e aplique-a apropriadamente na construção de um programa com valores adequados para `inicialização`, `teste` e `incremento`, por exemplo.

Para processar e armazenar estes esquemas, o cérebro humano possui duas memórias, a memória de trabalho e a memória de longo prazo, que colaboram intensamente uma com a outra [CLARK et al., 2005]. A memória de trabalho é a parte ativa do cérebro, onde acontecem o pensamento consciente e o aprendizado do ser humano [CLARK et al., 2005; SWELLER et al., 1998]. Ela é extremamente limitada e tem capacidade para processar apenas um número de 7 ± 2 esquemas por vez [MILLER, 1956]. Portanto, é importante considerar essas limitações no projeto instrucional, para que os alunos não sejam sobrecarregados enquanto estão aprendendo sobre um determinado assunto.

Contudo, se a memória de trabalho do ser humano tem uma capacidade tão limitada e constante, como é que pessoas mais experientes em um determinado assunto conseguem ter um desempenho superior quando comparadas àquelas menos experientes? A resposta a esta pergunta resulta dos estudos das diferenças entre os iniciantes e os experientes. Neste âmbito, um famoso experimento que foi realizado com jogadores de xadrez e citado por diferentes referências [BRANSFORD et al., 2000; CLARK et al., 2005; SWELLER et al., 1998] buscava entender as diferenças entre jogadores de diferentes níveis de conhecimento [CHASE; SIMON, 1973]. No experimento, os pesquisadores pediram que três diferentes tipos de jogadores de xadrez, um jogador extremamente experiente, um jogador muito bom (mas não tão experiente quanto o primeiro) e um jogador iniciante, olhassem diferentes configurações de tabuleiros por 5 segundos e depois tentassem montar a configuração memorizada em um segundo tabuleiro. Os pesquisadores contaram quantas vezes os jogadores precisaram rever o tabuleiro original até que conseguissem remontar toda a configuração no segundo tabuleiro. Na primeira tentativa, o jogador extremamente experiente conseguiu colocar em média 16 peças corretas, enquanto o jogador muito bom conseguiu colocar 8 peças e o jogador iniciante apenas 4 peças. Ao final, na média, o jogador extremamente experiente conseguiu reproduzir o tabuleiro original em 3 ou 4 tentativas, o jogador muito bom em 5 ou 6 tentativas e o jogador iniciante levou até 14 tentativas. No entanto, estes resultados foram obtidos somente quando as peças do tabuleiro estavam organizadas em configurações que faziam sentido (i.e., configurações possíveis) para um tabuleiro de xadrez. Quando as peças estavam organizadas aleatoriamente, os três tipos de jogadores apresentaram resultados similares, conseguindo se lembrar apenas de 2 ou 3 peças corretamente.

Destes estudos e muitos outros que pesquisaram sobre as diferenças entre experientes e iniciantes⁸, os pesquisadores encontraram evidências da importância da memória de longo prazo e dos esquemas para o desenvolvimento da habilidade intelectual de uma pessoa [SWELLER et al., 1998]. Ao contrário da memória de trabalho, a memória de longo prazo possui uma grande capacidade de armazenamento e representa o principal repositório de conhecimento do cérebro humano [CLARK et al., 2005]. Embora ela seja a parte inerte do nosso cérebro e não esteja diretamente

⁸ Para uma lista mais extensa, consultar [BRANSFORD et al., 2000].

envolvida nos processos de pensamento consciente e de aprendizagem, é na memória de longo prazo que estão armazenados os esquemas que vamos construindo ao longo de nossas vidas, a medida que adquirimos conhecimento e experiência sobre os mais diversos assuntos [CLARK et al., 2005; SWELLER et al., 1998].

Como dito anteriormente, os esquemas são as estruturas de memória que permitem que um grande número de elementos de informação sejam tratados como se fossem um só. No caso do experimento dos jogadores de xadrez, de acordo com Bransford et al. (2000), a habilidade que o jogador extremamente experiente possui de lembrar-se mais rapidamente dos tabuleiros com configurações possíveis está relacionada à maneira como os esquemas estão organizados na sua memória de longo prazo. Por conta de sua experiência, o jogador mais experiente consegue agrupar um conjunto muito maior de peças de xadrez em uma configuração possível que é determinada em função do seu conhecimento tático do jogo. Consequentemente, ele consegue se lembrar muito mais rapidamente dessas configurações, uma vez que elas representam padrões significativos para ele. Os outros dois jogadores, ao contrário, não possuem a mesma organização de esquemas que o jogador mais experiente possui e, portanto, não conseguem se lembrar tão facilmente das configurações possíveis do tabuleiro de xadrez.

O desenvolvimento dos esquemas ocorre de duas maneiras no sistema cognitivo humano, a primeira através da aquisição de esquemas e a outra através da recodificação ou *chunking* de esquemas. A aquisição de esquemas é um processo construtivo que acontece à medida que a nossa habilidade sobre determinado assunto vai sendo desenvolvida [SWELLER et al., 1998]. Assim, enquanto estamos aprendendo sobre um determinado assunto, a nova informação que adquirimos é processada pela memória de trabalho [CLARK et al., 2005]. Em seguida, ela é integrada aos esquemas existentes em nossa memória de longo prazo que foram recuperados e decodificados na memória de trabalho [CASPERSEN, 2007]. Consequentemente, a medida que vamos ganhando experiência em um determinado assunto, vamos produzindo um número de esquemas cada vez maiores e mais complexos pela integração de novas informações aos esquemas que possuímos na nossa memória de longo prazo.

O processo de recodificação ou *chunking*, por sua vez, ocorre quando combinamos muitos esquemas contendo poucas informações em poucos esquemas contendo um número maior de informações, o que resulta em esquemas maiores e mais

complexos [CASPERSEN, 2007]. Consequentemente, com a mesma capacidade de memória de trabalho que todos os seres humanos possuem (que consiste em 7 ± 2 esquemas), mas com esquemas que são maiores e mais complexos, a memória de trabalho pode ser utilizada de maneira mais eficiente, uma vez que os mais experientes conseguem processar um número muito maior de informações do que os menos experientes conseguem. A capacidade da memória de trabalho é então aumentada pelo aumento do número de informações contidas nos esquemas. Desta maneira, embora não esteja diretamente envolvida nos processos de pensamento consciente e de aprendizagem, a memória de longo prazo constitui-se em um elemento importante para as capacidades de raciocinar e de resolver problemas do ser humano [CLARK et al., 2005].

Ainda de acordo com Clark et al. (2005), em decorrência dos esquemas maiores e mais complexos, os mais experientes são mais habilidosos do que os iniciantes. Desta maneira, os mais experientes conseguem facilmente realizar atividades consideradas complexas pelos iniciantes e que exigem destes últimos um grande esforço. Ademais, como possuem um repositório com uma grande quantidade de esquemas, ao aprender novas habilidades no seu domínio, os mais experientes conseguem processar um número maior de informações e também guiar o seu próprio processo de aprendizado. Ao contrário dos experientes, os iniciantes não possuem o mesmo conjunto de esquemas que os experientes possuem, então os ambientes de ensino precisam considerar tais restrições. Uma maneira de reduzir a sobrecarga da memória de trabalho nessa fase inicial de aprendizado é através da introdução de atividades que possibilitem que os iniciantes adquiram esquemas que poderão ser utilizados posteriormente na resolução de problemas. Por exemplo, enquanto estão aprendendo programação, os estudantes iniciantes na disciplina podem ser sobrecarregados ao iniciarem o aprendizado pela proposição de extensas listas de exercícios. Veremos mais adiante que eles podem aprender mais eficientemente através da aplicação de exemplos trabalhados, uma atividade prática alternativa que os ajuda na aquisição de novos esquemas antes que eles passem a resolver problemas de maneira mais intensa.

Depois que já adquirimos os esquemas de conhecimento e habilidades que nos permitem resolver problemas adequadamente, uma outra maneira de lidar com os recursos limitados da memória de trabalho é através da automatização destes esquemas [CLARK et al., 2005]. A recuperação de esquemas de maneira automática e fluente é

uma das mais importantes características que adquirimos a medida que vamos ganhando experiência [BRANSFORD et al., 2000]. Conforme vamos desempenhando nossas atividades de maneira mais automática e fluente, conseguimos resolver problemas com o mínimo de pensamento consciente, causando a liberação dos recursos da memória de trabalho para a integração de esquemas cada vez mais complexos [CLARK et al., 2005]. Por exemplo, quando uma pessoa começa a dirigir, ela possui muitas coisas para administrar em um carro: os três espelhos retrovisores, os três pedais (para carros sem câmbio automático, temos a embreagem, o freio e o acelerador), o câmbio, além de toda a movimentação que se encontra ao seu redor. Neste começo, são tantas coisas para prestar atenção que muitas vezes é difícil conversar com uma outra pessoa no carro. No entanto, à medida que esta pessoa vai ganhando experiência, controlar os três espelhos retrovisores e os três pedais já não é mais tão complicado. Passar a marcha, sem esquecer de pisar na embreagem também não é mais um problema. Quando a pessoa chega neste ponto de automatismo enquanto dirige um carro, ela livra a sua memória de trabalho para outras atividades e, assim, nesse momento, já é mais fácil conversar com outras pessoas que estão no carro com ela ou conseguir prestar atenção na música que toca no rádio.

Desta explanação sobre a arquitetura cognitiva humana, podemos perceber que lidar com as limitações da memória de trabalho é o grande obstáculo a ser vencido quando propomos, enquanto professores, ambientes instrucionais para os nossos estudantes [CASPERSEN, 2007]. Dentro deste âmbito, o objetivo principal da teoria da carga cognitiva é então liberar a memória de trabalho, que possui capacidade reduzida, do esforço mental irrelevante e utilizar os recursos liberados na integração dos novos conhecimentos e habilidades adquiridos aos esquemas existentes na memória de longo prazo [CLARK et al., 2005]. Desta forma, é possível utilizar a capacidade da memória de trabalho com elementos que contribuam para a aquisição de esquemas e, por conseguinte, para obtenção de conhecimento por parte do estudante.

De acordo com Caspersen (2007), a carga cognitiva pode ser definida como a carga na memória de trabalho durante os processos de resolução de problema, pensamento e raciocínio. A teoria da carga cognitiva diferencia esta carga na memória de trabalho em três tipos [CASPERSEN, 2007; CLARK et al., 2005]:

- Carga cognitiva irrelevante: é aquela carga cognitiva resultante de atividades instrucionais que dificultam, ao invés de contribuir, para o

aprendizado pela sobrecarga da capacidade da memória de trabalho. Desta maneira, constitui-se no esforço mental improdutivo que não leva ao aprendizado;

- Carga cognitiva relevante: é aquela carga cognitiva imposta pelas atividades instrucionais e que contribui para a construção de novos esquemas e, conseqüentemente, para um melhor resultado de aprendizagem. Desta maneira, constitui-se no esforço mental produtivo necessário para a criação efetiva de novos esquemas; e
- Carga cognitiva intrínseca: é aquela carga cognitiva que resulta da complexidade dos conhecimentos e habilidades sendo aprendidos e que é determinada pelos objetivos instrucionais. É aquela carga cognitiva que não pode ser reduzida sem que o entendimento do conteúdo seja prejudicado.

Como dito anteriormente, o objetivo da teoria da carga cognitiva é liberar a memória de trabalho de esforço mental irrelevante e aplicá-lo para o trabalho necessário para integrar novos conhecimentos e habilidades aos esquemas presentes na memória de longo prazo. Para tanto, a teoria da carga cognitiva define que é necessário reduzir a carga cognitiva que sobrecarrega a memória de trabalho e dificulta o aprendizado, aumentar a carga cognitiva que contribui para a construção de esquemas e contribuir para o aprendizado e administrar a carga cognitiva resultante da complexidade do assunto sendo aprendido, conforme ilustrado na Figura 3.3.

Figura 3.3. Objetivo principal da Teoria da carga cognitiva.



Fonte: [CLARK et al., 2005].

Vimos anteriormente que programação é uma disciplina intrinsecamente difícil pelas muitas dificuldades pelos quais os estudantes passam ao iniciar suas vidas como programadores. Fazer com que os estudantes iniciantes em programação comecem o aprendizado desta disciplina resolvendo extensas listas de exercícios pode atrapalhá-los

pela sobrecarga da memória de trabalho. Veremos mais adiante, na Seção 3.2.1, que uma prática alternativa à resolução de problemas é fazer com que os alunos estudem exemplos trabalhados antes de passarem a solucionar as listas de exercícios. Tal atividade aumenta a carga cognitiva relevante, facilitando a aquisição de esquemas e, em consequência, contribui para o aprendizado.

3.2.1 Exemplos trabalhados

Uma maneira de aumentar a carga cognitiva relevante é através do estudo exemplos trabalhados (*worked examples*), um caso especial de exemplos, que levam à construção de esquemas na memória de longo prazo [CLARK et al., 2005]. De acordo com Clark et al. (2005), os exemplos trabalhados são uma sequência de passos que demonstram a realização de uma tarefa em particular ou a solução de um problema específico. Eles fornecem a solução para um problema que foi produzida por um profissional experiente e que pode ser utilizada por um estudante durante seus momentos de estudo [ATKINSON et al., 2000]. Os exemplos trabalhados são um tipo de material instrucional comprovadamente eficiente para estudantes iniciantes em diferentes disciplinas, auxiliando-os na construção de novos esquemas enquanto eles estão adquirindo um novo conhecimento ou habilidade [BENASSI et al., 2014; CLARK et al., 2005]. Exemplos são considerados, tanto por estudantes quanto por professores, o tipo de material instrucional mais útil para os processos de ensino e aprendizagem em linguagem de programação [LAHTINEN et al., 2005]. Ademais, eles foram recomendados pelo Departamento de Educação dos EUA como umas das sete práticas para melhoria dos processos de ensino e aprendizagem neste país [PASHLER et al., 2007].

Aprender a partir de exemplos trabalhados tem um papel fundamental nas fases iniciais do processo de aquisição da habilidade cognitiva, em diferentes domínios tais como Matemática, Física e Programação [ATKINSON et al., 2000; RENKL, 2002; VANLEHN, 1996]. Quando os estudantes estão iniciando a aquisição de habilidades para resolver problemas em atividades intelectuais, a prática de resolução de problemas de maneira exclusiva, realizada sem nenhum tipo de orientação, é ineficiente do ponto de vista do aprendizado [KIRSCHNER; SWELLER; CLARK, 2006]. De acordo com Kirschner et al. (2006), esta atividade resulta em uma alta carga cognitiva para os estudantes iniciantes, pois ao invés de se construir um esquema que pode ser aplicado

posteriormente para se resolver problemas, os estudantes precisam usar todo o esforço cognitivo unicamente para encontrar uma resposta para a solução do problema. Como a busca pela solução do problema ocorre na memória de trabalho que, como dissemos anteriormente, possui uma capacidade limitada para processamento da informação, ela pode ser demasiadamente sobrecarregada [KIRSCHNER et al., 2006].

Substituir exercícios por exemplos trabalhados, em contrapartida, reduz a carga cognitiva irrelevante e direciona a atenção dos estudantes para que eles percebam os diferentes passos de resolução do problema e construam os esquemas necessários a esta atividade a partir dos exemplos [SKUDDER; LUXTON-REILLY, 2014]. Consequentemente, os estudantes que utilizam exemplos trabalhados em conjunto com problemas podem obter resultados equivalentes ou melhores do que aqueles que utilizam apenas problemas, empregando menos tempo e esforço [SWELLER; COOPER, 1985]. Dentro deste contexto, uma referência citada frequentemente na literatura é a de Zhu e Simon (1987) que relatam a experiência de um curso de três anos de Matemática em uma escola na China que foi completado em apenas dois anos com a substituição da prática unicamente através de resolução de problemas pela prática da resolução de problemas associada aos exemplos trabalhados. Outro caso é dos estudantes em um experimento realizado por Paas (1992) na área de Estatística que comprovou que a instrução que enfatiza a prática a partir de exemplos trabalhados é mais eficiente, requer menos esforço por parte dos estudantes e apresenta resultados superiores com menos tempo de estudo. A este fenômeno positivo de utilização dos exemplos trabalhados como prática alternativa à resolução de problemas, que é baseado na teoria da carga cognitiva, dá-se o nome de efeito do exemplo trabalhado [KIRSCHNER et al., 2006].

Segundo Kirschner et al. (2006), estudar exemplos trabalhados é uma atividade de aprendizagem benéfica aos iniciantes em qualquer assunto. A partir do estudo de exemplos trabalhados, os estudantes aprendem a reconhecer padrões de sequências de passos para resolução de problemas. Como resultado, estes estudantes adquirem esquemas de resolução de problemas que correspondem aos padrões identificados. Os esquemas construídos são, posteriormente, codificados e armazenados na memória de longo prazo de onde podem ser recuperados posteriormente em momentos de resolução de problemas [CLARK et al., 2005].

Entretanto, quando os estudantes estão mais experientes no assunto e podem resolver os problemas sem esforço, estudar exemplos trabalhados redundantes e incorporar o conhecimento adquirido aos esquemas existentes gera uma carga cognitiva maior do que com a resolução de problemas [KALYUGA et al., 2003]. Neste momento, quando o estudo de exemplos passa a ser desnecessário, os estudantes devem ser encorajados a passar para a resolução de problemas onde eles poderão passar a aplicar os conhecimentos que foram incorporados aos seus esquemas. Assim, o estudo a partir de exemplos trabalhados é mais apropriado quando os estudantes são iniciantes ou têm pouca experiência no assunto abordado, enquanto que a resolução de problemas é mais apropriada para aqueles estudantes mais experientes [KALYUGA et al., 2001].

Exemplos trabalhados já foram utilizados em diferentes disciplinas como Física [CHI et al., 1989], Química [CRIPPEN; EARL, 2007], Probabilidade [ATKINSON; RENKL; MERRILL, 2003; BERTHOLD; RENKL, 2005a; RENKL, 1997], Matemática [WONG et al., 2002] e Programação [ABDUL-RAHMAN; DU BOULAY, 2014; BIELACZYK et al., 1995; CASPERSEN, 2007; MARGULIEUX; GUZDIAL; CATRAMBONE, 2012], entre outros. Eles são mais comumente apresentados de maneira estática, em formato textual [ABDUL-RAHMAN; DU BOULAY, 2014; BIELACZYK et al., 1995; CHI et al., 1989; CRIPPEN; EARL, 2007]. Porém, eles também foram apresentados através de vídeos e mais recentemente este formato de apresentação vem ganhando popularidade [CASPERSEN, 2007; JAMES et al., 2013; MARGULIEUX et al., 2012].

Embora utilizar exemplos trabalhados nos momentos iniciais de aquisição de uma habilidade cognitiva tenha inúmeras vantagens, eles são melhor assimilados quando são enriquecidos com mecanismos que orientem o seu processamento. Veremos mais adiante na Seção 3.3 que promover as auto-explicações dos estudantes enquanto eles estudam os exemplos trabalhados, por exemplo, permite que eles engajem-se mais no aprendizado destes exemplos trabalhados e os aprendam de maneira mais aprofundada [CHI et al., 1989; CHI et al., 1994].

3.3 Auto-explicação

Neste trabalho, o FSI foi utilizado para estruturar um conjunto de exemplos trabalhados gravados em vídeo pelo professor que servirão como material instrucional para os estudantes iniciantes em programação. Apesar do FSI fornecer orientação na

estruturação destes exemplos em vídeo, ele não oferece nenhum tipo de orientação na maneira que os estudantes podem estudar e aprender o processo de programação a partir deles. Como estes estudantes de programação, na sua maioria, parecem não estudar adequadamente para uma disciplina de programação, eles podem assistir esses exemplos em vídeo de uma maneira passiva ou processá-los superficialmente. Este tipo de estudo não leva ao aprendizado e, em consequência, os estudantes apresentam resultados insatisfatórios em suas disciplinas de programação.

Em virtude disso, propomos que a orientação na maneira como os estudantes irão estudar estes exemplos trabalhados aconteça através do uso das auto-explicações. Clark et al. (2005) definiram a auto-explicação como *“o diálogo mental que os aprendizes tem ao estudar um exemplo trabalhado que ajuda-os a entender o exemplo e a construir um esquema a partir dele.”* De acordo com Chiu e Chi (2014), a auto-explicação é uma atividade construtiva que promove o aprendizado a partir da elaboração da informação sendo estudada. Ademais, a auto-explicação promove a associação desta nova informação ao conhecimento prévio dos estudantes e também permite que eles construam novo conhecimento a partir da conexão de diferentes pedaços da informação fornecida.

O artigo apresentado por Chi et al. (1989) examinou como estudantes universitários sem experiência no domínio estudado aprendiam habilidades procedurais através da auto-explicação de exemplos trabalhados sobre o conteúdo de mecânica. Na pesquisa, eles coletaram os protocolos verbais dos estudantes enquanto eles estudavam um conjunto de exemplos trabalhados. Estes protocolos verbais foram depois analisados e comparados aos resultados de uma fase posterior de resolução de problemas. Na análise, os pesquisadores encontraram que os estudantes mais bem-sucedidos, aqueles que espontaneamente produziram uma grande quantidade de explicações enquanto estudavam os exemplos e que estudaram os exemplos por um período de tempo maior, obtiveram melhores notas quando comparados aos estudantes malsucedidos na etapa de resolução de problemas. De acordo com os pesquisadores, os estudantes mais bem-sucedidos tiveram melhores notas porque eles aprenderam os exemplos através da explicação e justificativa de cada ação, além de relacionarem essas ações aos princípios e conceitos apresentados no texto expositivo que eles estudaram em conjunto com os exemplos. Além disso, os estudantes bem-sucedidos monitoraram mais precisamente o seu entendimento a respeito do conteúdo enquanto estavam estudando os exemplos. Os

pesquisadores deram o nome de efeito da auto-explicação a este fenômeno dos estudantes que explicaram mais obterem os melhores resultados em fases posteriores de resolução de problemas.

Chi et al. (1994) expandiram o efeito da auto-explicação, demonstrando que estudantes do oitavo ano de uma escola pública nos USA, podiam ser estimulados a auto-explicar quando estudando a partir de um texto expositivo sobre o sistema circulatório. As descobertas destes pesquisadores mostraram que os estudantes que foram encorajados a explicar o texto para eles mesmos tiveram maiores ganhos de aprendizado quando comparados aqueles que não foram estimulados a auto-explicar. Além disso, no grupo que recebeu o estímulo para auto-explicar, os considerados explicadores superiores obtiveram melhores resultados do que os explicadores inferiores, pois produziram na média três vezes mais explicações do que os explicadores inferiores.

As pesquisas realizadas por Pirolli e Recker (1994) e Bielaczyc et al. (1995) foram as primeiras a demonstrar o efeito da auto-explicação no domínio de Programação. Os achados de Pirolli e Recker (1994) sugeriram que os estudantes universitários cujas auto-explicações consistiam principalmente do entendimento do funcionamento dos programas obtiveram melhores resultados em fases subsequentes de resolução de problemas, quando estavam construindo programas. Aliado a isso, Bielaczyc et al. (1995) demonstrou que estudantes universitários sem experiência em programação podiam ser treinados explicitamente a auto-explicar enquanto estavam estudando a partir de textos expositivos que incluíam exemplos. Suas descobertas mostraram que o desempenho dos estudantes treinados a auto-explicar o material instrucional foi superior ao daqueles que não foram treinados.

Além das pesquisas já citadas, uma outra pesquisa relevante e que também analisa os efeitos da auto-explicação no aprendizado a partir de exemplos trabalhados é a apresentada por Renkl (1997) no domínio de Probabilidade. Este pesquisador verificou que, nas pesquisas anteriores, os estudantes mais bem-sucedidos também eram aqueles que passavam mais tempo estudando. Segundo ele, era incerto se os ganhos de aprendizado destes estudantes estavam relacionados ao tempo que eles passavam estudando ou à qualidade das explicações que geravam. Sendo assim, ele resolveu investigar como os estudantes universitários sem experiência em Probabilidade se comportavam ao estudar os exemplos trabalhados por um tempo controlado. Em seus

resultados, Renkl (1997) conseguiu replicar o efeito da auto-explicação mesmo com a restrição de tempo de estudo dos exemplos trabalhados apresentados. Ademais, os seus resultados mostraram que houve dois tipos de estudantes bem-sucedidos, aqueles que produziram mais explicações que relacionavam os exemplos trabalhados aos princípios que os regiam e aqueles denominados raciocinadores antecipados. Este último tipo de estudante antecipava os passos subsequentes da resolução do exemplo trabalhado e comparava a resposta que ele obtinha com a apresentada no material instrucional estudado.

Após estes primeiros estudos demonstrando os benefícios da auto-explicação como uma atividade de aprendizagem, vários outros trabalhos replicaram os seus efeitos. Aleven e Koedinger (2002) investigaram se a atividade de auto-explicação poderia ser apoiada efetivamente através da utilização de um tutor cognitivo no domínio da Geometria. Este tutor solicitava que os estudantes justificassem os passos das soluções dos problemas que eles estavam resolvendo enquanto estudavam para a disciplina. Os resultados encontrados na realização de dois experimentos mostraram que os estudantes que foram estimulados a justificar seus passos de solução durante o momento de resolução de problemas tiveram um melhor resultado do que os estudantes que não explicaram os seus passos de solução dos problemas. Na mesma linha de investigação dos tutores inteligentes, Conati e VanLehn (1999) desenvolveram o *SE-Coach*, um módulo tutor para orientar os estudantes em seus momentos de estudo de exemplos trabalhados através da auto-explicação no domínio da Física. Os resultados encontrados por estes pesquisadores em um experimento mostraram que o desempenho dos estudantes que utilizaram o *SE-Coach* foi ligeiramente superior ao dos estudantes que não utilizaram o módulo tutor, embora a diferença não tenha sido estatisticamente significativa.

McNamara (2004) desenvolveu SERT, *Self-explanation Reading Training*, um treinamento que combina auto-explicação com outras estratégias de leitura, tais como parafrasear ou antecipar o que vai ser dito a seguir no texto, para leitura de textos considerados difíceis no domínio da Biologia. Para testar a eficácia de seu treinamento, a pesquisadora experimentou a sua abordagem com estudantes universitários nos cursos de Psicologia e Biologia. Os estudantes submetidos ao treinamento foram comparados a estudantes que apenas leram alto os mesmos textos de Biologia. Os resultados mostraram que o treinamento auxiliou aqueles estudantes que foram treinados e que

possuíam baixo conhecimento prévio a responder satisfatoriamente questões cujas respostas poderiam ser inferidas dos textos estudados. Ao mesmo tempo em que desenvolveu SERT, McNamara, Levinstein, Boonthum (2004) propuseram iSTART, *Interactive Strategy Training for Active Reading and Thinking*, um sistema de treinamento automatizado utilizando agentes pedagógicos que possibilitou que SERT, antes aplicado individualmente, pudesse ser utilizado simultaneamente por um grupo grande de estudantes.

Wong et al. (2002) realizaram um experimento com estudantes do nono ano em uma escola na Austrália. Eles compararam estudantes que foram treinados individualmente a auto-explicar com estudantes que usaram seus próprios métodos de estudo enquanto aprendiam um novo teorema da Geometria. O material instrucional utilizado consistiu em um folheto textual produzido a partir do livro texto utilizado pelos estudantes em sala de aula. Os resultados encontrados pelos pesquisadores mostraram que os estudantes treinados a auto-explicar obtiveram melhores resultados no pós-teste realizado ao final do experimento, principalmente nas questões consideradas mais difíceis.

Berthold e Renkl (2005) observaram que estudantes que aprenderam a partir de exemplos trabalhados que continham uma combinação de representações múltiplas (imagens e texto) e questões que estimulavam a auto-explicação obtiveram melhores resultados de aprendizagem no domínio de Probabilidade quando comparados aqueles que não tiveram as questões. Crippen e Earl (2007) mostraram que os estudantes que estudaram Química a partir de exemplos estáticos disponíveis online combinados com prompts para auto-explicação melhoraram sua performance, habilidades de resolução de problemas e auto-eficácia. Johnson e Mayer (2010) expandiram o efeito da auto-explicação a um jogo de computador sobre circuitos eletrônicos quando os estudantes selecionavam a resposta de questões de múltipla escolha.

Wylie, Koedinger e Mitamura (2010) apresentaram um dos primeiros trabalhos que investigou a aplicação da auto-explicação para o aprendizado de Inglês como língua estrangeira. Os pesquisadores desenvolveram duas versões de sistemas tutores para ensinar sobre a escolha do artigo na língua inglesa. Na primeira versão do sistema, os estudantes escolhiam o artigo que poderia ser utilizado corretamente. Na outra versão, eles escolhiam uma justificativa para a escolha do artigo que estava sendo utilizado. Em ambos os grupos, os estudantes foram bem sucedidos no que diz respeito ao ganho de

aprendizagem declarativo e procedural, uma vez que conseguiram identificar corretamente os artigos a ser utilizados e as explicações para escolha destes artigos.

Embora, de maneira geral, os estudantes que são treinados ou estimulados a auto-explicar o material instrucional estudado apresentem melhores resultados do que aqueles que não fazem uso da auto-explicação, nem sempre os estudantes geram explicações ou, ainda, geram explicações corretas. Para minimizar esta restrição, Renkl (2002) resolveu adicionar aos exemplos trabalhados, explicações instrucionais que auxiliassem os estudantes em seus momentos de estudo através da auto-explicação. Para avaliar a efetividade da inclusão das explicações instrucionais aos exemplos trabalhados no domínio da Probabilidade, o pesquisador realizou um experimento com estudantes da área de Educação. Os seus resultados mostraram que as explicações instrucionais foram utilizadas de maneira proveitosa somente pelos estudantes que tinham pouco conhecimento prévio no assunto abordado.

Uma pesquisa que também investigou explicações instrucionais juntamente com auto-explicações foi a realizado por Rittle-Johnson (2006). Esta pesquisadora executou um experimento com estudantes do terceiro ao quinto ano de uma escola nos EUA. Os estudantes estavam aprendendo sobre a relação de igualdade através de exemplos trabalhados, no domínio da Matemática. Eles foram divididos em quatro diferentes grupos, organizados em uma combinação 2 X 2 entre as condições (i) com ou sem explicações instrucionais e (ii) com ou sem estímulo a auto-explicação. Os resultados mostraram que os estudantes que receberam explicações instrucionais e foram estimulados a auto-explicar obtiveram os resultados mais elevados nos problemas semelhantes aos exemplos estudados. Além disso, aqueles estudantes que foram estimulados a auto-explicar os exemplos trabalhados, independente de ter recebido ou não as explicações instrucionais, obtiveram os resultados mais elevados nos problemas que abordavam o assunto de maneira diversa aos exemplos aprendidos pelos estudantes. A conclusão a qual chega a pesquisadora é que as explicações instrucionais são efetivas para ensinar a execução de procedimentos corretos aos estudantes (no caso do estudo, como resolver problemas que são semelhantes aos exemplos estudados). No entanto, a auto-explicação pode ser o mecanismo necessário para fazer com que os estudantes solucionem problemas diversos dos que foram estudados por eles.

A partir dos experimentos citados anteriormente, temos uma ideia da importância da atividade da auto-explicação como atividade de aprendizado de diferentes

disciplinas, tais como Matemática, Física, Química, Programação, Biologia e Inglês, para estudantes de diferentes faixas etárias e diversos níveis de educação. O motivo para os bons resultados obtidos a partir da auto-explicação reside no fato de ela ser uma atividade construtiva que exige que os estudantes envolvam-se ativamente em seus processos de aprendizagem [CHIU; CHI, 2014]. Quando os estudantes participam ativamente durante os seus momentos de estudo eles obtêm melhores resultados do que quando estão participando passivamente apenas [BRANSFORD et al., 2000]. A atividade de explicar para si mesmo possui dois mecanismos que justificam os seus benefícios [CHI, 2000]. O primeiro é que as auto-explicações auxiliam os estudantes a identificar lacunas no seu entendimento e tendem a completá-las com as informações que não estão presentes no material instrucional sendo estudado. O segundo é que as auto-explicações ajudam os alunos a revisarem o seu conhecimento, reparando os seus modelos mentais a respeito do assunto sendo estudado. Por seus comprovados benefícios, a auto-explicação foi recomendada pelo Departamento de Educação dos EUA como umas das sete práticas para melhoria dos processos de ensino e de aprendizagem dos estudantes neste país [PASHLER et al., 2007].

3.4 Aprendizagem cognitiva

O conceito de *cognitive apprenticeship* [BROWN; COLLINS; NEWMAN, 1989; COLLINS; BROWN; HOLUM, 1991], ou aprendizagem cognitiva, encontra-se intimamente relacionado aos conceitos vistos anteriormente do FSI, dos exemplos trabalhados e da auto-explicação. Entre outras coisas, os proponentes da aprendizagem cognitiva acreditam que, para promover o aprendizado, é necessário entender a prática dos mais experientes na área de estudo, tornando-a visível aos estudantes. Desta maneira, o raciocínio do professor, ou seja, o conhecimento tácito ao desenvolver a sua atividade, torna-se visível aos estudantes. Semelhantemente, os raciocínios dos estudantes também devem se tornar visíveis para o professor. Tornar o raciocínio visível significa que os processos cognitivos e metacognitivos envolvidos na execução de uma atividade devem se tornar explícitos durante a aprendizagem dessa atividade. Consequentemente, os estudantes podem observar, agir e praticar a atividade a partir do que for observado e também contando com a ajuda dos professores. Os professores, por sua vez, podem analisar a evolução de seus estudantes e tomar decisões de acordo com a sua percepção a respeito do processo de aprendizado deles.

De acordo com Brown et al. (1989) e Collins et al. (1991), o *framework* para ambientes de aprendizado definido pela aprendizagem cognitiva consiste, entre outras dimensões, de seis métodos de ensino: modelagem, treinamento, *scaffolding*, articulação, reflexão e exploração. Estes métodos devem ser utilizados pelos estudantes para observar, engajar-se e inventar ou descobrir as estratégias dos mais experientes enquanto estes últimos desempenham a sua atividade. Desta maneira, os estudantes poderão perceber como as estratégias dos mais experientes podem ser combinadas ao conhecimento que estes estudantes possuem e como eles podem adaptar a sua própria prática de acordo com o que foi percebido das técnicas empregadas pelos mais experientes.

Sendo assim, estes métodos estão organizados em três grupos conforme discutido a seguir. O primeiro grupo é formado pelos métodos de ensino modelagem, treinamento e *scaffolding*. Os métodos neste grupo foram desenvolvidos para auxiliar os alunos a integrar o conjunto de habilidades cognitivas e metacognitivas através de processos de observação e da prática orientada. A modelagem envolve a realização de uma atividade por uma pessoa mais experiente e a observação e subsequente construção dos modelos conceituais do processo necessário para a execução de tal atividade pelos estudantes. Para tanto, os processos cognitivos e metacognitivos internos dos mais experientes necessitam ser externalizados para que os estudantes possam verificar como eles aplicam os seus conhecimentos declarativos e procedimentais.

O treinamento consiste na observação do estudante e no oferecimento de dicas, orientação, *feedback*, modelagem, lembretes e novas tarefas que os auxiliarão a aproximar a atividade realizada por eles da atividade realizada pelos mais experientes. O treinamento serve para direcionar o estudante para aspectos que ainda não foram percebidos por ele ou simplesmente lembrá-lo de algum aspecto da atividade que não está recebendo a devida atenção. A interação decorrente do treinamento relacionada a aspectos específicos da atividade que surgem enquanto o estudante está tentando executar a atividade.

O *scaffolding*, por sua vez, refere-se ao apoio que o professor fornece para ajudar o estudante a executar uma atividade, tais como sugestões ou questionamentos realizados pelo professor. Ele também pode fornecer algo físico como fichas impressas contendo modelos de sentenças que podem ser utilizados pelos estudantes enquanto escrevem textos, por exemplo. Quando o auxílio é fornecido pelo professor, este pode

executar partes da atividade que o aluno ainda não é capaz de executar. Um ponto chave do *scaffolding* que é fornecido pelo professor é que, à medida que o aluno vai conseguindo realizar as suas atividades sozinho, este auxílio vai sendo gradualmente reduzido até que o estudante possa executar a atividade sem nenhum tipo de ajuda. A esta situação dá-se o nome de *fading*.

O segundo grupo é formado pelos métodos de ensino articulação e reflexão. Estes métodos foram desenvolvidos para auxiliar o estudante nas observações da resolução dos problemas pelos mais experientes e a ganhar consciência e controle sobre o seu próprio processo de resolução de problemas. O método da articulação envolve estratégias que levam o estudante a articular seu conhecimento, seu raciocínio e seus processos de resolução de problemas. Este método pode ser utilizado de várias maneiras diferentes, através do questionamento dos estudantes pelo professor, do encorajamento da organização de ideias dos estudantes ao resolver problemas ou a exposição destas ideias a outros estudantes. O método da reflexão, por sua vez, envolve o estabelecimento de estratégias para que os estudantes possam refletir os seus próprios processos de resolução de problemas comparando-os aos do professor, aos de outro estudante ou a um modelo cognitivo interno de experiência. Este método de reflexão pode ser aprofundado se os processos de resolução de problemas do estudante e do professor puderem ser comparados repetidamente.

O último grupo consiste apenas no método de exploração, que objetiva encorajar a autonomia do estudante em executar os processos de resolução de problemas mais complexos e também a definir e a formular problemas por si próprio. O método da exploração envolve encorajar o estudante a resolver os problemas por si próprio. Não somente isto, mais também a estabelecer metas e sub-metas para que este estudante consiga enriquecer o seu conhecimento. A exploração independente é conseguida quando o suporte fornecido pelo método do *scaffolding* é completamente retirado, podendo ser utilizado quando o estudante é mais experiente e pode passar a resolver problemas sozinho. Como estamos lidando apenas com estudantes no início da aquisição da habilidade cognitiva, a execução deste método está fora do escopo do trabalho aqui apresentado.

A aprendizagem cognitiva já foi utilizada com sucesso em diferentes disciplinas. Brown et al. (1989) e Collins et al. (1991), por exemplo, apresentam três exemplos onde essa abordagem foi utilizada. O primeiro método citado foi o método *reciprocal*

teaching por Palincsar e Brown (1984) para atividades de leitura. Para atividades de escrita foi citado o método *procedural facilitation of writing* por Scardamalia e Bereiter (1987). O último citado foi o método para resolução de problemas na matemática proposto por Schoenfeld (1983; 1985).

3.5 Trabalhos relacionados

Nesta seção iremos apresentar trabalhos relacionados à abordagem apresentada nesta tese.

3.5.1 Utilizando o *framework stepwise improvement*

O FSI foi empregado como referencial teórico para a construção de um processo para o aprendizado de programação orientada a objetos em Java para iniciantes chamado STREAM [CASPERSEN; KÖLLING, 2009], um acrônimo para os seis passos que fazem parte do processo *stubs*, testes, representação, avaliação, atributos e métodos. O objetivo do primeiro passo é fazer com que os estudantes criem um esqueleto de cada classe contendo os *stubs* dos métodos. Sendo assim, escreve-se somente o cabeçalho dos métodos com o corpo vazio, para aqueles métodos que não retornam nada, ou o corpo com uma única linha que retorna o valor padrão do tipo de retorno do método, para os métodos que retornam algum valor. O objetivo do segundo passo é garantir que há testes disponíveis para cada um dos métodos das classes criadas. Assim, nesta fase os estudantes devem criar os testes que serão utilizados para testar a funcionalidade dos métodos das classes. O objetivo do terceiro passo é garantir que pelo menos duas representações alternativas sejam consideradas para cada uma das classes. Em seguida, no quarto passo, os estudantes devem avaliar cada uma das representações para as classes pensadas anteriormente. Esta avaliação é normalmente realizada em função do nível de dificuldade⁹ para implementar cada um dos métodos definidos para a classe. A avaliação pode ser realizada pelos estudantes individualmente, em grupo ou com a ajuda do professor. O objetivo do quinto passo é definir as variáveis de instância das classes e os valores apropriados para a inicialização de cada uma delas. Por fim, o objetivo do sexto passo é implementar os métodos das classes, seguindo as atividades definidas pelo FSI. De acordo com os autores, a ordem de implementação dos métodos

⁹ Os autores propõem a utilização de uma escala de 5 níveis: trivial, fácil, mediana, desafiadora, difícil [CASPERSEN; KÖLLING, 2009].

é arbitrária, contudo eles recomendam que os métodos sejam implementados dos mais fáceis para os mais difíceis.

Veremos mais adiante nos Capítulos 4 e 5 que a abordagem para o ensino de programação introdutória proposta neste trabalho, assim como o STREAM, também utilizará o processo de programação proposto pelo FSI como parte do seu referencial teórico. A diferença proposta com a nossa abordagem se dará através da inclusão do métodos de ensino para a reflexão fornecidos pela combinação do processo de programação representado pelos exemplos em vídeo com perguntas para a auto-explicação como forma de estímulo aos estudantes no processo de aprendizado decorrente do estudo destes exemplos.

3.5.2 Utilizando vídeos para apoiar o ensino e a aprendizagem de programação

Atualmente, a utilização de vídeos nos processos de ensino e de aprendizagem de programação vem se tornando cada vez mais popular. Por exemplo, inúmeros vídeos estão disponíveis em canais no Youtube ou em organizações tais como Khan academy¹⁰ e Code.org¹¹ e MOOCs tais como Coursera¹², EdX¹³, Udacity¹⁴, entre tantos outros. Há também iniciativas de professores que estão procurando criar materiais complementares às suas aulas presenciais [CARLISLE, 2010; RAABE; BERNARDES; JUNIOR, 2014] ou como parte do material de uma disciplina ministrada na EaD [AURELIANO; TEDESCO, 2013; DOTTA et al., 2013; VILNER; ZUR; SAGI, 2012]. Contudo, a utilização de vídeos para o ensino de programação não é uma novidade. Alguns anos atrás, Benedsen e Caspersen (2008) mostraram os benefícios de utilizar vídeos como uma ferramenta para apresentar o processo de programação para estudantes. Ao invés de livros que são estáticos, os autores defendem que os vídeos são instrumentos importantes para exibir o processo de programação em razão de serem uma mídia essencialmente dinâmica. Além disso, os vídeos podem ser manipulados facilmente pelos estudantes, que podem reproduzir, avançar, retroceder ou pausar no seu próprio ritmo, quantas vezes forem necessárias e onde eles estiverem.

¹⁰Khan Academy. Disponível em: <www.khanacademy.org/> Acesso em: 26 jan. 16.

¹¹code.org. Disponível em: <code.org>, Acesso em: 26 jan. 16.

¹²Coursera. Disponível em: <www.coursera.org>. Acesso em: 26 jan. 16.

¹³edX. Disponível em: <www.edx.org>. Acesso em: 26 jan. 16.

¹⁴Udacity. Disponível em: www.udacity.com. Acesso em: 26 jan. 16.

No Khan Academy, os vídeos apresentam conhecimento declarativo enquanto vão apresentando também exemplos de construção do código, mas não há questões para que os estudantes auto-expliquem os programas que estão sendo construídos. No Code.org, os vídeos apresentam apenas conhecimento declarativo, mas não envolvem vídeos para apresentação da construção dos programas. Nos MOOCs, normalmente, os vídeos são vídeo-aulas que mesclam a apresentação de conhecimento declarativo com a construção de código, nos moldes de uma aula expositiva em sala de aula tradicional. Por exemplo, a disciplina “Fundamentos de programação com Python” ministrada pelo Professor Kunal Chawla do Udacity, contém vídeo-aulas que utilizam diversos recursos interativos, tais como botões de opção e de seleção e caixas de texto, e outros recursos gráficos sofisticados que parecem não estar disponíveis para professores que trabalham fora da organização. Outro exemplo é a disciplina “Programação com o Scratch” ministrada pela Professora Colleen Lewis da Harvey Mudd College através da plataforma EdX. Além do conhecimento declarativo, os vídeos nesta disciplina também apresentam a construção do código, mas não há questões para que os estudantes auto-expliquem o processo de construção dos programas que estão sendo construídos. Exercícios relacionados ao conteúdo são apresentados somente após a exibição do vídeo. O último exemplo é a disciplina “Programação para todos” ministrada pelo Professor Charles Severance da Universidade de Michigan através da plataforma Coursera. As vídeo-aulas, neste caso, são as mais semelhantes a aulas expositivas tradicionais, incluindo a exibição de slides. Há também vídeos contendo a resolução de exercícios, mas não há questões para que os estudantes auto-expliquem o processo de construção dos programas que estão sendo construídos. Exercícios relacionados ao conteúdo são apresentados após a exibição do vídeo.

Similarmente, as vídeo-aulas produzidas por Dotta et al. (2014), Raabe, Bernardes e Júnior (2013), Aureliano e Tedesco (2013), Vilner, Zur e Sage (2012) e Carlisle (2010) apresentam conhecimento declarativo juntamente com a apresentação de alguns exemplos de construção de código. Contudo, além de não seguirem as ideias propostas pelo FSI não realizam nenhum tipo de estímulo a auto-explicação durante o processo de construção dos programas.

3.5.3 Utilizando exemplos trabalhados no ensino e na aprendizagem de programação

Os exemplos trabalhados também já foram empregados anteriormente para o ensino-aprendizagem de programação. Os artigos que vamos apresentar nesta seção serão divididos em dois grupos [SKUDDER; LUXTON-REILLY, 2014]. O primeiro engloba aqueles que fizeram pesquisa com os exemplos trabalhados na área de programação, verificando como os estudantes compreendem tais exemplos ou como eles podem ser apresentados de maneira que o aprendizado dos estudantes seja mais eficiente. O segundo engloba aqueles que propõem ou relatam a experiência de utilização dos exemplos trabalhados em algum curso de programação como parte do material instrucional.

Na área de pesquisa, os primeiros trabalhos que apresentaram a utilização de exemplos trabalhados estáticos na área de programação como parte de seu material instrucional foram Pirolli e Recker (1994) e Bielaczyc, Pirolli e Brown (1995), conforme vimos anteriormente. Outro trabalho dentro do grupo da pesquisa foi apresentado por Margulieux, Guzdial e Catrambone (2012) que utilizaram exemplos trabalhados em vídeo rotulados para o aprendizado de programação através da ferramenta Android App Inventor. Os sujeitos desta pesquisa foram estudantes universitários com pouca ou nenhuma experiência em programação ou ciência da computação. Estes sujeitos foram divididos em dois grupos, o grupo instrucional que estudou os exemplos trabalhados rotulados e o grupo controle que estudou os mesmos exemplos trabalhados não rotulados. Os resultados encontrados pelos pesquisadores mostraram que os rótulos auxiliaram os estudantes no aprendizado dos objetivos que eles precisavam alcançar, reduzindo a carga cognitiva imposta aos iniciantes em programação. Além disso, os rótulos também auxiliaram os estudantes no desenvolvimento dos modelos mentais necessários para o aprendizado de programação na ferramenta utilizada.

Ainda dentro do grupo dos trabalhos relacionados à pesquisa, destacamos o trabalho realizado por Abdul-Rahman e du Boulay (2014). Estes pesquisadores resolveram estudar a maneira como os estudantes de diferentes estilos de aprendizagem aprendem programação a partir dos exemplos trabalhados. Os sujeitos da pesquisa foram estudantes universitários matriculados em uma disciplina introdutória de programação divididos em três grupos diferentes. No primeiro grupo, eles estudavam os

exemplos estáticos através da explicação de um determinado trecho de código; no segundo, eles estudavam os exemplos completando o código; e no terceiro, eles estudavam os exemplos através da combinação das duas abordagens anteriores. Entre outros resultados, estes pesquisadores revelaram não ter encontrado diferenças significativas entre o desempenho dos estudantes ativos e reflexivos nos diferentes tratamentos adotados no estudo.

Mais recentemente, Morrison, Margulieux e Guzdial (2015) apresentaram mais um trabalho com exemplos trabalhados rotulados para o aprendizado de programação de estudantes matriculados em diferentes cursos em uma universidade técnica. Neste trabalho, os sujeitos receberam três diferentes tratamentos, exemplos trabalhados não rotulados, exemplos trabalhados onde os rótulos foram fornecidos e exemplos trabalhados onde os rótulos eram gerados pelos estudantes. Entre outros resultados, eles encontraram que rotular os exemplos trabalhados melhora o desempenho dos estudantes. Além disso, fazer com que os estudantes gerem os rótulos nos exemplos trabalhados leva mais tempo e gera os mesmos resultados de fornecer os exemplos trabalhados já rotulados aos estudantes.

No grupo dos que somente utilizaram os exemplos trabalhados como parte do material instrucional em uma disciplina de programação, um dos primeiros trabalhos que temos é o apresentado por Caspersen e Bennedesen (2007). Eles utilizaram exemplos trabalhados como material de apoio em um curso introdutório de programação em Java. Os exemplos trabalhados ilustram o processo de programação e ajudam a revelar o conhecimento tácito e os processos implícitos envolvidos no desenvolvimento de programas. Combinados com exemplos apresentados em sala de aula e um conjunto de exercícios, estes exemplos podiam ser utilizados tanto dentro quanto fora de sala de aula, como forma de reforçar o tópico sendo aprendido.

Gray et al. (2007) apresentam uma proposta teórica da aplicação de exemplos trabalhados estáticos graduais para o ensino de uma disciplina introdutória de programação utilizando uma linguagem imperativa. A proposta sugere que os exemplos trabalhados vão sendo construído gradualmente através de diferentes atividades relacionadas a programação, tal como projeto, implementação e execução do programa. A proposta é realizada através da apresentação de um exemplo para cada um dos seguintes tópicos: estrutura condicional, estrutura de repetição e chamada de subrotina.

Lui, Cheung e Li (2008) também utilizaram exemplos trabalhados estáticos com estudantes universitários matriculados em um curso introdutório de programação. Neste curso, nas aulas de laboratório, os estudantes eram encorajados a compartilhar as soluções que eles próprios propunham para os problemas assim como a estudar as soluções propostas por seus colegas, como uma forma de instrução por pares. Embora menos da metade dos estudantes tenha compartilhado pelo menos uma solução em um período de 7 semanas com os seus colegas, aqueles que compartilharam acharam importante estudar as soluções alternativas às quais tiveram acesso. Além disso, os resultados mostraram que aqueles estudantes que tiveram os melhores resultados foram os que mais compartilharam programas com os seus colegas.

Sorva e Seppälä (2014) utilizaram exemplos trabalhados estáticos como parte do material instrucional em um dos *frameworks* propostos para a disciplina introdutória de programação que ministraram. Com base neste *framework*, os tópicos da disciplina foram introduzidos através de um projeto abrangente, mas dividido em pequenas partes que foram integradas ao final do projeto. O objetivo com a utilização do *framework* proposto foi prevenir a sobrecarga cognitiva, corrigir problemas de entendimento e promover o aprendizado efetivo através da prática estimulada pela construção do projeto.

A grande maioria dos exemplos trabalhados analisados neste Capítulo são apresentados textual e estaticamente aos estudantes. Os exemplos trabalhados utilizados na abordagem definida nesta tese são apresentados em vídeo. Além disso, adicionamos a estes exemplos trabalhados em vídeo, mecanismos para que os estudantes auto-expliquem o processo de programação apresentado nos exemplos trabalhados em vídeo.

3.5.4 Utilizando a auto-explicação no ensino e na aprendizagem de programação

A técnica da auto-explicação também foi empregada anteriormente em trabalhos na área de ensino-aprendizagem de programação. Na nossa pesquisa, os primeiros trabalhos que relatam o estudo da técnica da auto-explicação em programação são as pesquisas de Pirolli e Recker (1994) e Bielaczyc, Pirolli e Brown (1995) mencionadas anteriormente. Nos últimos 10 anos, alguns estudos voltaram a chamar a atenção da comunidade para a auto-explicação na área de programação. Tobar, Oliveira e Freitas (2006) implementaram uma ferramenta de apoio ao entendimento de exemplos apresentados estaticamente aos estudantes e que também oferece dicas a eles em caso de

erros nas explicações. O sistema não foi avaliado com estudantes, mas foi avaliado através de um conjunto de explicações testes. Os resultados mostraram que o sistema funciona com alguns defeitos, mas possui bom tempo de resposta e precisão de 70% no que diz respeito ao resultado esperado.

Kwon, Kumalasari e Howland (2011) combinaram a auto-explicação com a detecção e correção de erros em HTML. Os sujeitos do estudo foram estudantes universitários participando de um curso de “Introdução a desenvolvimento web”. Estes estudantes foram divididos em dois grupos. O primeiro formado por aqueles que após detectar e corrigir o erro proposto eram estimulados a responder perguntas de auto-explicação sobre o significado das mensagens de erro e o motivo do erro. O segundo grupo foi formado por estudantes que completavam as mesmas explicações fornecidas pelo professor a partir de palavras-chave presentes em um menu. Após fornecerem as explicações, os dois grupos de estudantes eram levados a corrigir o erro exibido. Os resultados mostraram que os estudantes do primeiro grupo tiveram um melhor desempenho do que os do segundo grupo. Além disso, aqueles estudantes que geraram mais explicações corretas no primeiro grupo foram o que resolveram os problemas no pós-teste de maneira mais precisa.

Yen-Chu (2012) investigou a combinação de auto-explicação com diagramas de arquitetura de computadores com o objetivo de aprender a linguagem de programação Assembly. Os sujeitos desta pesquisa foram estudantes universitários de um departamento de Engenharia da Computação. Estes estudantes foram divididos em dois grupos, instrucional e controle. O grupo instrucional contou com aulas adicionais de estudo onde eles tinham a oportunidade de aprender como descrever o fluxo de execução dos programas escritos em Assembly nos diagramas de arquitetura. O grupo controle não participou destas aulas adicionais. Os resultados mostraram que os estudantes do grupo instrucional tiveram melhores resultados no pós-teste do que aqueles do grupo controle.

Lee, Ko, Kwan (2013) investigaram os efeitos da inclusão de níveis de avaliação a um jogo educacional que ensina programação através da retirada de erros de programas escritos em uma linguagem de programação similar a Python. Estes níveis de avaliação continham questões de múltipla escolha e prompts de auto-explicação que deveriam ser respondidos a medida que eles iam passando de nível no jogo. Os sujeitos que participaram desta pesquisa não tinham nenhum tipo de experiência em disciplinas de

programação ou em escrever qualquer tipo de código. Os resultados encontrados sugeriram que os estudantes no grupo instrucional completaram mais níveis no decorrer do jogo e passaram mais tempo jogando o jogo, o que sugere um maior engajamento dos estudantes. Além disso, os resultados também mostraram que os estudantes no grupo instrucional foram mais rápidos ao completar os níveis do jogo do que os do grupo controle.

Gross et al. (2013) apresentaram uma proposta de modelo de um sistema tutor inteligente que estimula o processo de auto-explicação de estudantes quando eles cometem erros sintáticos em programas escritos em Java. Quando um erro sintático ocorre, o tutor inicia um diálogo com o estudante para detectar o tipo de erro cometido por ele, além de sugerir material de leitura e dicas para que o estudante corrija os erros. O sistema proposto ainda está em fase de modelagem, contudo os seus autores acreditam que utilizar um tutor que dialoga com os estudantes a partir dos erros cometidos por ele, encoraja o estudante a refletir sobre o seu conhecimento através de um processo de auto-explicação e os estudantes conseguirão resolver os problemas sintáticos gerados.

Kumar (2014) avaliou os efeitos da auto-explicação associada a um sistema tutor inteligente enquanto os estudantes executavam códigos sobre estruturas de seleção. Contudo, as explicações só eram estimuladas pelo tutor se os estudantes cometessem algum erro enquanto estavam tentando executar o código. Os resultados mostraram que os estudantes do grupo controle praticaram menos atividades e, por esta razão, tiveram um resultado no pós-teste abaixo do resultado obtido pelo grupo controle. Segundo o autor deste estudo, a falta do efeito da auto-explicação se deveu às condições do estudo realizado, já que a ferramenta foi disponibilizada online e foi utilizada arbitrariamente pelos sujeitos do estudo, sem a supervisão de um professor.

Recentemente, Vihavainen, Miller e Settle (2015) investigaram a combinação de questões de auto-explicação em listas de exercícios de programação. Os sujeitos investigados foram estudantes universitários que estavam cursando um curso introdutório de programação em Java ministrado durante as férias. Os seus resultados demonstraram que os estudantes do grupo instrucional, que receberam o material instrucional com as perguntas para auto-explicar as respostas fornecidas, tiveram um melhor desempenho do que aqueles do grupo controle, que não tinham no seu material tais perguntas.

Embora a técnica de auto-explicação tenha sido pouco explorada, encontramos alguns exemplos da sua utilização na área de programação. No entanto, nenhum destes exemplos combinou o processo de programação previsto pelo FSI com a auto-explicação. Veremos mais adiante nos Capítulos 4 e 5 como será implementada a abordagem proposta nesta tese que preenche esta lacuna.

3.5.5 Utilizando a aprendizagem cognitiva no ensino e na aprendizagem de programação

A aprendizagem cognitiva também já foi empregada para o ensino de programação. Uma ferramenta chamada SmallTALKER que suporta o aprendizado de programação através da linguagem Smalltalk foi proposta por Chee (1995). Através desta ferramenta, Chee apoia os seis métodos de ensino que a teoria da aprendizagem cognitiva define. Para a modelagem, a ferramenta SmallTALKER apresenta um profissional mais experiente no assunto apresentando conceitos da linguagem Smalltalk aos estudantes através de vídeos gravados. Para promover o treinamento, a ferramenta SmallTALKER oferece um recurso de *feedback* às ações e aos erros do estudante enquanto ele está programando através de mensagens associadas às ações executadas ou aos erros cometidos por este estudante logo depois que eles acontecem. Para promover o *scaffolding*, além dos vídeos que podem ser vistos e revistos pelos estudantes sempre que for necessário, a ferramenta possui um sistema de ajuda que pode ser acessado pelos estudantes. Quando os estudantes acessam o sistema, é listado um conjunto de tópicos relacionado ao tipo de atividade que eles estão desempenhando no momento. Segundo Chee (1995), fazer com que os estudantes solicitem ajuda permite que eles tenham uma maior responsabilidade sobre os seus próprios processos de aprendizado, tornando-se estudantes mais ativos.

Para promover a articulação, a SmallTALKER oferece um conjunto de questões sobre o conhecimento declarativo que podem ser respondidas pelos estudantes quando este finaliza uma parte do material de instrução. A SmallTALKER oferece um mecanismo semelhante para a reflexão, contudo esse mecanismo somente é acessado pelo estudante se ele assim quiser. Neste caso, as questões envolvem conceitos declarativos mais profundos sobre programação ou detalhes relacionados a prática de programação. Além disso, para o processo de reflexão, as questões possuem um vídeo contendo a resposta dada por um profissional experiente. Por fim, a exploração é

oferecida aos estudantes através de uma funcionalidade que traz a oportunidade para que eles utilizem outra ferramenta que possui módulos instrucionais não cobertos pela SmallTALKER.

Embora a ferramenta proposta ofereça mecanismos que implementam os métodos de ensino definidos pela Aprendizagem Cognitiva, a utilização de vídeos e o aparecimento de questões na SmallTALKER não acontece durante o estudo do processo de construção de programas realizado pelos estudantes nem enquanto os estudantes estão programando. Ademais, os vídeos são majoritariamente direcionados ao conhecimento declarativo e não procedural, ou seja, não envolve a construção dos programas. Por fim, a construção de uma ferramenta para uma linguagem específica limita a sua utilização a situações em que somente essa linguagem seja ensinada. Veremos mais adiante nos Capítulos 4 e 5 que a abordagem para o ensino de programação introdutória proposta neste trabalho é utilizada durante o processo de programação e também não é dependente de uma ferramenta ou linguagem de programação específica.

Outra abordagem que utiliza a teoria da Aprendizagem Cognitiva no processo de programação para iniciantes foi definida por Caspersen e Bennedsen (2007). Vimos anteriormente na Seção 3.4, que uma das características da aprendizagem cognitiva define que os processos e o raciocínio dos mais experientes durante a execução de uma atividade devem se tornar visíveis para os iniciantes que estão aprendendo aquela atividade. Caspersen e Bennedsen (2007) utilizaram esta ideia para revelar o processo de programação e o raciocínio envolvido neste processo em sessões de construção de exemplos para a apresentação do conteúdo da disciplina. Segundo os autores, embora implementar os exemplos em sala de aula seja importante, construí-los ao vivo nem sempre traz os resultados esperados, uma vez que (i) os estudantes podem não estar atentos o suficiente para acompanhar a implementação no quadro, (ii) o professor pode não apresentar o exemplo em um ritmo adequado para o estudante ou (iii) não repetir o exemplo para o estudante a quantidade de vezes que ele necessite. Sendo assim, nem todos os estudantes acompanham a implementação do programa no mesmo ritmo que é ditado pelo professor. Por isso, além de apresentar os exemplos em sala de aula, Bennedsen e Caspersen (2007) resolveram também gravar a execução destes exemplos em vídeo, para permitir ao aluno a sua utilização também fora de sala de aula. Os exemplos gravados em vídeo são importantes ferramentas no processo de aprendizado

do estudante, já que eles permitem uma exploração mais detalhada por aqueles com menos conhecimento na disciplina, pois eles podem pausar, reprisar, rever parte ou todo o vídeo, quantas vezes forem necessárias para entender completamente o assunto sendo abordado.

Veremos mais adiante nos Capítulos 4 e 5 que a abordagem para o ensino de programação introdutória proposta neste trabalho utilizará uma ideia semelhante para a construção dos exemplos tomando-se por base a Aprendizagem Cognitiva. A diferença proposta com a abordagem se dará através da inclusão do métodos de ensino para a reflexão fornecidos pela combinação dos exemplos em vídeo com perguntas para a auto-explicação que deverão auxiliar os alunos no processo de aprendizado decorrente do estudo destes exemplos.

Mais recentemente, Vihavainen, Paksula e Luukkainen (2011) também apresentaram uma técnica para o ensino-aprendizagem de programação para iniciantes baseada na Aprendizagem Cognitiva, a qual nomearam Aprendizagem extrema. Nesta técnica, eles definiram um conjunto de valores e práticas que enfatizam o aprender fazendo e o *feedback* contínuo como forma de tornar o aprendizado mais eficiente. Para executar o valor do aprender fazendo, os autores sugerem que haja poucas aulas expositivas (2 horas somente por semana ao invés de 5 horas como ocorre normalmente na universidade dos pesquisadores) e muitas aulas para a realização de extensas listas de exercícios (contendo de 15 a 40 problemas cada uma), começando já nas primeiras semanas da disciplina. As aulas de exercícios ocorrem nos laboratórios da universidade onde o método foi implementado e os professores e monitores estão sempre presentes para orientar o processo de resolução dos problemas pelos estudantes. Para suprir as poucas aulas expositivas, o material das aulas é disponibilizado online. Este material instrucional inclui exemplos trabalhados apresentados em formato textual. O feedback contínuo, por sua vez, é fornecido aos estudantes pelos professores e monitores que acompanham o progresso dos estudantes em suas aulas para resolução de exercícios e também através de avaliações rápidas realizadas a cada duas semanas na disciplina, além da avaliação final.

Veremos mais adiante no Capítulo 4 que o trabalho apresentado nesta tese é similar a este na utilização de exemplos trabalhados. Porém, ao invés de utilizarmos os exemplos trabalhados em formato textual, utilizaremos os exemplos trabalhados em formato de vídeo para que apresentar o processo de construção dos programas

dinamicamente. Aliado aos exemplos trabalhados em vídeo, utilizaremos a técnica da auto-explicação como forma de promover o processo de reflexão dos estudantes durante o processo de construção dos programas.

3.6 Implicações

Este terceiro capítulo apresentou o referencial teórico utilizado na presente tese. Vimos que o FSI é um *framework* que descreve a programação como um processo de construção iterativo e incremental que compreende as atividades de extensão, refinamento e reestruturação. A partir da definição deste *framework*, pensamos a ação de programar como uma sequência de passos que são executados pelo programador com a finalidade de construir uma implementação que atenda uma especificação estabelecida a partir do conjunto de requisitos determinado para o programa. Ademais, o FSI fornece orientação nas três atividades que engloba, oferecendo orientação na maneira que materiais instrucionais (e.g., livro-texto, exercícios, aulas e exemplos) podem ser estruturados.

Vimos também que os exemplos trabalhados representam uma sequência de passos que demonstram a realização de uma tarefa em particular ou a solução de um problema específico. Estudar exemplos trabalhados constitui-se em uma atividade de aprendizagem superior a resolução de problemas para os estudantes nas fases iniciais de aquisição de habilidades cognitivas, enquanto os esquemas de resolução de problemas ainda estão sendo adquiridos. Quando os estudantes estão mais experientes e podem resolver os problemas sem maiores dificuldades, eles devem ser incentivados a resolver problemas para aplicar o conhecimento adquirido nos momentos de estudo dos exemplos.

Neste trabalho, o FSI foi utilizado para estruturar um conjunto de exemplos trabalhados gravados em vídeo pelo professor que serviram como parte do material instrucional para os estudantes iniciantes em programação. Apesar do FSI fornecer orientação na estruturação destes exemplos em vídeo, ele não oferece nenhum tipo de orientação na maneira que os alunos podem estudar e aprender o processo de programação a partir deles. Sabemos também que os estudantes podem adotar uma atitude passiva enquanto estudam estes exemplos, sem processá-los de uma maneira aprofundada. Vimos que uma maneira de promover a aprendizagem a partir de exemplos trabalhados é estimular os estudantes a auto-explicarem tais exemplos.

A revisão da literatura apresentada no Capítulo 2 mostrou que o maior obstáculo encontrado pelos iniciantes em programação não está em entender as estruturas de uma linguagem, mas em usá-las e combiná-las adequadamente para construir programas. Para lidar com esse problema, os alunos iniciantes em programação precisam aprender o processo de programação. Este processo de programação é adequadamente definido pelo FSI nas três atividades que engloba. Para aprender esse processo de programação, aos estudantes serão oferecidos exemplos trabalhados que seguem os preceitos da construção de programas segundo o que é definido pelo *framework*. Estes exemplos serão apresentados em vídeos, pois eles são considerados a forma ideal de apresentar o processo de programação aos estudantes. Como estamos focando em estudantes iniciantes em programação, a utilização de exemplos trabalhados se enquadra adequadamente ao início da aquisição da habilidade cognitiva que eles estão adquirindo.

Ademais, utilizaremos dois dos métodos de ensino da aprendizagem cognitiva, o *scaffolding* e a reflexão. Um problema levantado no Capítulo 2 diz respeito à maneira inadequada de estudo por parte dos alunos de programação. Para lidar com esse problema, acreditamos que a abordagem de programação deve promover a auto-explicação destes exemplos em vídeo pelos estudantes para auxiliá-los em seus processos de estudo. Para fazer isso, iremos acrescentar perguntas aos exemplos em vídeo que servirão como mecanismos de *scaffolding* para orientar os estudantes enquanto eles estudam o processo de construção de programas sendo realizado pelo professor. Estas perguntas auxiliarão os estudantes a perceber as estratégias adotadas pelo professor ao construir o programa e a adquirir os esquemas que utilizados em fases subsequentes de resolução de problemas. Além disso, espera-se que as perguntas para auto-explicação ao longo do vídeo façam com que os estudantes adotem uma estratégia de estudo mais profunda, deixando de ser passivos e tornando-se mais ativos ao longo de seus processos de estudo. Estratégias de aprendizagem mais profundas estão associadas ao sucesso dos estudantes em programação.

Além de refletirem a partir das perguntas que os orientam em seus processos de auto-explicação, também veremos mais adiante na versão refinada da abordagem que os estudantes são convidados a refletir sobre as estratégias adotadas pelo professor através da reprodução das atividades sendo executadas nos exemplos. Ademais, a utilização do método da reflexão permite que os estudantes manipulem (i.e., reproduzam, pausem, voltem, avancem) os vídeos para que possam repetir as atividades que estão sendo

realizadas pelo professor, enquanto estudam a construção de programas que é apresentada neles. Essa reprodução auxiliará os estudantes a comparar o seu processo de resolução de problema com o do professor, ajustando-o caso seja necessário.

Em seguida, no Capítulo 4 explicamos como estas teorias foram utilizadas na definição da abordagem proposta na presente tese.

4 Stepwise self-explanation

Este quarto capítulo tem por objetivo apresentar a abordagem de aprendizado de programação proposta nesta tese, a qual nomeamos *Stepwise self-explanation*. Para tanto, primeiro apresentamos as razões que motivaram este trabalho na Seção 4.1. Em seguida, na Seção 4.2 descrevemos a versão piloto da abordagem, juntamente com as suas fases e os conceitos envolvidos. Semelhantemente, na Seção 4.3 descrevemos a versão refinada da abordagem. Por fim, na Seção 4.4 apresentamos as considerações finais do capítulo.

4.1 Motivação

Como mencionado anteriormente programar é uma atividade intrinsecamente difícil e, por esta razão, os estudantes que iniciam suas vidas como programadores passam por muitos obstáculos. Caspersen e Kölling (2009) e Lahtinen et al. (2005) declaram, em seus trabalhos, que os alunos iniciantes em programação não conseguem combinar e utilizar adequadamente os conceitos de programação aprendidos para a construção de um determinado programa. Este problema está intimamente relacionado às deficiências que são inerentes aos estudantes iniciantes, que naturalmente não possuem as mesmas habilidades que já foram adquiridas por aqueles mais experientes [ROBINS et al., 2003; WINSLOW, 1996]. De acordo com Caspersen (2007) e Caspersen e Kölling (2009), uma forma de minimizar as dificuldades que os estudantes iniciantes em programação vivenciam é a apresentação de materiais instrucionais que seguem o FSI.

Neste trabalho, propomos que os estudantes sejam apresentados ao processo de programação pelo estudo de exemplos trabalhados estruturados de acordo com o FSI. Os exemplos são considerados tanto por professores quanto por estudantes o material instrucional mais útil para o ensino e a aprendizagem em programação [LAHTINEN et al., 2005]. Além disso, a utilização de exemplos trabalhados é uma prática eficaz para o aprendizado de iniciantes em diferentes disciplinas [PASHLER et al., 2007]. No entanto, a adoção do estudo de exemplos trabalhados ainda é pouco empregada pelos pesquisadores na área de educação em programação [SKUDDER; LUXTON-REILLY, 2014]. Quando são empregados, a maioria dos trabalhos menciona mais frequentemente a utilização dos exemplos dentro de uma disciplina de programação, tais como

Caspersen (2007) e Gray et al. (2007), do que a realização de uma pesquisa mais formal, tais como os artigos apresentados por Margulieux et al. (2012) e Morrison et al. (2015).

Além disso, os exemplos trabalhados são normalmente apresentados textual e estaticamente aos estudantes [ABDUL-RAHMAN; DU BOULAY, 2014; BIELACZYK et al., 1995; PIROLI; RECKER, 1994; SORVA; SEPPÄLÄ, 2014]. Contudo, o formato de apresentação em vídeo vem se tornando mais popular entre os pesquisadores [CASPERSEN, 2007; MARGULIEUX et al., 2012]. Uma das vantagens da utilização de exemplos em vídeo em disciplinas de programação é a possibilidade de apresentar o processo de construção de programas por profissionais mais experientes [BENNEDSEN; CASPERSEN, 2008].

Aliado a isso, muitos estudantes estudam para uma disciplina de programação de maneira inadequada [GOMES; MENDES, 2007]. Por esta razão, estes estudantes podem estudar os exemplos trabalhados em vídeo de maneira passiva ou superficial. Uma maneira de orientar os estudantes nesse processo de estudo dos exemplos trabalhados em vídeo é através da auto-explicação, uma prática cujas evidências são positivas para aqueles estudantes que dela fazem uso [PASHLER et al., 2007]. A técnica da auto-explicação auxilia os estudantes no entendimento dos exemplos trabalhados e na construção de esquemas a partir deste exemplo [CLARK et al., 2005]. Auto-explicar diferentes tipos de materiais instrucionais em diferentes disciplinas é relacionada a melhores resultados em fases subsequentes de resolução de problemas por parte daqueles estudantes que auto-explicam espontaneamente ou que são estimulados ou treinados a auto-explicar.

Ademais, dentre os métodos de ensino da teoria da Aprendizagem cognitiva, destacamos dois deles, o *scaffolding* e a reflexão. A atividade de auto-explicação será executada através da utilização de questões que servirão como mecanismos de *scaffolding* para orientar os estudantes em seus processos de reflexão enquanto estudam programas sendo implementados pelo professor. Além disso, esse processo de reflexão será reforçado na versão refinada da abordagem com o objetivo de fazer com que os estudantes tenham uma atividade adicional para refletirem sobre as estratégias adotadas pelo professor para a construção dos programas, através da reprodução pelos estudantes do que é executado nos exemplos em vídeo.

Diante deste contexto, a presente tese apresenta uma abordagem para o aprendizado de programação para estudantes iniciantes na disciplina para auto-explicação passo a passo de exemplos trabalhados em vídeo. O objetivo da abordagem proposta é minimizar dois problemas: (i) a dificuldade de combinar as diferentes estruturas de uma linguagem de programação para construir programas sintática, semântica e logicamente corretos; (ii) a maneira inadequada (ou insuficiente) que os estudantes estudam para uma disciplina introdutória de programação. Por meio da minimização destes dois problemas é possível que os estudantes iniciantes em programação tenham um desempenho mais satisfatório e apresentem melhores resultados do ponto de vista da construção de programas com menos erros sintáticos, semânticos e lógicos. Em consequência, espera-se também que eles apresentem melhores notas em disciplinas de programação.

4.2 Descrição da versão piloto da abordagem proposta

Uma disciplina introdutória de programação pode fazer parte de diferentes cursos, em diferentes níveis de escolaridade. No Brasil, os cursos de computação estão divididos em três níveis, cursos de graduação, cursos superiores de tecnologia e cursos técnicos de nível médio. Os dois primeiros englobam aqueles estudantes que concluíram o ensino médio e cursam um curso de nível superior, de longa ou curta duração, respectivamente. De acordo com o currículo de referência para os cursos de Ciência e Engenharia da Computação (2005) da Sociedade Brasileira de Computação (SBC), os cursos de graduação dividem-se naqueles que tem a Computação como atividade-fim, naqueles que têm a computação como atividade-meio e nos cursos de Licenciatura em Computação. Os cursos que tem a computação como atividade-fim são os cursos de Bacharelado em Ciência e em Engenharia da Computação. O curso que tem Computação como atividade-meio é o curso de Bacharelado em Sistemas da Informação.

Os cursos superiores de tecnologia também são cursos de graduação, mas contém carga horária menor do que os cursos de bacharelado e licenciatura e, consequentemente, são cursos mais específicos. Os cursos superiores de tecnologia em Análise e Desenvolvimento de Sistemas, em Banco de Dados e em Sistemas para Internet são alguns dos cursos na área de computação definidos no Catálogo Nacional de Cursos Superiores de Tecnologia (2010) desenvolvido pelo Ministério da Educação

(MEC). Os cursos técnicos de nível médio, por sua vez, podem ser de dois tipos: os cursos de nível médio integrado e os cursos subsequentes. Os cursos de nível médio integrado acontecem quando o aluno realiza o curso técnico ao mesmo tempo em que cursa as disciplinas do ensino médio, tais como português e matemática, por exemplo. Os cursos de nível médio subsequente ocorrem quando o aluno já finalizou o ensino médio e está cursando apenas as disciplinas do ensino técnico. Os cursos técnicos em Informática, em Informática para Internet e em Programação para Jogos Digitais são alguns dos cursos técnicos definidos no Catálogo Nacional dos Cursos Técnicos (2012) definido pelo MEC. Dentro deste contexto, a abordagem de ensino Stepwise self-explanation tem como foco os estudantes iniciantes em programação, podendo ou não estar matriculados em disciplinas introdutórias de programação.

A Stepwise self-explanation utiliza o FSI, a teoria dos exemplos trabalhados, a teoria das auto-explicações e a teoria da aprendizagem cognitiva como referencial teórico. O FSI é utilizado na estruturação de um conjunto de exemplos trabalhados gravados em vídeo por professores que são utilizados para apresentar o processo de construção de programas aos estudantes. Estes exemplos em vídeos fazem parte do material instrucional de estudantes matriculados em uma disciplina introdutória de programação. O estudo de exemplos trabalhados reduz a carga cognitiva que dificulta o aprendizado destes estudantes, proporcionando a assimilação dos diferentes passos de resolução de um determinado problema por estes estudante e a consequente construção de esquemas de resolução de problemas correspondentes [SKUDDER; LUXTON-REILLY, 2014].

Aliado a isso, os estudantes não empregam metodologias de estudo adequadas às necessidades de uma disciplina introdutória de programação [GOMES; MENDES, 2007]. Portanto, a nossa abordagem pretende estimulá-los a refletir e orientá-los a auto-explicar os exemplos trabalhados através da inserção de questões a respeito da construção de programas exibida ao longo do processo de construção dos programas nos exemplos em vídeo. Estas questões constituem-se em mecanismos de *scaffolding* e de estímulo à reflexão, métodos de ensino da Aprendizagem cognitiva que serão utilizados na nossa abordagem.

Nas seções a seguir, falaremos mais detalhadamente sobre a Stepwise self-explanation, destacando o tipo de material instrucional utilizado e o momento de auto-explicação realizado pelos estudantes.

4.2.1 Exemplos trabalhados em vídeo

Os exemplos trabalhados (Seção 3.2.1) são uma sequência de passos que demonstram a solução de um problema específico, produzida pelo professor e que pode ser utilizada pelos estudantes em seus momentos de estudo [ATKINSON et al., 2000]. Eles são um tipo de material instrucional comprovadamente eficiente para estudantes iniciantes em diferentes disciplinas, porque auxiliam a construção de novos esquemas enquanto os estudantes estão adquirindo um novo conhecimento ou habilidade [CLARK et al., 2005]. Quando apresentados em formato de vídeo, além da sequência de passos que demonstra a resolução do problema, os exemplos trabalhados possuem as explicações dadas pelo professor para a construção da solução da maneira que é apresentada. Estas explicações demonstram situações e maneiras como as estruturas de programação podem ser utilizadas e como elas podem ser combinadas para a construção da solução.

Na versão piloto da abordagem, utilizamos exemplos trabalhados em vídeo da série *The Joy of Code* que ensinam a linguagem de programação Java utilizando a ferramenta *Greenfoot* e refletem as ideias de construção incremental do código definidas no FSI. Nestes exemplos em vídeo, o professor realiza apenas trajetórias uniformes de desenvolvimento, compostas somente por extensões seguidas por refinamentos, e por esta razão adequados ao propósito de ensinar estudantes iniciantes em programação, conforme objetiva a nossa abordagem. Ao executar uma atividade de extensão, o professor descreve brevemente o objetivo que vai ser alcançado. Ao executar uma atividade de refinamento, o professor constrói a implementação que alcança o objetivo identificado. Portanto, à medida que vai aprendendo a partir destes exemplos em vídeo, os estudantes devem alcançar os objetivos de aprendizagem definidos a partir do FSI e apresentados no Quadro 4.1.

Quadro 4.1. Objetivos de aprendizagem que são alcançados ao se programar a partir do FSI.

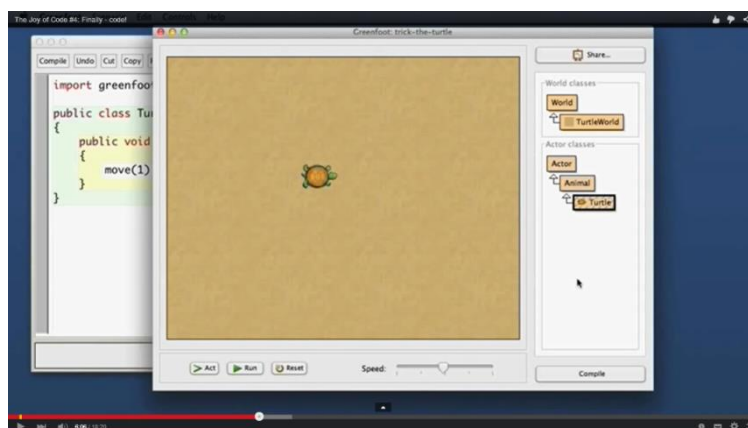
Atividade	Objetivos de aprendizagem
Extensão	Identificar uma meta que deve ser alcançada com o caso (de uso) sendo estendido.
Refinamento	Aplicar os conceitos de uma linguagem de programação para construir uma implementação que corresponde à meta descrita previamente.

Fonte: Própria autora.

Para ilustrar essa execução, uma imagem da tela logo depois que o professor executa uma atividade de refinamento para o objetivo “Fazer a tartaruga se mover” em

um trecho de vídeo¹⁵ do episódio número 4 da série *The Joy of Code* é exibida na Figura 4.1. Antes de chegar a este ponto do vídeo, o professor informa que vai fazer a tartaruga se mover e escreve o código `move(1)`; dentro do método `void act()` que alcança o objetivo descrito.

Figura 4.1. Imagem de tela de trecho do exemplo em vídeo *The Joy of Code*.



Fonte: <http://bit.ly/1QipxNo>

Logo depois, o professor continua informando que vai construir uma implementação cujo objetivo é “Fazer a tartaruga se mover um pouco mais rápido”. Para isso ele modifica o código para `move(4)`; dentro do mesmo método `act`. Reproduzimos Figura 4.2 a parte da representação diagramática em notação BPMN [OMG, 2011] do FSI que contém somente as atividades de extensão e refinamento, uma vez que são estas as atividades nas quais estamos interessados. Utilizamos essa representação para ilustrar como a sequência de passos do FSI é refletida nos exemplos trabalhados utilizados na primeira versão da abordagem.

¹⁵ Episódio número 4 do The Joy of Code. Disponível em: < <http://bit.ly/1QipxNo>>. Acesso em: 11 out. 2015.


```

        move (4) ;
    }
    • (...)

```

Como as atividades nos vídeos são realizadas pelo professor, através destes exemplos em vídeo é possível aos estudantes observar a maneira como as atividades são realizadas e construir esquemas do processo de construção de programas que são empregados pelo professor. Por exemplo, na descrição realizada anteriormente, durante a primeira sequência de extensão e refinamento, o estudante consegue observar que o método `move` foi utilizado com o objetivo de fazer o objeto tartaruga se mover, que ele foi escrito recebendo por parâmetro a quantidade de passos que deve ser dada pelo objeto e que ele foi escrito dentro do método `act`. No entanto, embora as explicações instrucionais sejam importantes para os estudantes iniciantes, apresentá-las somente aos estudantes não é suficiente para que eles construam esquemas. É importante que estas explicações fornecidas pelos estudantes sejam combinadas com atividades de construção do conhecimento [WITTEWER; RENKL, 2008]. Veremos na Seção 4.2.2 que apoiamos a construção de esquemas do processo de construção de programas exibido aos estudantes nos exemplos em vídeo por meio da auto-explicação.

4.2.2 Momentos de auto-explicação

Vimos na revisão da literatura do Capítulo 2 que enquanto estudam para uma disciplina de programação, os estudantes podem adotar estratégias de estudo que focam mais na memorização do que no entendimento dos conceitos [GOMES; MENDES, 2007]. Além disso, os estudantes podem estudar de maneira passiva, sem se engajar adequadamente nos seus processos de aprendizagem [CHIU; CHI, 2014]. Consequentemente, tais estudantes acabam não processando o conteúdo da disciplina de programação profundamente, levando a um aprendizado deficiente [CLARK et al., 2005].

Além disso, vimos na Seção 3.3 que as auto-explicações são uma atividade construtiva de aprendizagem que auxilia os estudantes no entendimento de um exemplo e na construção de um esquema a partir deste exemplo. Com base nesta definição, para que o aprendizado ocorra, propomos que os estudantes reflitam ativamente sobre o processo de construção de programas apresentado nos exemplos em vídeo através da atividade de auto-explicação. A auto-explicação pode ser explicitamente solicitada por

um mediador, pode ser incitada através de questões e também pode ser treinada. No caso da primeira versão da nossa abordagem, a atividade de auto-explicação foi estimulada através da inclusão de um conjunto de questões que deviam ser respondidas por escrito pelos estudantes no decorrer da execução dos exemplos em vídeo.

Aliado a isso, vimos na Seção 3.4 que o método de ensino *scaffolding* da teoria da aprendizagem cognitiva, refere-se ao apoio fornecido pelo professor para auxiliar o estudante na execução de uma atividade. Sendo assim, estas questões para a auto-explicação funcionam como mecanismos de *scaffolding* que orientam os estudantes durante o processo de reflexão envolvido no estudo dos exemplos em vídeo. Espera-se com a inclusão destas questões que os estudantes sejam estimulados a refletir sobre as atividades executadas nos exemplos estudados e, conseqüentemente, aprendam a construir programas mais corretamente, aplicando as estruturas de programação aprendidas de maneira adequada. Este objetivo está alinhado com os objetivos de aprendizagem relacionadas à escrita e à implementação de programas definidos para a disciplina de Conceitos Fundamentais de Programação pelo mais recente Currículo da ACM [J. T. F. ON COMPUTING CURRICULA, 2013]. São eles:

- Escrever programas que utilizem dados de tipos primitivos (e.g., números, caracteres, booleanos); e
- (...) Implementar (...) um programa que utilize o seguinte construto fundamental da programação: computação básica (...).

Como a nossa abordagem visa os estudantes iniciantes em programação, estamos considerando apenas trajetórias de desenvolvimento uniforme, que contém apenas atividades de extensão seguidas por atividades de refinamento. Para ajudar os estudantes a estudar o material disponibilizado, depois de cada extensão e refinamento executado pelo professor os alunos são convidados a se engajar em uma atividade de auto-explicação que está alinhada com a extensão e o refinamento que eles acabaram de assistir nos vídeos. Levando isso em consideração, nesta primeira versão da abordagem, temos duas categorias de questões para promover a auto-explicação que estão alinhadas com as questões adotadas na estratégia de treinamento para auto-explicação proposta em [BIELACZYK et al., 1995]. São elas:

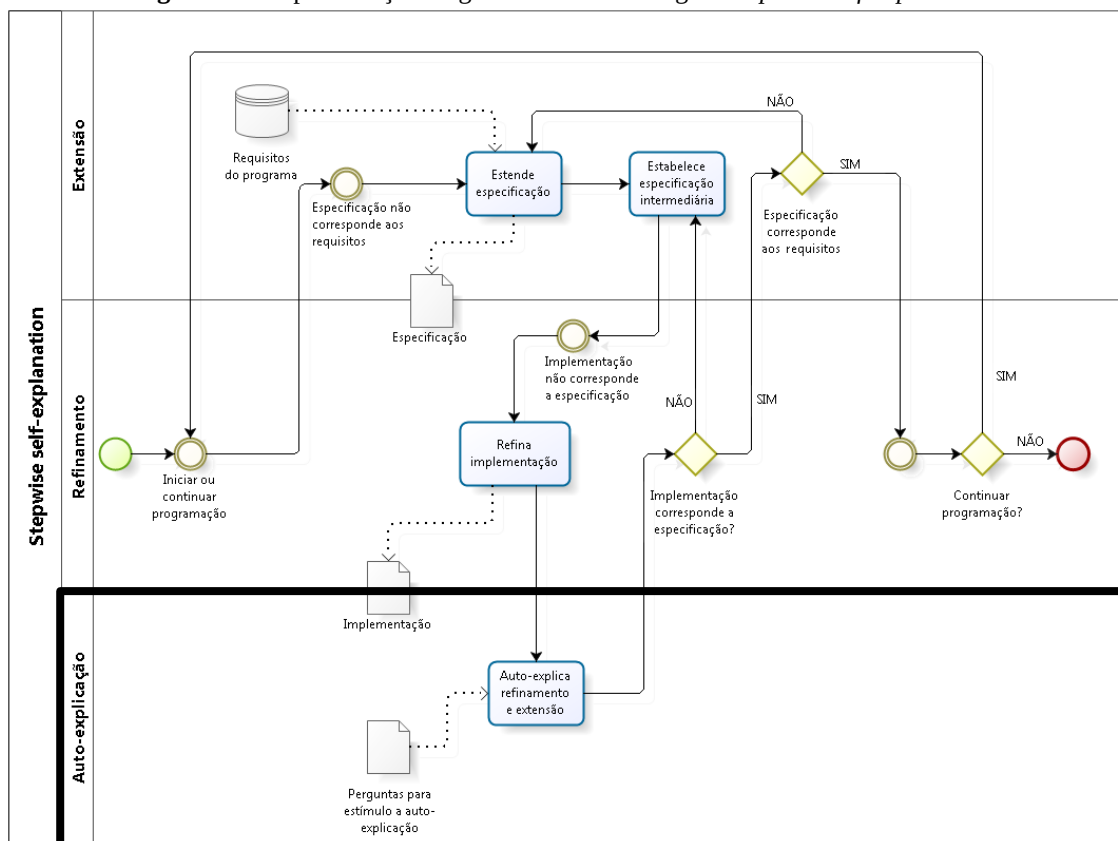
- (i) questões relacionadas à atividade de extensão, que auxiliam na reflexão sobre a identificação do objetivo ou da proposta do código sendo estendido. Assim,

o estudante é levado a refletir sobre a situação em que determinado trecho de código é utilizado no exemplo; e

- (ii) questões relacionadas à atividade de refinamento, que auxiliam na reflexão sobre a aplicação dos conceitos de uma linguagem de programação para alcançar o objetivo identificado. Assim, o estudante é levado a refletir sobre características do exemplo que sejam importantes para o seu entendimento, tais como a sintaxe e a semântica do código implementado, a combinação de dois métodos ou de um método e seus parâmetros.

Essa combinação de assistir sequências da extensão de uma especificação seguida pela implementação de um pequeno pedaço de código e depois auto-explicar os passos que foram executados é ilustrada na Figura 4.3 através de uma representação diagramática em notação BPMN [OMG, 2011]. Observem que, ao diagrama apresentado na Figura 4.3, foi acrescentada a tarefa “Auto-explica extensão e refinamento” que corresponde aos momentos nos quais os alunos irão refletir sobre a atividade que foi executada pelo professor a partir das questões de auto-explicação. Esta abordagem foi nomeado *Stepwise Self-explanation*, ou seja, auto-explicação passo a passo, por dois motivos. Primeiro, porque ele tem como um dos seus referenciais teóricos o FSI, um *framework* que indica que o processo de programação precisa ser exibido ao estudante através da construção de programas realizada de maneira iterativa e incremental. Segundo, porque os estudantes estão auto-explicando pequenos pedaços de código que estão sendo criados iterativa e incrementalmente nos exemplos em vídeo, uma vez que eles foram estruturados de acordo com o FSI. Nesta versão piloto do *framework*, ele tem como entradas: (i) o exemplo trabalhado em vídeo contendo (ii) a descrição dos requisitos e (iii) a especificação do programa e (iv) as questões para auto-explicação. Além disso, ele tem como saídas: (i) as implementações do professor e (ii) do estudante para o programa especificado, escrito na linguagem de programação que está sendo utilizada e (iii) as respostas fornecidas pelo estudante para as questões de auto-explicação.

Figura 4.3. Representação diagramática da abordagem *Stepwise self-explanation*.



Fonte: própria autora.

Neste momento, voltamos ao exemplo do trecho de vídeo do episódio número 4 da série *The Joy of Code* descrito na Seção 4.2.1. Depois de assistir ao trecho de vídeo cuja pausa é apresentada na imagem de tela da Figura 4.1, os estudantes foram convidados a responder a um conjunto de questões relacionadas às atividades de extensão e refinamento realizadas. Relembrando, neste trecho de vídeo, o professor tem como objetivo escrever o código que fará a tartaruga se mover. Logo depois, o vídeo é pausado e os estudantes podem responder às questões para auto-explicação relacionadas às atividades que foram desenvolvidas. As questões de auto-explicação definidas para o trecho do exemplo descrito são apresentadas no Quadro 4.2.

Quadro 4.2. Questões para a auto-explicação relacionadas ao vídeo do The Joy of Code.

Atividade	Questões
Extensão	Qual a principal objetivo do pedaço de código que ele (professor) acabou de escrever?
Refinamento	Quais os passos que ele (professor) realizou para alcançar os objetivos que ele desejava? Qual o código que ele (professor) escreveu para fazer a tartaruga andar? Qual a proposta do parâmetro (número '1') no método <code>move</code> ? Onde ele (professor) escreveu o código para fazer a tartaruga andar? Como os métodos <code>act</code> e <code>move</code> estão relacionados? Quando ele (professor) utilizou o método <code>move</code> ?

	Depois que ele (professor) compilou e rodou o código, como a tartaruga se comportou?
--	--

Fonte: própria autora.

Neste quadro, temos a questão “Qual o principal objetivo do pedaço de código que ele (professor) acabou de escrever?” que é relacionada à atividade de extensão e foi definida para que os estudantes fossem estimulados a refletir sobre qual o objetivo do trecho de código que foi implementado pelo professor no exemplo em vídeo. No caso do exemplo, o estudante é estimulado a refletir sobre a situação em que ele deve utilizar o método que faz o objeto tartaruga se mover. Para a atividade de refinamento, por exemplo, temos as questões “Qual o código que ele (professor) escreveu para fazer a tartaruga andar?” e “Qual a proposta do parâmetro (número ‘1’) no método `move`?” que foram definidas para que os estudantes fossem estimulados a refletir sobre a implementação do código e o significado do parâmetro que o método recebe, respectivamente. Sendo assim, através destas questões, o estudante é estimulado a refletir sobre (i) a sintaxe do código que o professor escreveu, (ii) o significado do parâmetro no método e (iii) sobre a combinação do método e do parâmetro para fazer com que o objeto tartaruga se movesse.

Com a inclusão deste conjunto de questões ao término de cada sequência de extensão e refinamento, atribuímos um papel ao estudante na trajetória percorrida na representação diagramática da Figura 4.3. Sendo assim, parte do caminho percorrido agora na representação diagramática ocorrerá da seguinte maneira:

- Professor:
 - Extensão [Tarefa: Estende especificação]:
 - Meta a ser alcançada: “Fazer a tartaruga se mover”
 - Refinamento [Tarefa: Estabelece especificação intermediária seguida pela tarefa Refina implementação]:


```
public void act() {
    move(1);
}
```
- Estudante:
 - Auto-explicação [Tarefa: Auto-explica extensão e refinamento]
 - É estimulado a refletir sobre as atividades de extensão e refinamento desempenhadas pelo professor no exemplo através das questões apresentadas no Quadro 4.2.

- (...)

De acordo com Wittwer e Renkl (2008), fornecer explicações instrucionais aos estudantes quando eles são iniciantes em determinado assunto é uma maneira apropriada de auxiliá-los na construção de esquemas. No entanto, ainda de acordo com estes pesquisadores, para que o aprendizado ocorra de maneira efetiva é necessário que as explicações instrucionais não substituam as atividades de construção de conhecimento realizadas pelos estudantes, mas que sejam integradas a estas atividades. Portanto, a abordagem de fazer com que os estudantes assistam os exemplos em vídeo e depois façam o seu processamento através da auto-explicação está completamente alinhada com estas diretrizes.

4.3 Descrição da versão refinada da abordagem proposta

A versão piloto da abordagem *Stepwise Self-explanation* proposta nesta tese foi revisada e ampliada. A descrição da versão refinada da abordagem, juntamente com os conceitos envolvidos, será apresentada nesta seção.

4.3.1 Exemplos trabalhados em vídeo

Da mesma forma que os exemplos em vídeo do *The Joy of Code*, os exemplos em vídeo produzidos na versão refinada da abordagem apresentam apenas trajetórias uniformes de desenvolvimento, compostas por extensões seguidas por refinamentos. No entanto, as extensões aqui serão realizadas de uma maneira ligeiramente diferente, uma vez que teremos extensões principais e extensões intermediárias ao longo do processo de construção de programas. Agora, cada exemplo em vídeo representa uma atividade de extensão principal que é seguida por um conjunto de extensões intermediárias. Além disso, para cada atividade de extensão intermediária, temos uma atividade de refinamento associada.

Sendo assim, os exemplos trabalhados produzidos na versão refinada da abordagem foram definidos de acordo com a estrutura descrita a seguir. Em um primeiro momento, o professor descreve o enunciado do problema que vai constituir-se na descrição do requisito que precisa ser alcançado pela especificação sendo estendida. Logo depois, é apresentada a especificação correspondente a este requisito, ou seja, o que precisa ser alcançado através de um conjunto de entradas e saídas. Em um terceiro momento, o professor começa a construir o programa iterativa e incrementalmente a

partir de especificações intermediárias, que correspondem a descrições concisas de objetivos intermediários que precisam ser alcançados até que se alcance a especificação principal. A cada especificação intermediária descrita, o professor executa uma atividade de refinamento através da construção de uma implementação que atende ao objetivo intermediário correspondente. Por fim, ao construir uma implementação que corresponda à especificação principal apresentada no início do exemplo, o professor mostra uma execução do código em funcionamento. Sendo assim, à medida que vão aprendendo o processo de programação, os estudantes devem alcançar os mesmos objetivos de aprendizagem definidos a partir do FSI para a primeira versão da abordagem e reproduzidos no Quadro 4.3.

Quadro 4.3. Objetivos de aprendizagem que são alcançados ao se programar a partir do FSI.

Atividade	Objetivos de aprendizagem
Extensão	Identificar uma meta que deve ser alcançada com o caso (de uso) sendo estendido.
Refinamento	Aplicar os conceitos de uma linguagem de programação para construir uma implementação que corresponde à meta descrita previamente.

Fonte: própria autora.

Para ilustrar a estrutura dos exemplos trabalhados em vídeo produzidos, apresentamos no Quadro 4.4 um exemplo trabalhado utilizado como material instrucional em um dos experimentos realizados nesta segunda versão da abordagem¹⁷. Este exemplo trabalhado foi construído para estudantes repetentes¹⁸ matriculados em uma disciplina introdutória de programação sendo ministrada de uma maneira tradicional (aula expositiva combinada com apresentação de exemplos estáticos), utilizando a linguagem de programação C e a ferramenta de desenvolvimento Codeblocks¹⁹. Este exemplo trabalhado foi utilizado como material de reforço em uma aula extracurricular ministrada pela autora deste trabalho para os estudantes voluntários e tinha como objetivo de aprendizagem:

[OA1] Aplicar os conceitos de variáveis, estruturas de repetição e estruturas de decisão para a construção de um programa na linguagem de programação C.

¹⁷ Exemplo trabalhado. Disponível em < <http://bit.ly/1JoAny9> >. Acesso em: 19 nov. 2015.

¹⁸ Tais estudantes foram classificados como estudantes com sérias dificuldades na disciplina, pois estavam repetindo-a pelo menos pela segunda vez.

¹⁹ Codeblocks. Disponível em: < <http://www.codeblocks.org/> >. Acesso em: 19 nov. 2015.

Além disso, também temos representadas entre colchetes as atividades de extensão e refinamento correspondentes na representação diagramática do FSI que apresentamos na Figura 3.2 e foram executadas pelo professor no exemplo em vídeo descrito no Quadro 4.4.

Quadro 4.4. Script do exemplo de exemplo trabalhado em vídeo.

Enunciado do problema	No Brasil, elegemos os nossos governantes através do voto. Os brasileiros menores de 16 anos não podem votar. Os jovens entre 16 e 18 anos e os idosos a partir dos 70 anos tem voto facultativo. Já para quem tem entre 18 e 70 anos, o voto é obrigatório. Com base nessas informações, construa um programa que recebe como entrada a idade de uma pessoa e informa como saída se a pessoa pode votar e se o voto é obrigatório ou facultativo. Esse programa deve executar repetidas vezes e deve parar quando a idade informada for -1.	
Especificação do problema (Extensão principal)	Extensão principal [Tarefa: estende especificação]	
	Entrada	Saída
	Digite uma idade: 15	Não pode votar.
	Digite uma idade: 16	Pode votar. Voto opcional.
	Digite uma idade: 18	Pode votar. Voto obrigatório.
	Digite uma idade: 70	Pode votar. Voto opcional.
	Digite uma idade: -1	
Passo 1 (Extensão intermediária seguida de Refinamento ²⁰)	- Extensão intermediária [Tarefa: Estabelece especificação intermediária] - Ler o número que vai ser digitado pelo usuário. - Refinamento [Tarefa: Refina implementação] <pre>int main(void) { int idade = 0; printf("Digite uma idade: "); scanf("%d", &idade); system("PAUSE"); return 0; }</pre>	
Passo 2 (Extensão intermediária seguida de Refinamento)	- Extensão intermediária [Tarefa: Estabelece especificação intermediária] - Informar quem não pode votar. - Refinamento [Tarefa: Refina implementação] <pre>int main(void) { int idade = 0; printf("Digite uma idade: "); scanf("%d", &idade); if(idade >= 0 && idade < 16){</pre>	

²⁰ O código que vai sendo apresentado em cada um dos passos no quadro está destacado em linhas pontilhadas.

	<pre> printf("Não pode votar"); } system("PAUSE"); return 0; } </pre>
Passo 3 (Extensão intermediária seguida de Refinamento)	- Extensão intermediária [Tarefa: Estabelece especificação intermediária] - Informar quem pode votar, mas o voto é opcional. - Refinamento [Tarefa: Refina implementação] <pre> int main(void) { int idade = 0; printf("Digite uma idade: "); scanf("%d", &idade); if(idade >= 0 && idade < 16){ printf("Não pode votar"); } else if(idade >= 16 && idade < 18 idade >= 70){ printf("Pode votar. Voto opcional."); } system("PAUSE"); return 0; } </pre>
Passo 4 (Extensão intermediária seguida de Refinamento)	- Extensão intermediária [Tarefa: Estabelece especificação intermediária] - Informar quem pode votar, mas o voto é obrigatório. - Refinamento [Tarefa: Refina implementação] <pre> int main(void) { int idade = 0; printf("Digite uma idade: "); scanf("%d", &idade); if(idade >= 0 && idade < 16){ printf("Não pode votar"); } else if(idade >= 16 && idade < 18 idade >= 70){ printf("Pode votar. Voto opcional."); } else if(idade >= 18 && idade < 70){ printf("Pode votar. Voto obrigatório."); } system("PAUSE"); return 0; } </pre>
Passo 5 (Extensão)	Extensão intermediária [Tarefa: Estabelece especificação intermediária]

intermediária seguida de Refinamento)	○ Criar o laço para fazer com que várias pessoas verifiquem se podem votar ou não até que alguém digite -1 como idade. • Refinamento [Tarefa: Refina implementação] <pre> int main(void) { int idade = 0; while(idade >= 0) { printf("Digite uma idade: "); scanf("%d", &idade); if(idade >= 0 && idade < 16){ printf("Não pode votar"); } else if(idade >= 16 && idade < 18 idade >= 70){ printf("Pode votar. Voto opcional."); } else if(idade >= 18 && idade < 70){ printf("Pode votar. Voto obrigatório."); } } system("PAUSE"); return 0; } </pre>
---------------------------------------	--

Fonte: própria autora.

Dissemos anteriormente que os exemplos trabalhados em vídeo estruturados de acordo com o FSI seriam utilizados pelos estudantes para que eles minimizassem o problema de não conseguirem combinar as diferentes estruturas de uma linguagem de programação para construir um programa. Através da construção passo a passo exibida no exemplo trabalhado em vídeo (mostrado no Quadro 4.4), os alunos podem perceber como as diferentes estruturas são combinadas. Por exemplo, vimos que no passo 1 o professor primeiro descreveu uma meta intermediária “Ler o número que vai ser digitado pelo usuário” para à qual ele iria construir código. Logo depois, ele escreveu um comando `printf` para indicar ao usuário o que ele deveria digitar e, em seguida, ele combinou a esta estrutura um comando `scanf` que faz a leitura do dado digitado pelo usuário. No passo 2, vimos que o professor descreveu uma meta intermediária “Informar quem não pode votar” para a qual ele iria implementar código. Em seguida, ele combinou ao código já existente o comando `if` que verifica se a idade está entre 0 e 16 e, em caso positivo, exibe uma mensagem informando que a pessoa não pode votar. E assim, ele foi sucessivamente combinando novos trechos de código a cada novo passo que executou até finalizar com um programa que atende a especificação principal

definida no início do exemplo trabalhado, fazendo com que o programa se tornasse consistente. A especificação, por sua vez, corresponde ao conjunto de requisitos indicado no enunciado do problema, portanto o programa também ficou total.

Embora as explicações instrucionais sejam importantes para os estudantes iniciantes, simplesmente apresentá-las aos estudantes e não engajá-los em atividades de construção do conhecimento não é suficiente para o aprendizado efetivo. É importante que estas explicações fornecidas pelos estudantes sejam combinadas com atividades de construção do conhecimento [WITTEWER; RENKL, 2008], pois somente desta maneira eles vão conseguir adquirir os esquemas que eles necessitam para posteriormente aplicá-los na construção de programas. Veremos na Seção 4.3.2 como apoiamos a construção de esquemas do processo de construção de programas na versão refinada da abordagem.

4.3.2 Momentos de auto-explicação

Na versão piloto da abordagem *Stepwise Self-explanation*, os estudantes foram estimulados a auto-explicar após uma sequência de extensão e refinamento. No entanto, os resultados que obtivemos no experimento piloto realizado com a versão da abordagem nos indicaram que, embora as perguntas utilizadas na nossa abordagem fossem um ponto de partida conveniente, elas precisavam ser refinadas. Sendo assim, voltamos ao referencial teórico sobre auto-explicação e procuramos analisar como poderíamos fazer para estimular esta atividade na versão refinada da abordagem.

Vimos na Seção 4.3.1 que os exemplos trabalhados em vídeo criados são estruturados da seguinte maneira: primeiro, o enunciado do problema é apresentado. Este enunciado é seguido por exemplos de execução do problema com entradas específicas e suas saídas correspondentes, o que constitui-se na extensão principal. Logo depois, é apresentada uma possível solução para o problema constituída por uma sequência de passos que soluciona o problema apresentado. Cada passo nesta solução representa uma extensão intermediária seguida por um refinamento. Por fim, para exibir o funcionamento do programa é apresentada uma amostra de sua execução logo depois que a sua implementação foi finalizada.

Para ajudar os estudantes a estudar os exemplos trabalhados disponibilizados, incluímos ao longo destes exemplos, perguntas que serviram como mecanismos de *scaffolding* para estimular a auto-explicação dos estudantes e orientá-los durante o processo de reflexão envolvido neste momento de estudo. Levando isso em

consideração, a versão refinada da abordagem, temos quatro categorias de questões para estímulo a auto-explicação, todas alinhadas com as atividades que os estudantes estão observando nos exemplos em vídeo. São elas:

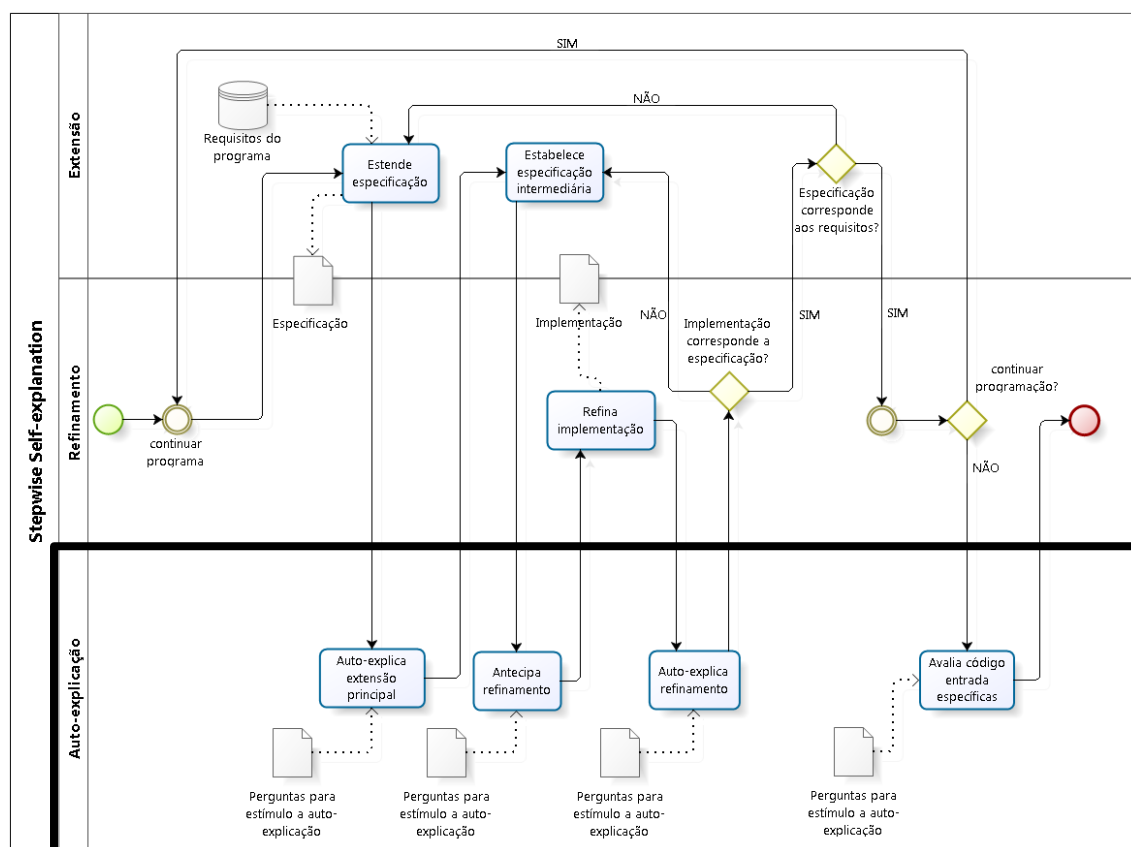
- (i) questões relacionadas à atividade de extensão principal, que incluem a identificação do objetivo geral do programa que está sendo estendido. Estas perguntas são apresentadas após o enunciado do problema e de seus exemplos de entrada e saída. Elas têm por finalidade levar o estudante a refletir sobre o propósito geral do programa que será implementado. Questões desta categoria estão alinhadas com a estratégia de treinamento para auto-explicação proposta por Bielaczyc et al. (1995);
- (ii) questões relacionadas à atividade de extensão intermediária, que incluem antecipar a implementação de um trecho de código que atenda a meta intermediária que será alcançada com o passo em questão. Estas perguntas são apresentadas após se informar a funcionalidade que será implementada no passo e antes de se apresentar a implementação para o referido passo. Elas têm como propósito levar o estudante a refletir sobre a aplicação das estruturas de uma linguagem de programação para o passo em questão, ou seja, sobre uma possível solução para o passo que será implementado a seguir no vídeo, antes da solução do professor ser apresentada. No caso em que o estudante fornecer uma solução para o passo em questão, sugere-se que ele compare a solução dada por ele com a solução que é dada pelo professor. Questões desta categoria estão alinhadas com os resultados que foram encontrados por Renkl (2002) que indicaram que os estudantes que foram raciocinadores antecipativos tiveram melhores resultados na sua pesquisa;
- (iii) questões relacionadas à atividade de refinamento, que incluem a aplicação dos conceitos de uma linguagem de programação para alcançar o objetivo do passo em questão e o funcionamento do trecho de código que foi implementado. Elas têm como propósito levar o estudante a refletir sobre a solução que foi apresentada pelo professor, processando as informações que foram fornecidas por ele. Questões desta categoria estão alinhadas com os achados de Berthold e Renkl (2010) que indicam

que perguntas para verificar posteriormente o entendimento dos estudantes a respeito das explicações fornecidas pelo professor aumentam os seus efeitos; e

- (iv) questões relacionadas à atividade de resolução do problema como um todo, que incluem o funcionamento global do programa depois que toda a implementação para ele foi fornecida. Elas têm como propósito levar o estudante a refletir sobre a execução do programa apresentado, calculando mentalmente a saída do código a partir de uma determinada entrada. Questões desta categoria estão alinhadas com a estratégia para auto-explicação proposta por Bielaczyc et al. (1995) e por Pirolli et al. (1994).

A versão refinada da abordagem *Stepwise Self-explanations* é ilustrada na Figura 4.4 através de uma representação diagramática em notação BPMN [OMG, 2011]. Observem que, ao diagrama apresentado na Figura 3.2, foi acrescentada a raia “Auto-explicação” juntamente com as tarefas “Auto-explica extensão principal”, “Antecipa refinamento”, “Auto-explica refinamento” e “Avalia código com entradas específicas” que correspondem aos momentos nos quais os alunos irão refletir sobre as atividades que foram executadas pelo professor a partir das perguntas de estímulo a auto-explicação. A tarefa “Auto-explica extensão principal” corresponde às questões (i) relacionadas à atividade de extensão principal; a tarefa “Antecipa refinamento” corresponde às questões (ii) relacionadas à atividade de extensão intermediária; a tarefa “Auto-explica refinamento” corresponde às questões (iii) relacionadas à atividade de refinamento e, finalmente, a tarefa “Avalia código com entradas específicas” corresponde às questões (iv) relacionadas à atividade de resolução do problema como um todo. A versão refinada da abordagem *Stepwise self-explanation* tem como entradas: (i) o exemplo trabalhado em vídeo contendo (ii) a descrição dos requisitos e (iii) a especificação do programa e (iv) as questões para auto-explicação. Além disso, ele tem como saídas: (i) as implementações do professor e (ii) do estudante para o programa especificado, escrito na linguagem de programação que está sendo utilizada e (iii) as respostas fornecidas pelo estudante para as questões de auto-explicação.

Figura 4.4. Versão refinada da abordagem *Stepwise self-explanation*.



Fonte: Própria autora.

Além do *scaffolding*, os estudantes foram estimulados a adotar o método de ensino da reflexão que também faz parte da aprendizagem cognitiva. Para colocar este método em prática, antes que os estudantes começassem a estudar os exemplos trabalhados em vídeo, foi sugerido a eles que, além de visualizar os exemplos e responder às questões como forma de estudo do conteúdo sendo apresentado, eles reproduzissem o código que estava sendo construído, para que pudessem comparar os processos de resolução de problemas do professor com os deles próprios. Com os exemplos trabalhados sendo apresentados em vídeo é possível que os estudantes manipulem (i.e., reproduzam, pausem, voltem, avancem) trechos do vídeo, refazendo os passos realizados pelos professor e compará-los repetidamente com os passos que eles próprios percorrem enquanto estudam a construção dos programas apresentados. No Quadro 4.5, voltamos ao exemplo de exemplo trabalhado em vídeo apresentado anteriormente no Quadro 4.4 para ilustrar como acontecem os momentos de auto-explicação previstos na versão refinada da abordagem.

Quadro 4.5. Script do exemplo de exemplo trabalhado em vídeo contendo os momentos de auto-explicação.

• Professor:		
Enunciado do problema	No Brasil, elegemos os nossos governantes através do voto. Os brasileiros menores de 16 anos não podem votar. Os jovens entre 16 e 18 anos e os idosos a partir dos 70 anos tem voto facultativo. Já para quem tem entre 18 e 70 anos, o voto é obrigatório. Com base nessas informações, construa um programa que recebe como entrada a idade de uma pessoa e informa como saída se a pessoa pode votar e se o voto é obrigatório ou facultativo. Esse programa deve executar repetidas vezes e deve parar quando a idade informada for -1.	
Especificação do problema (Extensão principal)	• Extensão principal [Tarefa: estende especificação]	
	Entrada	Saída
	Digite uma idade: 15	Não pode votar.
	Digite uma idade: 16	Pode votar. Voto opcional.
	Digite uma idade: 18	Pode votar. Voto obrigatório.
	Digite uma idade: 70	Pode votar. Voto opcional.
Digite uma idade: -1		
• Estudante:		
Momento de auto-explicação (Auto-explica extensão principal)	• Pergunta para auto-explicação: ○ Explique abaixo o que o programa apresentado nesse vídeo deve fazer.	
• Professor:		
Passo 1 (Extensão intermediária)	• Extensão intermediária [Tarefa: Estabelece especificação intermediária] ○ Meta: Ler o número que vai ser digitado pelo usuário.	
• Estudante:		
Momento de auto-explicação (Antecipa refinamento)	• Pergunta para auto-explicação: ○ Eu tenho alguma ideia para a implementação desse primeiro passo (ler o número que vai ser digitado pelo usuário)? Se sim, como eu poderia implementá-lo? Escreva aqui o código resultante e depois compare mentalmente com o código que será apresentado a seguir.	
• Professor:		
Continuação do Passo 1 (Refinamento ²¹)	• Refinamento [Tarefa: Refina implementação] <pre>int main(void) { int idade = 0; printf("Digite uma idade: "); scanf("%d", &idade); system("PAUSE"); return 0; }</pre>	
• Estudante:		

²¹ O código que vai sendo apresentado em cada um dos passos no quadro está destacado em linhas pontilhadas.

Momento de auto-explicação (Auto-explica refinamento)	- Pergunta para auto-explicação: - Explique o que faz o trecho de código que acabou de ser implementado ('int idade', 'printf' e 'scanf'), destacando porque as estruturas utilizadas são as mais adequadas para a sua implementação e como elas funcionam.
• Professor:	
Passo 2 (Extensão intermediária)	- Extensão intermediária [Tarefa: Estabelece especificação intermediária] - Meta: Informar quem não pode votar.
• Estudante:	
Momento de auto-explicação (Antecipa refinamento)	- Pergunta para auto-explicação: - Eu tenho alguma ideia para a implementação desse segundo passo (informar quem não pode votar)? Se sim, como eu poderia implementá-lo? Escreva aqui o código resultante e depois compare mentalmente com o código que será apresentado a seguir.
• Professor:	
Continuação do Passo 2 (Refinamento)	Refinamento [Tarefa: Refina implementação] <pre>int main(void) { int idade = 0; printf("Digite uma idade: "); scanf("%d", &idade); if(idade >= 0 && idade < 16){ printf("Não pode votar"); } system("PAUSE"); return 0; }</pre>
• Estudante:	
Momento de auto-explicação (Auto-explica refinamento)	- Pergunta para auto-explicação: - Explique o que faz o trecho de código que acabou de ser implementado, destacando porque as estruturas utilizadas ('if' e 'printf') são as mais adequadas para a sua implementação e como elas funcionam.
• Professor:	
Passo 3 (Extensão intermediária)	- Extensão intermediária [Tarefa: Estabelece especificação intermediária] - Meta: Informar quem pode votar, mas o voto é opcional.
• Estudante:	
Momento de auto-explicação (Antecipa refinamento)	- Pergunta para auto-explicação: - Eu tenho alguma ideia para a implementação desse terceiro passo (informar quem pode votar, mas o voto é opcional)? Se sim, como eu poderia implementá-lo? Escreva aqui o código resultante e depois compare mentalmente com o código que será apresentado a seguir.
• Professor:	
Continuação do Passo 3 (Refinamento)	Refinamento [Tarefa: Refina implementação] <pre>int main(void) { int idade = 0;</pre>

	<pre> printf("Digite uma idade: "); scanf("%d", &idade); if(idade >= 0 && idade < 16){ printf("Não pode votar"); } else if(idade >= 16 && idade < 18 idade >= 70){ printf("Pode votar. Voto opcional."); } system("PAUSE"); return 0; } </pre>
• Estudante:	
Momento de auto-explicação (Auto-explica refinamento)	- Pergunta para auto-explicação: - Explique o que faz o trecho de código que acabou de ser implementado, destacando porque as estruturas utilizadas ('else if' e 'printf') são as mais adequadas para a sua implementação e como elas funcionam.
• Professor:	
Passo 4 (Extensão intermediária)	- Extensão intermediária [Tarefa: Estabelece especificação intermediária] - Meta: Informar quem pode votar, mas o voto é obrigatório.
• Estudante:	
Momento de auto-explicação (Antecipa refinamento)	- Pergunta para auto-explicação: - Eu tenho alguma ideia para a implementação desse quarto passo (informar quem pode votar, mas o voto é obrigatório)? Se sim, como eu poderia implementá-lo? Escreva aqui o código resultante e depois compare mentalmente com o código que será apresentado a seguir.
• Professor:	
Continuação do Passo 4 (Refinamento)	Refinamento [Tarefa: Refina implementação] <pre> int main(void) { int idade = 0; printf("Digite uma idade: "); scanf("%d", &idade); if(idade >= 0 && idade < 16){ printf("Não pode votar"); } else if(idade >= 16 && idade < 18 idade >= 70){ printf("Pode votar. Voto opcional."); } else if(idade >= 18 && idade < 70){ printf("Pode votar. Voto obrigatório."); } system("PAUSE"); return 0; } </pre>

• Estudante:	
Momento de auto-explicação (Auto-explica refinamento)	- Pergunta para auto-explicação: - Explique o que faz o trecho de código que acabou de ser implementado, destacando porque as estruturas utilizadas ('else if' e 'printf') são as mais adequadas para a sua implementação e como elas funcionam.
• Professor:	
Passo 5 (Extensão intermediária)	- Extensão intermediária [Tarefa: Estabelece especificação intermediária] - Meta: Criar o laço para fazer com que várias pessoas verifiquem se podem votar ou não até que alguém digite -1 como idade.
• Estudante:	
Momento de auto-explicação (Antecipa refinamento)	- Pergunta para auto-explicação: - Eu tenho alguma ideia para a implementação desse quinto passo (criar o laço que fazer várias pessoas verificarem se podem votar ou não)? Se sim, como eu poderia implementá-lo? Escreva aqui o código resultante e depois compare mentalmente com o código que será apresentado a seguir.
• Professor:	
Continuação do Passo 5 (Refinamento)	Refinamento [Tarefa: Refina implementação] <pre> int main(void) { int idade = 0; while(idade >= 0) { printf("Digite uma idade: "); scanf("%d", &idade); if(idade >= 0 && idade < 16){ printf("Não pode votar"); } else if(idade >= 16 && idade < 18 idade >= 70){ printf("Pode votar. Voto opcional."); } else if(idade >= 18 && idade < 70){ printf("Pode votar. Voto obrigatório."); } } system("PAUSE"); return 0; } </pre>
• Estudante:	
Momento de auto-explicação (Auto-explica refinamento)	- Pergunta para auto-explicação: - Explique o que faz o trecho de código que acabou de ser implementado, destacando porque as estruturas utilizadas ('while') são as mais adequadas para a sua implementação e como elas funcionam.
• Professor:	
Mostra execução do programa	Execução do código: Digite uma idade: 1 Não pode votar.

	Digite uma idade: 15 Não pode votar. Digite uma idade: 16 Pode votar. Voto opcional. Digite uma idade: 18 Pode votar. Voto obrigatório. Digite uma idade: 55 Pode votar. Voto obrigatório. Digite uma idade: 66 Pode votar. Voto obrigatório. Digite uma idade: -1 Pressione qualquer teclar para continuar...
• Estudante:	
Momento de auto-explicação (Avalia código entrada específicas)	• Perguntas para auto-explicação: ○ Se eu executar mentalmente o código resultante desse vídeo e, nessa execução, o usuário fornecer como entrada 11, qual será a saída do programa? ○ Se eu executar mentalmente o código resultante desse vídeo e, nessa execução, o usuário fornecer como entrada 33, qual será a saída do programa? ○ Se eu executar mentalmente o código resultante desse vídeo e, nessa execução, o usuário fornecer como entrada -2, qual será a saída do programa?

Fonte: Própria autora.

Na Figura 4.5 apresentamos o trecho referente ao passo 1 do exemplo em vídeo apresentando no Quadro 4.5, onde utilizamos a ferramenta *Codeblocks*. Juntamente com o trecho de vídeo (Fonte: Própria autora.

(c)), apresentamos também as questões para os momentos anterior (Fonte: Própria autora.

(a)) e posterior (Fonte: Própria autora.

(b)) à implementação do passo de resolução do problema. A cada questão apresentada, o vídeo pausa; em seguida, a questão é carregada e o estudante continua o vídeo após a submissão de sua resposta. Para gravar todos os exemplos em vídeo e capturar as respostas fornecidas pelos estudantes, utilizamos a ferramenta Camtasia²².

²² Camtasia. Disponível em: <www.techsmith.com/camtasia.html>. Acesso em: 28 jan. 2016.

Figura 4.5. Trecho de exemplo em vídeo, apresentando passo da implementação do problema (c) juntamente com as questões anterior (a) e posterior (b) à implementação do passo do problema.

1 of 1

1. Eu tenho alguma ideia para a implementação desse primeiro passo (ler o número que vai ser digitado pelo usuário)? Se sim, como eu poderia implementá-lo? Escreva aqui o código resultante e depois compare mentalmente com o código que será apresentado a seguir.

(a)

1 of 1

1. Explique o que faz o trecho de código que acabou de ser implementado, destacando porque as estruturas utilizadas ('int idade', 'printf' e 'scanf') são as mais adequadas para a sua implementação e como elas funcionam.

(b)

```

1  #include<stdio.h>
2
3  int main(void) {
4
5      int idade = 0;
6
7      printf("Digite uma idade: ");
8      scanf("%d", &idade);
9
10     system("PAUSE");
11     return 0;
12 }
13

```

(c)

Fonte: Própria autora.

Realizamos três quase-experimentos com a versão refinada da abordagem *Stepwise self-explanation*. Estes quase-experimentos serão detalhados a seguir no Capítulo 5.

4.3.3 Utilização da Stepwise Self-explanation

Para que um professor utilize a versão refinada da Stepwise Self-explanation é necessário que ele realize os seguintes passos:

- I. após definir os objetivos de aprendizagem que vão ser alcançados com a disciplina introdutória de programação, selecione os exemplos trabalhados que atendem a cada um destes objetivos de aprendizagem;
- II. estruture os exemplos trabalhados de forma que eles possuam:
 - a. um enunciado do problema;
 - b. um conjunto de entradas e saídas de funcionamento do código;
 - c. uma sequência de passos de solução para o problema implementado na linguagem de programação adotada na disciplina;

- d. um conjunto de entradas e saídas, diferente do anterior, para mostrar o funcionamento do código para o estudante, após a implementação do problema proposto.
- III. após estruturar os exemplos trabalhados, realize a gravação dos vídeos da seguinte maneira:
- a. apresente o enunciado do problema e o conjunto de entradas e saídas de funcionamento do código;
 - b. logo após, inicie a apresentação da sequência de passos de resolução do problema. Para cada um dos passos de resolução do problema:
 - i. antes de iniciar a implementação, descreva verbalmente a funcionalidade que será implementada com este passo de resolução do problema;
 - ii. em seguida, implemente o trecho de código relativo ao que foi descrito para o passo;
 - iii. por fim, explique verbalmente como funciona o trecho de código implementado.
 - c. ao finalizar a implementação do programa, apresente a execução do código com o segundo conjunto de entradas e saídas selecionado.
- IV. adicione as questões para auto-explicação ao longo do vídeo dos exemplos trabalhados nos seguintes momentos:
- a. após apresentar o enunciado do problema e o conjunto de entradas e saídas de funcionamento do código. Esta questão tem como objetivo fazer o estudante refletir sobre o propósito geral do programa que será implementado;
 - b. para cada um dos passos de resolução do problema:
 - i. após descrever verbalmente a funcionalidade que será implementada no passo e antes de apresentar a implementação do código para o referido passo. Esta questão tem como objetivo fazer o estudante refletir sobre como ele vai aplicar as estruturas de uma linguagem de programação para o passo em questão, antes do professor fornecer a sua solução;
 - ii. após apresentar a implementação do trecho de código do passo. Esta questão tem por objetivo refletir sobre a solução que foi

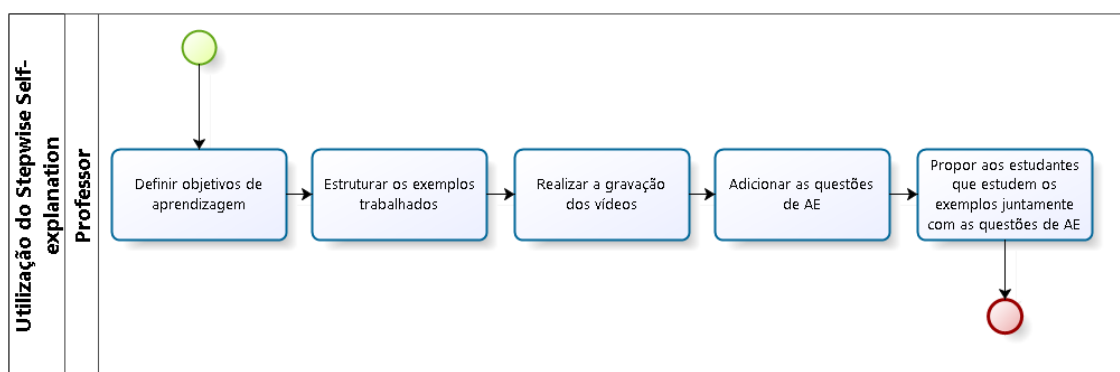
apresentada pelo professor, processando as informações que foram fornecidas por ele;

- c. após apresentar a execução do código com o segundo conjunto de entradas e saídas. Esta questão tem por objetivo fazer com que o estudante reflita sobre a execução do programa apresentado, calculando mentalmente a saída do código a partir de determinadas entradas.

- V. proponha aos estudantes que eles estudem os exemplos trabalhados enquanto o estão realizando as atividades de ensino e de aprendizagem propostas para alcançar os objetivos de aprendizagem definidos anteriormente.

Na Figura 4.6 temos uma versão diagramática dos passos que devem ser seguidos por um professor de uma disciplina de introdução a programação para que ele utilize a abordagem.

Figura 4.6. Workflow de utilização da Stepwise Self-explanation.



Fonte: Própria autora.

4.4 Implicações

Este quarto capítulo apresentou as duas versões da abordagem *Stepwise self-explanation*, a versão piloto e a versão refinada. A versão piloto da abordagem contou com exemplos trabalhados em vídeo da série *The Joy of Code* que ensinam a linguagem de programação Java utilizando a ferramenta *Greenfoot*. Estes exemplos trabalhados em vídeo contém apenas atividades de extensão seguidas por atividades de refinamento apresentadas pelo professor de programação. Para auxiliar os estudantes a estudar estes exemplos, depois de cada extensão e refinamento executado pelo professor no exemplo, os estudantes são convidados a se engajar em uma atividade de auto-explicação alinhada com a extensão e o refinamento que eles acabaram de assistir nos vídeos. Para promover a auto-explicação, definimos questões para que os estudantes pudessem refletir sobre o

objetivo do código sendo estendido (i.e., sobre a extensão) e sobre a aplicação dos conceitos de uma linguagem de programação para que ele alcançasse o objetivo definido (i.e., sobre o refinamento).

A versão refinada da Stepwise self-explanation, por sua vez, contou com exemplos trabalhados em vídeo que possuem atividades de extensão e refinamentos apresentadas pelo professor de programação. Estes exemplos trabalhados em vídeo foram organizados de acordo com a estrutura descrita a seguir. Em um primeiro momento, o professor narra o enunciado do problema que vai constituir-se na descrição do requisito que precisa ser alcançado pela especificação sendo estendida. Logo depois, é apresentada a especificação correspondente a este requisito, ou seja, o objetivo que precisa ser alcançado, através de um conjunto de entradas e saídas. Em um terceiro momento, o professor começa a construir o programa iterativa e incrementalmente a partir de especificações intermediárias, que correspondem a descrições concisas de objetivos intermediários que precisam ser alcançados até que se alcance a especificação principal. A cada especificação intermediária descrita, o professor executa uma atividade de refinamento através da construção de uma implementação que atende ao objetivo intermediário correspondente. Por fim, ao construir uma implementação que corresponda à especificação principal apresentada no início do exemplo, o professor mostra uma execução do código em funcionamento. Para auxiliar os estudantes no estudo destes exemplos em vídeo através da auto-explicação, definimos questões para refletir sobre o propósito geral do programa que será implementado, sobre a aplicação das estruturas de uma linguagem de programação, sobre a solução apresentada pelo professor e sobre a execução do programa apresentado. Ao final do capítulo, apresentamos um *workflow* de utilização da versão refinada da Stepwise Self-explanation de forma que outros professores de disciplinas de programação possam utilizá-la.

Para avaliar a eficácia das duas versões da abordagem proposta, realizamos um experimento piloto e três quase-experimentos²³ que serão descritos detalhadamente no próximo capítulo.

²³ Adotou-se a nomenclatura quase-experimentos para os estudos realizados para avaliar a versão refinada da abordagem porque, para tais estudos, os estudantes não foram divididos aleatoriamente nos diferentes grupos participantes, como aconteceu no estudo piloto. A nomenclatura quase-experimento é

5 Procedimentos metodológicos

Este quinto capítulo tem por objetivo apresentar os procedimentos metodológicos adotados na presente tese. Nela, adotamos a visão geral filosófica pragmática. Para os pesquisadores que optam por esta visão, o foco da pesquisa está em fornecer uma resposta ao problema em questão [CRESWELL; GARRETT, 2008]. Assim, pesquisadores pragmáticos optam por escolher métodos, técnicas e procedimentos de pesquisa que respondam a sua questão de pesquisa a partir de diferentes perspectivas e, por esta razão, normalmente eles lançam mão de métodos mistos de pesquisa [CRESWELL, 2013].

Dentro da esta visão pragmática, adotamos um projeto de pesquisa misto com diferentes casos. Primeiro, realizamos um experimento piloto, por meio do qual avaliamos a eficácia da primeira versão da abordagem *Stepwise Self-explanation*. Para este experimento piloto, adotamos uma análise quantitativa dos programas construídos pelos participantes do experimento. Em seguida, realizamos três quase-experimentos, por meio dos quais avaliamos a eficácia da segunda versão da abordagem *Stepwise Self-explanation*. Para estes quase-experimentos, adotamos uma análise qualitativa dos tipos de erros cometidos pelos participantes do experimento nos programas construídos por eles. Em seguida, realizamos uma análise quantitativa destes mesmos programas.

A seguir, na Seção 5.1 descreveremos o experimento piloto realizado para avaliar a primeira versão da abordagem. Logo após, na Seção 5.2 descreveremos os quase-experimentos realizados para avaliar a segunda versão da abordagem. Por fim, na Seção 5.2.4 apresentaremos as implicações decorrentes dos experimentos realizados.

5.1 Experimento piloto

O principal objetivo deste experimento piloto foi avaliar a eficácia da primeira versão da abordagem *Stepwise Self-explanation*. Para alcançar esse objetivo, comparamos a *Stepwise Self-explanation* com uma abordagem tradicional, onde os estudantes estudaram os vídeos sem as questões para o estímulo a auto-explicação e fizeram uso da sua própria maneira de estudar (i.e, auto-estudo). Além disso,

sugerida por Dancey e Reidy (2013) para aqueles estudos que não são estritamente experimentais, o que é o caso dos referidos estudos.

verificamos se existia uma relação entre as fases de aprendizado e de programação (i.e., resolução de problemas) no grupo instrucional, que estudou seguindo os preceitos da abordagem proposta. Assim, com este estudo, respondemos as seguintes questões:

[QP1] O aprendizado de programação através da AE de exemplos em vídeo apresenta melhores resultados quando comparado ao aprendizado de programação somente através de exemplos em vídeo?

[QP2] Existe alguma correlação entre as respostas às questões de auto-explicação que foram respondidas corretamente e o resultado obtido no pós-teste no grupo instrucional?

Neste experimento, os resultados foram mensurados em função das notas obtidas no pós-teste aplicado aos estudantes durante a fase de programação.

5.1.1 Participantes

Quartoze estudantes voluntários participaram do estudo. Estes voluntários foram divididos aleatoriamente em dois grupos. Durante a fase de estudo, o grupo instrucional (GI) estudou os exemplos em vídeo de acordo com a nossa abordagem enquanto o grupo controle (GC) estudou os exemplos em vídeo sem utilizar as questões de estímulo a auto-explicação e fizeram uso de sua própria maneira de estudar (i.e. auto-estudo). Dados mais detalhados a respeito dos voluntários são apresentados posteriormente na Seção 5.1.1.

5.1.2 Materiais instrucionais

Os materiais instrucionais utilizados durante este primeiro estudo foram:

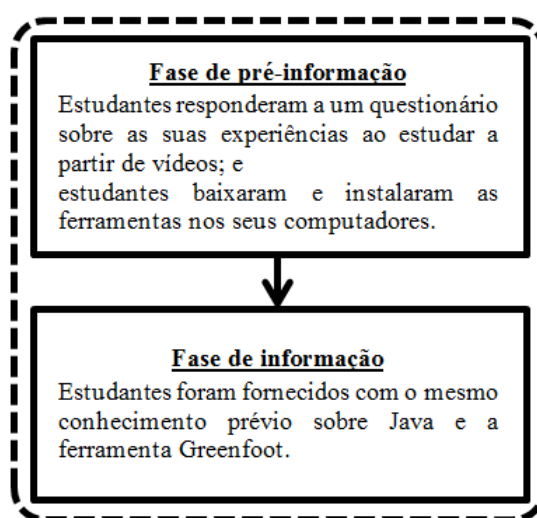
- (i) os três primeiros episódios dos exemplos em vídeo *The Joy of Code*²⁴. Estes vídeos foram divididos em diferentes seções que correspondiam a cada uma das sequências de atividades de extensão e refinamento;
- (ii) os cenários da Tartaruga e do Caranguejo do *The Joy of Code*;
- (iii) os dois exercícios práticos utilizados no estudo. Estes exercícios foram baseados nos exercícios do livro *Introduction to Programming with Greenfoot* [KÖLLING, 2009];
- (iv) as fichas contendo as questões para auto-explicação do GI.

²⁴ The joy of code. Disponível em: <<http://bit.ly/2phn7Lz>>. Acesso em: 18 abr. 2016.

5.1.3 Execução

Este estudo foi realizado em uma instituição de ensino que atua na educação de nível médio e técnico na cidade de Aarhus, Dinamarca, no segundo semestre de 2014. Ele consistiu em 5 fases: (i) fase de pré-informação; (ii) fase de informação; (iii) fase de aprendizado; (iv) fase de programação; (v) fase de pós-programação, realizado em dois diferentes dias. A primeira sessão foi realizada online através do uso de uma página web²⁵ e foi planejada para durar aproximadamente 1.5 hora. Ela compreendeu as fases pré-informação e informação do estudo, conforme exibido na Figura 5.1.

Figura 5.1. Sessão online.



Fonte: Própria autora.

A fase de pré-informação consistiu de 3 atividades. Primeiro, explicamos o experimento aos estudantes e coletamos o seu consentimento (ou dos seus pais, se eles fossem menores de 18 anos). O termo de consentimento livre e esclarecido (TCLE) foi fornecido previamente aos participantes e eles tiveram que trazer de volta para a sessão de sala de aula. Depois, coletamos dados demográficos dos participantes e dados sobre a sua experiência prévia quando estudando a partir de vídeos através de um questionário. Como não tínhamos disponível um laboratório de informática para execução do estudo, solicitamos que os estudantes usassem seus próprios computadores. Por esta razão, no final da fase de pré-informação pedimos que os estudantes instalassem a ferramenta de desenvolvimento *Greenfoot* e um software para que eles gravassem os seus monitores durante a fase de resolução dos exercícios.

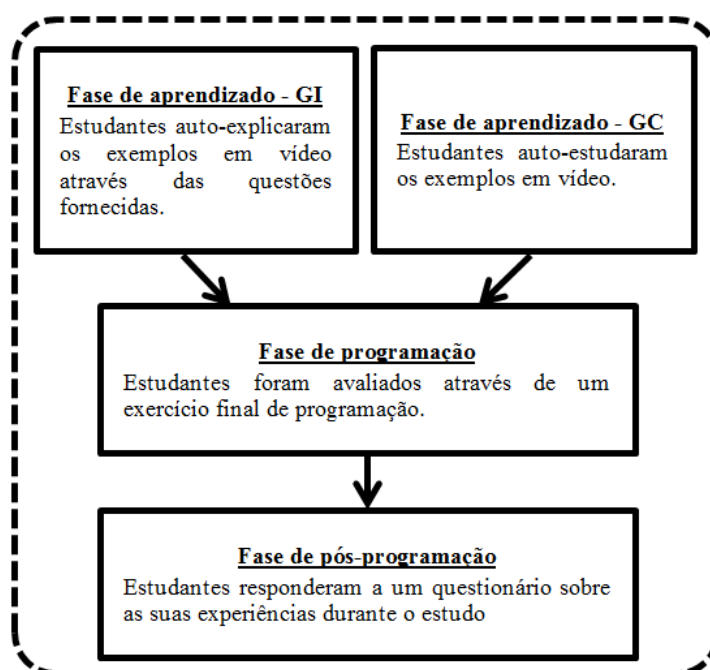
²⁵ Disponível em: <<http://bit.ly/VNKKb9>>. Acesso em: 18 nov. 2015.

A fase de informação consistiu de duas atividades. Primeiro, os participantes viram trechos dos episódios 1 e 3 dos exemplos em vídeo *The Joy of Code*. O trecho do episódio 1 apresentou alguns exemplos do que os estudantes podiam fazer usando a ferramenta de desenvolvimento *Greenfoot* e foi usada para motivá-los a participar do estudo. O trecho do episódio 3 apresentou alguns conceitos básicos em Java e, como os estudantes eram todos completamente iniciantes em programação, este vídeo foi usado para lhes oferecer o conhecimento prévio necessário para a sessão em sala de aula realizada durante o estudo. Os estudantes foram instruídos a não assistir a cada um dos vídeos mais do que duas vezes durante esta fase. Depois de assistir aos vídeos, os estudantes tinham que responder a um exercício prático inicial que foi utilizado como um exercício de aquecimento bem como para assegurar que eles podiam (i) utilizar o *Greenfoot*; (ii) gravar as suas telas de computadores enquanto estavam respondendo o exercício e, depois de finalizar, eles podiam fazer o *upload* do vídeo resultante em uma conta Dropbox.

A segunda sessão foi realizada na escola dos participantes e também durou 1.5 hora. Ela consistiu das fases de aprendizado (i.e., fase de estudo dos exemplos em vídeo), programação e pós-programação do estudo como mostrado na Figura 5.2. Nessa sessão, como dissemos anteriormente, os estudantes foram aleatoriamente divididos em 2 grupos, o grupo instrucional (GI) e o grupo controle (GC). Para conduzir as atividades em paralelo com os grupos, eles foram colocados em diferentes salas de aula. Além disso, para que as atividades fossem realizadas individualmente pelos estudantes, cada um deles usou o seu próprio computador e fone de ouvido. Para controlar as atividades, cada grupo tinha um mediador diferente, o GI foi conduzido pela autora do presente trabalho e o GC foi conduzido por um segundo pesquisador²⁶ treinado para mediação pela primeira mediadora.

²⁶ Pesquisador do grupo de pesquisa do *Center for Science Education*, na Universidade de Aarhus, Dinamarca, onde a autora da presente tese realizou o seu período de doutorado sanduíche no exterior.

Figura 5.2. Sessão em sala de aula.



Fonte: Própria autora.

Durante a fase de aprendizado, os estudantes tiveram que assistir a um conjunto de 4 trechos dos exemplos em vídeo *The Joy of Code*. Depois de assistir a cada trecho de vídeo, os estudantes de ambos os grupos tinham o mesmo tempo para estudá-los. Para coletar dados sobre como os estudantes no GI auto-explicavam cada seção do vídeo utilizando as questões correspondentes os alunos foram instruídos para estudar cada trecho respondendo por escrito as fichas de auto-explicação que estavam em folhas de papel. Da mesma forma, estudantes no GC tinham folhas de papel em branco enquanto estudavam cada trecho de vídeo que podiam ser utilizadas caso eles quisessem realizar quaisquer anotações. Durante o tempo de estudo, os estudantes de ambos os grupos podiam assistir a cada trecho de vídeo mais uma vez.

Para comparar o desempenho de ambos os grupos, durante a fase de programação os estudantes foram avaliados através de um pós-teste. A duração do pós-teste e o pós-teste propriamente dito eram idênticos para ambos os grupos. Para analisar o processo de construção dos programas feitos pelos estudantes, estes foram instruídos a gravar os seus monitores enquanto eles estavam respondendo o pós-teste e, em seguida, carregar o vídeo resultante na conta do Dropbox usada na sessão online.

Finalmente, no final da sessão de sala de aula, estudantes de ambos os grupos responderam um questionário final para avaliar a experiência que tiveram aprendendo

programação a partir de vídeos neste estudo. Além disso, o GI também avaliou a utilização das questões que os estimulavam a auto-explicar os vídeos enquanto estavam estudando programação a partir dos vídeos.

5.1.4 Análise dos dados

Ao final do estudo, os estudantes realizaram um pós-teste para avaliar o seu processo de resolução de problemas de programação após o estudo dos exemplos. O pós-teste consistiu de 5 questões onde os estudantes foram solicitados a construir programas similares àqueles que eles estudaram nos exemplos em vídeo. A cada questão no pós-teste foi atribuída uma nota de no máximo 8 pontos (5 para o programa construído e 3 para a justificativa da questão). Assim, os estudantes podiam atingir um total de 40 pontos. Além disso, a cada questão para o estímulo a auto-explicação foi atribuída uma nota de no máximo 1 ponto (resposta completamente errada – 0; resposta meio certa – 0,5; resposta certa – 1). Estes dados foram então utilizados para uma análise estatística que respondeu as perguntas de pesquisa QP1 e QP2.

5.2 Quase-experimentos

Nas Seções 5.2.1, 5.2.2 e 5.2.3 a seguir serão apresentados os três quase-experimentos realizados para avaliar a eficácia da versão refinada da abordagem *Stepwise self-explanation*.

5.2.1 Primeiro quase-experimento

Neste primeiro quase-experimento, tivemos dois grupos instrucionais, um onde os exemplos em vídeo foram estudados através de perguntas abertas (como no exemplo apresentado no Quadro 4.5) e outro onde os exemplos em vídeo foram estudados juntamente com perguntas de múltipla escolha. Além disso, tivemos um grupo controle, no qual os estudantes estudaram apenas os exemplos em vídeo a sua própria maneira (i.e., auto-estudo). Sendo assim, para alcançar o objetivo mencionado anteriormente, comparamos as duas versões da nossa abordagem com a abordagem de auto-estudo utilizada pelo grupo controle. Além disso, verificamos se existiu uma relação entre as fases de aprendizado e de programação (i.e., resolução de problemas) nos grupos instrucionais. Portanto, com este estudo, respondemos as seguintes questões:

- [QP1] O aprendizado de programação através da AE de exemplos em vídeo (i.e., dos grupos instrucionais) apresenta melhores resultados quando

comparado ao aprendizado de programação somente através de exemplos em vídeo (i.e., do grupo controle)?

[QP2] Existe alguma diferença em relação ao aprendizado de programação e os resultados obtidos entre os diferentes grupos instrucionais?

[QP3] Existe alguma correlação entre as respostas às questões de auto-explicação, o tempo de estudo dos exemplos em vídeo juntamente com as questões de auto-explicação e os resultados obtidos no pós-teste em cada um dos grupos instrucionais?

5.2.1.1 *Participantes*

Os participantes deste quase-experimento eram estudantes matriculados no primeiro semestre de 2015 em uma disciplina de Introdução a Programação do curso de Engenharia da Computação em uma instituição pública de ensino superior no Estado de Pernambuco. A disciplina contava com 42 estudantes matriculados, que foram convidados pessoalmente pela autora desta tese em algumas aulas da disciplina. Do total de estudantes da disciplina, tivemos 8 participantes voluntários²⁷ em todas as etapas do experimento. Estes participantes foram organizados em três diferentes grupos, o grupo controle (GC) e dois grupos instrucionais (GI1 e GI2), de acordo com a disponibilidade dos estudantes. O GC contou com 3 participantes, o GI1 contou com 2 participantes e o GI2 contou com 3 participantes. Durante a execução do experimento, o GC fez uso somente dos exemplos trabalhados em vídeo durante o seu momento de estudo. O GI1 e GI2, por sua vez, utilizaram os mesmos exemplos trabalhados em vídeo em conjunto com as perguntas de AE desenvolvidas para estes vídeos. As questões de AE utilizadas por estes grupos foram abertas e de múltipla escolha, respectivamente.

5.2.1.2 *Materiais instrucionais*

Os materiais instrucionais utilizados durante este primeiro estudo foram:

- (i) 2 exemplos trabalhados em vídeo produzidos de acordo com a estrutura do FSI, conforme apresentado na Seção 4.3.1. Estes vídeos foram gravados pela autora deste trabalho;

²⁷ Na verdade, o grupo de participantes total ao longo do experimento foi de 18 participantes. No entanto, somente 8 destes participantes participaram dos 2 encontros para o estudo dos exemplos em vídeo e entregaram todos os materiais que foram coletados durante estes encontros.

- (ii) um pré-teste e um pós-teste para cada um dos exemplos utilizados no estudo. Cada pré e pós-teste continha apenas um problema para verificar a capacidade de aplicação dos conhecimentos de programação adquiridos pelos estudantes antes e após as intervenções. Os problemas abordados nos exemplos e nos pré e pós-testes foram baseados em problemas da Olimpíada Brasileira de Informática (OBI)²⁸; e
- (iii) as perguntas para estímulo a AE que serviram de *scaffolding* para os dois grupos instrucionais. Para o GI1, as perguntas foram abertas e para o GI2, as perguntas foram de múltipla escolha.

5.2.1.3 Execução

Este experimento foi realizado na mesma instituição pública de ensino superior a qual nos referimos na Seção 5.2.1.1. A disciplina de programação na qual os participantes deste quase-experimento estavam matriculados foi ministrada utilizando-se C como linguagem de programação e o CodeBlocks como ferramenta de desenvolvimento. A maioria dos estudantes matriculados nesta disciplina estava cursando-a pela segunda vez e, por este motivo, o professor ministrou a disciplina exclusivamente através de resolução de problemas, sem aula expositiva para apresentação do conteúdo declarativo. Ao longo da disciplina, o professor analisou formativamente a evolução dos estudantes e, a cada aula, ele selecionou e/ou produziu um novo conjunto de exercícios adequados ao nível de dificuldade que os estudantes apresentaram. Enquanto resolviam os problemas durante as aulas, os estudantes podiam discutí-los livremente com seus pares e também podiam tirar dúvidas com o professor ou com um dos monitores da disciplina.

Assim, resolvemos realizar o experimento oferecendo aulas extras para as quais os estudantes foram convidados pessoalmente a participar pela autora desta tese. Nestas aulas extras, os estudantes que se voluntariaram tiveram a oportunidade de participar de um momento adicional de estudo para disciplina de programação através do estudo de exemplos trabalhados em vídeo durante os encontros. Além disso, estes estudantes ganharam uma pontuação adicional na disciplina pela participação nestas aulas. Os estudantes que se voluntariaram foram organizados em três diferentes grupos, sendo um grupo controle e dois grupos instrucionais. Como a autora deste trabalho conduziu a

²⁸ OBI. Disponível em < <http://olimpiada.ic.unicamp.br/> >. Último acesso: 23 nov. 2015.

realização dos experimentos nos três grupos, os estudantes foram alocados nos diferentes grupos de acordo com o dia e o horário que eles tinham disponível.

Para cada um dos três grupos, tivemos duas aulas extras, com duração de 2 horas cada uma. Estas aulas extras ocorreram nos laboratórios de informática da referida instituição de ensino. Como os estudantes desta turma tinham apenas aulas com resolução de problemas, eles praticavam em suas aulas regulares os problemas básicos normalmente abordados em disciplinas introdutórias de programação, tais como o exemplo apresentado no Quadro 4.5. Sendo assim, este exemplo trabalhado ou similar poderia ser considerado simples para estes estudantes. Por esta razão, baseado nos tópicos que haviam sido abordados e nos exercícios que haviam sido propostos aos estudantes pelo professor da disciplina até o início das aulas extras, resolvemos adotar problemas que fossem considerados problemas desafios para os estudantes voluntários.

Sendo assim, os tópicos abordados e objetivos de aprendizagem alcançados com as aulas extras ministradas aos estudantes neste primeiro experimento estão listados no Quadro 5.1. Na linguagem de programação C, não possuímos um tipo de dado string, portanto nesta linguagem strings são representadas como arrays do tipo char [MIZRAHI, 2008]. Como gostaríamos de abordar o assunto strings nas aulas extras, os tópicos cobertos com as duas aulas extras envolveram arrays e strings. Os tópicos e os objetivos de aprendizagem apresentados no Quadro 5.1 fazem parte da disciplina Estruturas de dados fundamentais de programação da área de conhecimento de Fundamentos de desenvolvimento de software do Currículo da ACM [J. T. F. ON COMPUTING CURRICULA, 2013]. Além disso, eles estão compreendidos nos tópicos e objetivos de aprendizagem cobertos pela disciplina introdutória de programação da turma deste experimento.

Quadro 5.1. Tópicos e objetivos de aprendizagem ministrados nas aulas extras do primeiro experimento.

Aula	Tópicos	Objetivos de aprendizagem
1	Arrays unidimensionais e strings	Escrever programas que usem as seguintes estruturas de dados: arrays [unidimensionais], strings (...)
2	Arrays multidimensionais e strings	Escrever programas que usem as seguintes estruturas de dados: arrays [multidimensionais], strings (...)

Fonte: própria autora.

Cada uma das aulas extras compreendeu as seguintes atividades: (i) realização de um pré-teste; (ii) momento de estudo de um exemplo trabalhado em vídeo; e (iii) realização de um pós-teste. Como dito anteriormente, cada pré e pós-teste continha

apenas um problema para verificar a capacidade de aplicação dos conhecimentos de programação adquiridos pelos estudantes antes e depois do estudo dos exemplos. Os problemas abordados nestes testes foram todos alinhados com os tópicos e os objetivos de aprendizagem definidos Quadro 5.1. Durante a realização do pré-teste e pós-teste, os estudantes podiam utilizar exclusivamente a folha de papel contendo o problema que eles precisavam resolver e a ferramenta *CodeBlocks*.

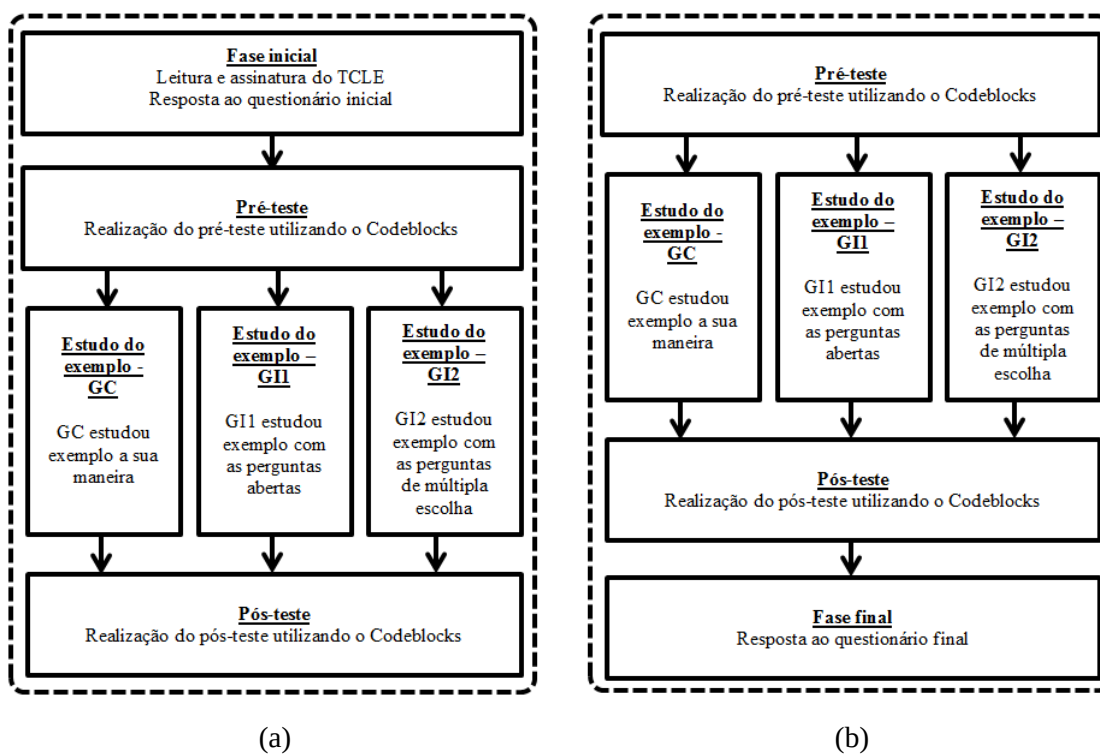
Durante o momento de estudo, os estudantes estudaram individualmente um exemplo trabalhado em vídeo que seguiu a mesma estrutura do exemplo apresentado no Quadro 4.5. Antes de começar este momento de estudo propriamente dito, os estudantes receberam instruções de como proceder para estudar o exemplo. Todos os grupos receberam orientação de que deveriam estudar os exemplos como se eles fossem realizar uma prova após este momento de estudo. Além disso, o GC foi orientado a estudar os exemplos utilizando a sua própria maneira de estudar. Já o GI1 e o GI2 receberam orientações de que, durante a execução do vídeo, perguntas para que eles refletissem a respeito da construção do exemplo apareceriam, que eles deveriam utilizá-las como se estivessem explicando para eles próprios os passos que estavam sendo executados no exemplo, lendo as perguntas atentamente e procurando respondê-las²⁹ mesmo que eles não tivessem certeza das respostas. Os estudantes do GI1 e GI2 foram orientados também a reproduzir o código que estava sendo construído pelo professor, como forma de auxiliar os seus processos de reflexão. Por fim, eles receberam como última instrução que se quisessem fazer algum tipo de anotação, teriam folhas de papel disponíveis.

Aliado a isso, tivemos como atividade da primeira aula extra, conforme descrito na Figura 5.3 (a), a leitura e assinatura do TCLE e para realização de um questionário inicial para coleta de informações demográficas e da experiência prévia dos estudantes ao estudar para a disciplina de programação. Tivemos também como atividade da segunda aula extra, conforme descrito na Figura 5.3 (b), a realização de um questionário final para avaliar a experiência dos estudantes ao estudar para a disciplina de programação a partir dos exemplos em vídeo nos três diferentes grupos. Além disso, os

²⁹ O GI1, para o qual as perguntas eram abertas, deveria respondê-las por escrito. O GI2, para o qual as perguntas eram de múltipla escolha, deveria respondê-las selecionando uma das opções fornecidas.

grupos instrucionais avaliaram a utilização das perguntas que os estimulavam a auto-explicar os exemplos trabalhados em vídeo.

Figura 5.3. Primeira aula extra (a). Segunda aula extra (b).



Fonte: própria autora.

5.2.2 Segundo quase-experimento

Neste segundo quase-experimento, tivemos apenas um grupo instrucional que estudou os exemplos em vídeo através de perguntas abertas. Neste caso, como temos apenas o grupo instrucional, analisamos a evolução dos estudantes a partir do estudo dos exemplos trabalhados em vídeo. Além disso, verificamos também a existência ou não de uma relação entre as fases de aprendizado e de programação para estes estudantes. Portanto, com este estudo, respondemos as seguintes questões:

- [QP1] Como foi a evolução do aprendizado de programação dos estudantes a partir da AE de exemplos trabalhados em vídeo ao longo dos momentos de estudo durante a execução deste experimento?
- [QP2] Existe alguma correlação entre as respostas às questões de auto-explicação, o tempo de estudo dos exemplos em vídeos juntamente com as questões de auto-explicação e os resultados obtidos no pós-teste em cada um dos grupos instrucionais?

5.2.2.1 Participantes

Os participantes deste quase-experimento eram estudantes matriculados no primeiro semestre de 2015 em uma disciplina de Introdução a Programação do curso de Ciência da Computação em uma segunda instituição pública de ensino superior no Estado de Pernambuco. A disciplina contava com 19 estudantes matriculados, que foram convidados pessoalmente pela autora desta tese em algumas aulas da disciplina. Do total de estudantes da disciplina, tivemos 2 participantes voluntários³⁰ em todas as etapas do experimento. Estes participantes foram organizados em um único grupo instrucional GI. Durante a execução do experimento, este grupo utilizou 4 diferentes exemplos trabalhados em vídeo em conjunto com as perguntas de AE desenvolvidas para estes exemplos. Os exemplos utilizados por este grupo foram diferentes dos exemplos utilizados pelos grupos do primeiro experimento (Seção 5.2.1). Além disso, as questões de AE utilizadas por este grupo foram abertas.

5.2.2.2 Materiais instrucionais

Os materiais instrucionais utilizados durante este segundo experimento foram:

- (i) 4 exemplos trabalhados em vídeo estruturados de acordo com o FSI;
- (ii) um pré-teste e um pós-teste para cada um dos exemplos utilizados no estudo. Cada pré e pós-teste continha apenas um problema para verificar a capacidade de aplicação dos conhecimentos de programação adquiridos pelos estudantes antes e após as intervenções. Os problemas abordados nos exemplos e nos pré e pós-testes foram baseados em problemas dos livros Fundamentos de programação de computadores [ASCENCIO; DE CAMPOS, 2008] e Treinamento em Linguagem C [MIZRAHI, 2008]; e
- (iii) perguntas abertas para estímulo a auto-explicação que serviram de *scaffolding* e reflexão para o grupo instrucional.

5.2.2.3 Execução

Este experimento foi realizado na mesma instituição pública de ensino superior a qual nos referimos na Seção 5.2.2.1. A disciplina de programação na qual os

³⁰ Na verdade, o grupo de participantes total foi 5. No entanto, somente 2 participantes participaram dos 4 aulas extras e entregaram todos os materiais que foram coletados durante estas aulas.

participantes deste quase-experimento estavam matriculados foi ministrada utilizando-se C como linguagem de programação e o CodeBlocks como ferramenta de desenvolvimento. A maioria dos estudantes matriculados nesta disciplina estava cursando-a pelo menos pela terceira vez. O professor que lecionou a disciplina ministrou-a da maneira tradicional, com aulas expositivas para apresentação do conteúdo declarativo e com a apresentação de alguns exemplos estáticos (i.e., programas já prontos nos slides utilizados como material instrucional da aula) para apresentação do conteúdo procedural. Ao longo da disciplina, o professor também disponibilizou alguns exercícios que os estudantes resolvessem utilizando o sistema de submissão e correção automática de exercícios run.codes³¹.

Dentro deste âmbito, realizamos o experimento oferecendo aulas extras para as quais os estudantes foram convidados a participar pessoalmente pela autora deste trabalho. Nestas aulas extras, os estudantes voluntários participaram de um momento adicional de estudo para a disciplina de programação através do estudo de exemplos em vídeo e, além disso, ganharam uma pontuação adicional na lista de exercícios pela participação. Conforme dito anteriormente, os estudantes voluntários neste experimento foram organizados em um único grupo instrucional.

Neste experimento, tivemos 4 aulas extras, onde cada uma delas durou aproximadamente 2 horas. Elas ocorreram nos laboratórios de informática da referida instituição de ensino em dias e horários de acordo com a disponibilidade dos estudantes. Como os estudantes desta turma estavam repetindo a disciplina pelo menos pela segunda vez, consideramos que eles eram estudantes com sérias dificuldades de aprendizagem na disciplina de programação. Em função disso, resolvemos adotar exemplos e problemas que não tinham sido abordados ainda em sala de aula, mas que iriam revisar alguns dos tópicos já abordados pelo professor afim de reforçar o conhecimento dos estudantes. Por esta razão, os exemplos e problemas selecionados para este experimento foram mais simples do que os adotados no primeiro quase-experimento (Seção 5.2.1), sendo um deles apresentado no Quadro 4.5.

Sendo assim, os tópicos abordados e objetivos de aprendizagem alcançados com as aulas extras ministradas aos estudantes neste segundo experimento estão listados no Quadro 5.2. Estes objetivos de aprendizagem fazem parte das disciplinas Conceitos

³¹ run.codes. Disponível em: < <https://run.codes/> >. Último acesso em: 01 dezembro 2015.

fundamentais de programação e Estruturas de dados fundamentais de programação, ambas da área de conhecimento de Fundamentos de desenvolvimento de software do Currículo da ACM [J. T. F. ON COMPUTING CURRICULA, 2013]. Além disso, eles estão compreendidos nos tópicos e objetivos de aprendizagem cobertos pela disciplina introdutória de programação da turma deste experimento.

Quadro 5.2. Tópicos e objetivos de aprendizagem ministrados nas aulas extras do segundo experimento.

Aula	Tópicos	Objetivos de aprendizagem
1	Estruturas condicionais e iterativas	(...) Implementar (...) programas que utilizem cada um dos seguintes construtos fundamentais de programação: estruturas condicionais e iterativas.
2		
3	Arrays unidimensionais	Escrever programas que usem cada uma das seguintes estruturas de dados: arrays [unidimensionais e multidimensionais] (...).
4	Arrays multidimensionais	

Fonte: própria autora.

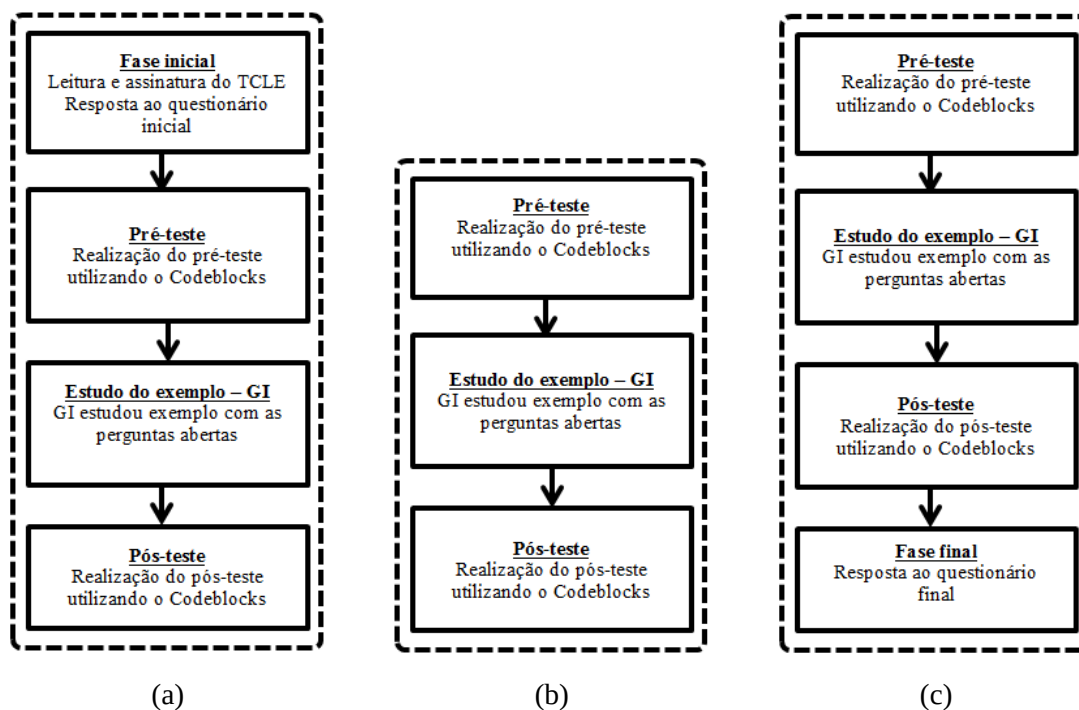
As aulas extras neste segundo experimento tiveram uma estrutura similar às aulas do primeiro experimento, cada uma delas compreendendo as seguintes atividades: (i) realização de um pré-teste; (ii) momento de estudo de um exemplo trabalhado em vídeo; e (iii) realização de um pós-teste, conforme apresentado na Figura 5.4. Como dito anteriormente, cada pré e pós-teste continha apenas um problema para verificar a capacidade de aplicação dos conhecimentos de programação adquiridos pelos estudantes antes e depois do estudo dos exemplos. Os problemas abordados nestes testes foram todos alinhados com os tópicos e os objetivos de aprendizagem definidos no Quadro 5.2. Durante a realização dos pré e pós-teste, os estudantes podiam utilizar exclusivamente a folha de papel contendo o problema que eles precisavam resolver e a ferramenta de desenvolvimento CodeBlocks.

Durante o momento de estudo, os estudantes estudavam individualmente um exemplo trabalhado em vídeo que seguiu a mesma estrutura do exemplo apresentado no Quadro 4.5 (sendo um deles o próprio exemplo apresentado neste quadro). Antes de começar este momento de estudo propriamente dito, os estudantes receberam instruções de que deveriam estudar os exemplos como se eles fossem realizar uma prova após este momento de estudo. Além disso, os estudantes receberam orientações de que durante a execução do vídeo perguntas para que eles refletissem a respeito da construção do exemplo apareceriam, que eles deveriam utilizá-las como se estivessem explicando para eles próprios os passos que estavam sendo executados no exemplo, lendo-as atentamente e procurando respondê-las mesmo que eles não tivessem certeza das respostas. Os estudantes do GI1 e GI2 foram orientados também a reproduzir o código

que estava sendo construído pelo professor, como forma de auxiliar os seus processos de reflexão. Por fim, eles receberam como instruções que se quisessem fazer algum tipo de anotação, teriam folhas de papel disponíveis que poderiam ser solicitadas.

Aliado a isso, tivemos como atividade do primeiro encontro, conforme descrito na Figura 5.4 (a), uma fase inicial para leitura e assinatura do TCLE e para realização de um questionário inicial para coleta de informações demográficas e da experiência prévia dos estudantes ao estudar para a disciplina de programação. Tivemos também como atividade do último encontro, conforme descrito na Figura 5.4 (c), a realização de um questionário final para avaliar a experiência dos estudantes ao estudar a partir dos exemplos em vídeo e sobre a utilização das perguntas que os estimulavam a refletir e os orientava a auto-explicar os exemplos trabalhados em vídeo.

Figura 5.4. Primeira aula extra (a). Segunda e terceira aulas extras (b). Quarta aula extra (c).



Fonte: própria autora.

5.2.3 Terceiro quase-experimento

Neste quase-experimento, tivemos um grupo instrucional, que estudou os exemplos em vídeo através de perguntas de múltipla escolha. Além disso, tivemos um grupo controle, no qual os estudantes estudaram apenas os exemplos em vídeo a sua própria maneira (i.e., auto-estudo). Sendo assim, para alcançar o objetivo mencionado anteriormente, comparamos uma versão da nossa abordagem com a abordagem utilizada com o grupo controle. Além disso, verificamos se existiu uma relação entre as fases de

aprendizado e de programação no grupo instrucional. Portanto, com este estudo, respondemos as seguintes questões:

- [QP1] O aprendizado de programação através da AE de exemplos em vídeo (i.e., do grupo instrucional) apresenta melhores resultados quando comparado ao aprendizado de programação somente através de exemplos em vídeo (i.e., do grupo controle)?
- [QP2] Existe alguma correlação entre as respostas às questões de auto-explicação, o tempo de estudo dos exemplos em vídeos juntamente com as questões de auto-explicação e os resultados obtidos no pós-teste em cada um dos grupos instrucionais?

5.2.3.1 Participantes

Os participantes deste quase-experimento eram estudantes matriculados no primeiro semestre de 2015 em uma instituição pública de ensino médio, técnico e superior no Estado de Pernambuco. Este quase-experimento foi realizado como parte de uma semana de tecnologia promovida pela instituição. Sendo assim, o convite para a participação no experimento ocorreu pessoalmente pela autora deste trabalho antes da e durante a realização do evento.

O experimento contou com 12 participantes voluntários em todas as etapas do experimento. Estes participantes foram organizados em dois diferentes grupos, o grupo controle (GC) e o grupo instrucional (GI), de acordo com a disponibilidade relatada por eles para a participação no experimento. Durante a execução do experimento, o GC fez uso de 4 exemplos trabalhados em vídeos, enquanto que o GI fez uso dos mesmos exemplos em vídeos em conjunto com as questões de AE de múltipla escolha desenvolvidas para estes vídeos.

5.2.3.2 Materiais instrucionais

Os materiais instrucionais produzidos para este terceiro experimento foram:

- (i) apresentação utilizada para guiar a aula expositiva;
- (ii) um conjunto de 4 exemplos trabalhados em vídeo que exibiam a aplicação dos conceitos apresentados na aula expositiva estruturados de acordo o FSI. Estes vídeos foram gravados pela autora deste trabalho;

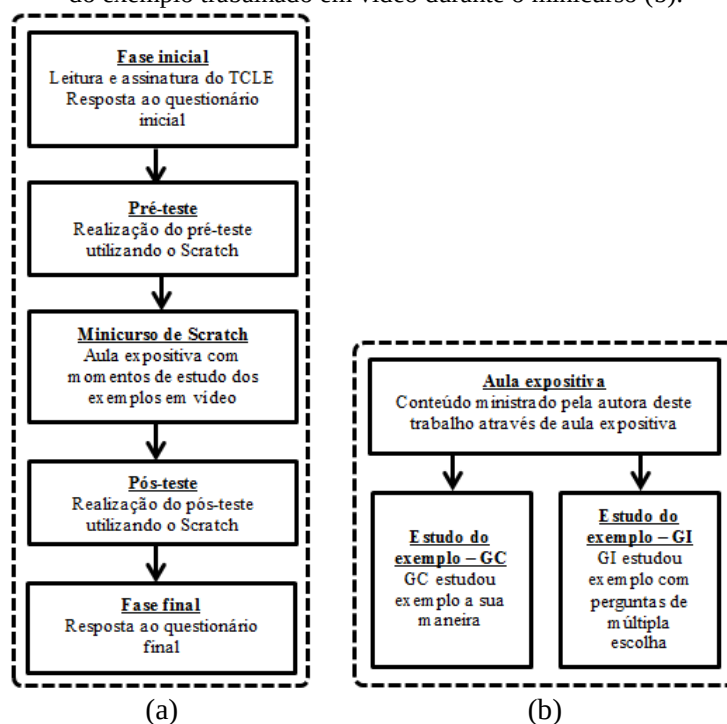
- (iii) pré e pós-testes para avaliar o desempenho dos alunos antes e após o minicurso ministrado. Cada pré e pós-teste continha 7 questões para verificar a capacidade de aplicação dos conhecimentos de programação adquiridos pelos estudantes antes e após o minicurso. Todo o material produzido foi baseado no *Curriculum Guide do Scratch* [BRENNAN; CHUNG; HAWSON, 2011];
- (iv) as perguntas para estímulo a AE que serviram de *scaffolding* e reflexão para o GI. No caso deste quase-experimento, as perguntas foram de múltipla escolha.

5.2.3.3 Execução

Este quase-experimento foi executado em dois dias distintos (um para o GC e outro para o GI) no formato de um minicurso com a duração de aproximadamente 4 horas durante a realização de uma semana de tecnologia promovida pela instituição. O experimento contou com estudantes matriculados em uma disciplina introdutória de programação e com estudantes que não tinham nenhum tipo de experiência em programação. Os estudantes matriculados na disciplina introdutória de programação estavam cursando-a pela primeira vez e, segundo os seus próprios relatos, as dificuldades eram imensas. Para abarcar estes dois tipos de estudante, o minicurso foi planejado de maneira a ser um momento de reforço para aqueles estudantes que já estavam cursando a disciplina introdutória de programação pela primeira vez e também um momento de aprendizado inicial para aqueles que não tinham nenhum tipo de experiência nesta disciplina. Portanto, resolvemos utilizar o Scratch como ferramenta de desenvolvimento e linguagem de programação, uma vez que seria relativamente simples e rápido aplicá-lo em um minicurso introdutório de programação, conforme descrito por Aureliano e Tedesco (2012).

Dentro deste âmbito, o experimento aconteceu no laboratório de informática da referida instituição de ensino e consistiu em uma sequência de 5 atividades conforme apresentado na Figura 5.5 (a): (i) fase inicial; (ii) realização de um pré-teste; (iii) execução de um minicurso; (iv) realização de um pós-teste; e (v) fase final.

Figura 5.5. Estrutura geral do terceiro experimento (a). Momentos de aula expositiva seguidos por estudo do exemplo trabalhado em vídeo durante o minicurso (b).



Fonte: própria autora.

Nesta figura, notamos que tivemos como atividades da fase inicial a leitura e assinatura do TCLE³² e o preenchimento de um questionário inicial para coleta de informações demográficas dos estudantes, além de informações sobre os seus hábitos de estudo e sua experiência prévia com vídeos para estudar. Para aqueles estudantes cursando a disciplina de programação, as respostas ao questionário inicial foram relativas à própria disciplina. Os estudantes que não estavam cursando a disciplina de programação, responderam o questionário inicial sobre as disciplinas em geral que eles cursavam. O pré-teste foi utilizado para avaliar o conhecimento prévio que os estudantes possuíam em programação antes do minicurso ser ministrado. O minicurso, por sua vez, combinou períodos de aula expositiva seguidos por períodos de estudo individual dos exemplos em vídeo, conforme detalhado na Figura 5.5 (b). Durante os períodos de aula expositiva foi abordado o conteúdo declarativo e durante os períodos de estudo individual foi abordado o conteúdo procedural do minicurso. Os tópicos e objetivos de aprendizagem abordados no minicurso são apresentados no Quadro 5.3. Este conjunto de tópicos e objetivos de aprendizagem seguem a definição da disciplina

³² Para aqueles estudantes menores de 18 anos, o TCLE foi entregue antes da realização do experimento e eles trouxeram preenchidos e assinados pelos pais no dia do experimento.

Conceitos fundamentais de programação definida no Currículo da ACM [J. T. F. ON COMPUTING CURRICULA, 2013].

Quadro 5.3. Tópicos e objetivos de aprendizagem ministrados no minicurso do terceiro experimento.

Tópicos	Objetivos de aprendizagem
Introdução a programação com o <i>Scratch</i> Estrutura de repetição - sempre	(...) Implementar (...) programas que usem cada um dos seguintes construtos fundamentais de programação: estruturas condicionais e iterativas (...)
Estruturas de decisão – se	
Estruturas de decisão – espere até	
Variáveis	Escrever programas que utilizem os tipos primitivos de dados: números (...)

Fonte: própria autora.

Como dissemos anteriormente, os estudantes foram divididos no GC e no GI. Portanto, durante os períodos de estudo individual, os estudantes do GC estudavam apenas os exemplos em vídeo, enquanto os estudantes do GI estudavam os exemplos em vídeo auxiliados pelas perguntas de estímulo a auto-explicação. Por conta do tempo reduzido para a execução do minicurso com o estudo dos exemplos em vídeo, decidimos que neste experimento utilizaríamos apenas perguntas de auto-explicação de múltipla escolha.

Antes de começar estes momentos de estudo, os estudantes recebiam instruções de como deveriam proceder para estudar o exemplo. Todos os grupos receberam orientação de que deveriam estudar os exemplos como se eles fossem realizar uma prova após este momento. Além disso, o GC foi orientado a estudar os exemplos utilizando a sua própria maneira de estudar. Já o GI recebeu orientações de que durante a execução do vídeo perguntas para que eles refletissem a respeito da construção do exemplo apareceriam, que os estudantes deveriam utilizá-las como se estivessem explicando para eles os passos que estavam sendo executados no exemplo, lendo-as atentamente e procurando respondê-las mesmo que eles não tivessem certeza das respostas. Os estudantes do GI foram orientados também a reproduzir o código que estava sendo construído pelo professor, como forma de auxiliar os seus processos de reflexão. Por fim, ambos os grupos receberam como instruções que se quisessem fazer algum tipo de anotação, teriam folhas de papel disponíveis que eles poderiam ser solicitadas.

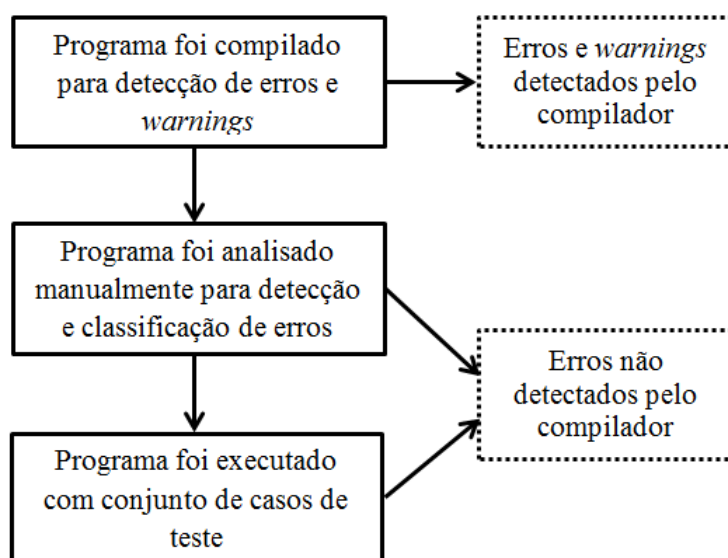
Logo depois do minicurso, tivemos a realização de um pós-teste, utilizado para avaliar o conhecimento adquirido pelos estudantes. Para concluir, tivemos como última atividade a fase final do experimento que contou com o preenchimento de um

questionário final que teve por objetivo coletar a opinião dos participantes a respeito do material instrucional utilizado por eles. Os estudantes de ambos os grupos deram suas opiniões sobre os exemplos em vídeo. Além disso, os estudantes do GI forneceram opinião sobre as perguntas de múltipla escolha utilizadas durante os momentos de estudo dos exemplos em vídeo.

5.2.4 Análise dos dados

Para responder as questões de pesquisa dos três quase-experimentos, examinamos os programas construídos pelos estudantes como respostas aos pré e pós-testes por meio de uma abordagem mista. Para todos os quase-experimentos realizados, cada um dos programas construídos pelos estudantes nos pré e pós-testes passou por uma análise realizada em três etapas conforme apresentado na Figura 5.6.

Figura 5.6. Etapas de análise dos programas no pré e pós-testes.



Fonte: própria autora.

Na primeira etapa, cada programa foi compilado para detecção de erros gerados pelo compilador, ou seja, para a identificação de erros sintáticos e avisos (*warnings*) detectados pelo *CodeBlocks*. Na segunda etapa, cada programa foi analisado manualmente para detecção dos erros não sintáticos. Na terceira etapa, os programas passaram por um conjunto de casos de teste para detecção de problemas manualmente. Para a análise realizada nas segunda e terceira etapas, para cada pré e pós-teste definimos um conjunto de metas intermediárias que deveriam ser alcançadas para que o programa implementado atendesse a especificação e ao requisito definido para o

problema do teste correspondente. Por fim, os erros encontrados foram contabilizados e utilizados posteriormente nos resultados. Para ilustrar a análise realizada, apresentamos no Quadro 5.4, o pós-teste utilizado na segunda aula extra do primeiro quase-experimento.

Quadro 5.4. Pós-teste utilizado na segunda aula extra do primeiro quase-experimento.

Enunciado do problema	Construa um programa que é uma versão simplificada do jogo Sonic®. O programa deve receber um valor inteiro que representa as dimensões de um tabuleiro quadrado e o próprio tabuleiro que é uma matriz de caracteres, onde: • O caracter ‘.’ representa um espaço vazio; • O caracter ‘o’ representa um anel de ouro; e • O caracter ‘E’ representa um espinho. Considere que: • a dimensão do tabuleiro poderá ser até 100 x 100 caracteres; • que o Sonic começa no canto superior esquerdo do tabuleiro; • ele percorre toda a linha da esquerda para a direita até chegar ao lado direito do tabuleiro; • depois ele desce uma posição e percorre toda linha, desta vez da direita para a esquerda; e • vai repetindo esses dois últimos passos até acabar de percorrer o tabuleiro. O Sonic percorre o tabuleiro por completo e a medida que vai percorrendo ele vai coletando os anéis de ouro. Quando ele encontra um espinho, ele perde todos os anéis de ouro coletados. O programa deve imprimir como resultado a quantidade máxima de anéis de ouro que o Sonic pode coletar antes de encontrar um fantasma.	
Especificação do problema	O programa deverá funcionar da seguinte maneira:	
	Entrada	Saída
	4 .oo. ..oo ..oo Eooo	9
Metas intermediárias identificadas	3 .Eo ooE oo.	4
	• Entrada 1: Lê a dimensão da matriz; • Entrada 2: Lê a matriz propriamente dita; • Executa laço: Percorre a matriz como descrito no problema; • Acumulador: Contador que acumula o valor dos anéis de ouro encontrados; • Reinicia acumulador: Quando encontra um espinho no caminho, o contador de anéis de ouro é reiniciado; • Atualiza maior: Atualiza o maior valor encontrado para a sequência de anéis de ouro até o momento corrente; • Saída: Imprime o maior valor encontrado para a sequência de anéis de ouro.	
	Possível codificação <pre>#include<stdio.h></pre>	

para as metas	<pre> int main() { int n, i, j, aneis = 0, maiorAneis = 0; char tab[102][102], c; scanf("%d", &n); fflush(stdin); for(i = 0; i < n; i++) { gets(tab[i]); } for(i = 0; i < n; i++) { if(i % 2 == 0) { for(j = 0; j < n; j++) { c = tab[i][j]; if(c == 'o') { aneis = aneis + 1; } else if(c == 'E') { if(aneis > maiorAneis) { maiorAneis = aneis; } aneis = 0; } } } else { for(j = n-1; j >= 0; j--) { c = tab[i][j]; if(c == 'o') { aneis = aneis + 1; } else if(c == 'E') { if(aneis > maiorAneis) { maiorAneis = aneis; } aneis = 0; } } } } if(aneis > maiorAneis) { maiorAneis = aneis; } printf("%d\n", maiorAneis); system("PAUSE"); return 0; } </pre>
---------------	--

Fonte: própria autora.

Neste quadro, encontramos o enunciado do problema, a especificação do problema, um conjunto de metas intermediárias que devem ser alcançadas para que o programa corresponda à especificação e uma possível implementação para o problema que atende às metas intermediárias. Na primeira etapa da análise adotada, compilamos o código na *Codeblocks* para encontrar os erros e avisos detectados pelo compilador. Em seguida, na segunda etapa da análise, para a implementação de cada programa,

procuramos por trechos de implementação (esquemas) que correspondiam às metas intermediárias identificadas para o programa. Em cada um destes trechos de implementação, procuramos por erros não sintáticos que foram categorizados de acordo com a classificação de erros cometidos por iniciantes em programação proposta por Spohrer et al. (1985). Cada erro foi então categorizado como [SPOHRER et al., 1985 p. 175]:

- ausente: quando uma parte do trecho de implementação para a meta não estava presente no programa;
- mal formado: quando uma parte do trecho de implementação para a meta estava presente, mas não foi corretamente implementado;
- desnecessário: quando uma parte do trecho de implementação para a meta estava presente, mas não deveria estar; e
- posicionado incorretamente: quando uma parte do trecho de implementação para a meta estava presente, mas posicionada no lugar errado do programa.

Além disso, para cada trecho de implementação encontrado com um erro não sintático, uma segunda dimensão de classificação para o erro foi utilizada. As categorias dessa classificação foram as seguintes [SPOHRER et al., 1985 p. 175] (adaptadas para a linguagem de programação C):

- Entrada: lendo a entrada de dados, tais como comandos de `scanf` e `gets`;
- Saída: exibindo uma saída, tais como comandos de `printf` e `puts`;
- Inicialização: inicialização de uma variável;
- Atualização: modificando o valor da variável através de uma atribuição;
- Guarda: condição de teste nas estruturas condicionais `if`, `else if` ou nas estruturas de repetição `while`, `for` e `do while`;
- Declaração: característica de dados, tais como variáveis, tipos e declaração de constantes; e
- Sintaxe: conectivos de sintaxe que delimitam o escopo de blocos de código, tais como `{` e `}`.

Para cada programa, as metas intermediárias que eles tinham que alcançar foram então classificadas como:

- meta errada: quando um ou mais erros no trecho de implementação (de acordo com a classificação apresentada) para a meta foi encontrado;
- meta correta: quando nenhum erro no trecho de implementação para a meta foi encontrado; e
- meta omitida: quando nenhuma implementação para a meta foi encontrada.

Por fim, na terceira etapa, se a implementação do programa não tivesse nenhum erro que impedisse o seu funcionamento ou para o qual já sabíamos que a saída não seria a definida na especificação, ele era então testado com um conjunto de testes para detecção de outros erros semânticos de execução e/ou erros lógicos. Se estes erros existissem, o trecho de código que o gerava era então localizado e, em seguida, eles eram categorizados com a mesma classificação adotada na segunda etapa da análise. Depois da categorização dos erros encontrados, eles foram contabilizados e uma análise descritiva e estatística para verificar as diferenças entre os diferentes grupos presentes nos quase-experimentos foi realizada.

Um exemplo deste processo de análise aplicado à implementação de um dos participantes do primeiro quase-experimento, dada como resposta ao pós-teste do Quadro 5.4, é apresentada no Quadro 5.5. Podemos perceber neste exemplo que, na primeira etapa da análise, encontramos dois erros e um aviso (*warning*) detectado pelo compilador. Na segunda etapa da análise, encontramos cinco erros que não foram detectados pelo compilador. Em seguida, esses erros categorizaram também as metas. Por fim, não realizamos a terceira etapa da análise, uma vez que o programa construído não compilou com sucesso e, por esta razão, não executa.

Quadro 5.5. Implementação de um dos participantes do primeiro quase-experimento para o pós-teste do **Quadro 5.4.**

```

1  #include <stdio.h>
2  void main() {
3      int d,i,j,qtde=0,var0=0,var1=0;
4      char v[100][100];
5
6      scanf("%d", &d);
7      fflush(stdin);
8
9      for(i=0;i<d;i++){
10         c=gets(v[i]);
11     }
12     for(i=0;i<d;i++){
13         if(d%2){
14             for(j=d;j>=0;j--){

```

```

15             if(v[i][j]=='o'){
16                 var0= qtdc++;
17             } else if(c=='E'){
18                 qtdc=o;
19             }
20         }
21     } else {
22         for(j=0;j<d;j++){
23             if((v[i][j]=='E')){
24                 var0= qtdc++;
25             } else if(v[i][j]=='o'){
26                 qtdc=o;
27             }
28         }
29     }
30 }
31 printf("%d", var0+var1);
32 }

```

Primeira etapa da análise:

- Erros detectados pelo compilador:
 - Linha 10: '`c`' não declarado;
 - Linha 18: '`o`' não declarado.
- Aviso detectado pelo compilador:
 - Linha 2: Tipo de retorno do método '`main`' não é '`int`'.

Segunda etapa da análise:

- Erros não gerados pelo compilador:
 - Linha 4: Declaração mal formada, pois faltou deixar espaço para o caracter '`\0`' nas duas dimensões da matriz;
 - Linha 14: Inicialização mal formada, pois era para a variável `j`, que itera no laço, ter iniciado com o valor `d-1`;
 - Linha 18: Atribuição mal formada, pois a variável `qtdc` está recebendo um '`o`' ao invés de zero.
 - Linha 23: Guarda mal formada, pois era para ter comparado `v[i][j]` com '`o`';
 - Linha 25: Guarda mal formada, pois era para ter comparado `v[i][j]` com '`E`';
 - Linha 26: Atribuição mal formada, pois a variável `qtdc` está recebendo um '`o`' ao invés de zero.
 - Linha 31: Saída mal formada, pois não necessitava ter o `+var1` dentro do `printf`.

Então para as metas intermediárias do programa (Quadro 5.4), temos o seguinte:

- Entrada 1: meta certa;
- Entrada 2: meta errada, pois usa variável não declarada (linha 10);
- Executa laço: meta errada, pois tem 1 inicialização mal formada (linha 14);
- Acumulador: meta errada, pois tem 1 guarda mal formada (linha 23);
- Reinicia acumulador: meta errada, pois usa variável não declarada (linha 18) e tem 2 atribuições mal formadas (linhas 18 e 26) e tem 1 guarda mal formada (linha 25);
- Atualiza maior: meta omitida;
- Saída: meta errada, pois tem 1 saída mal formada (linha 31).

Terceira etapa da análise:

- Não realizada, uma vez que o programa não compila com sucesso.

Fonte: própria autora.

Para finalizar a análise dos programas construídos pelos estudantes, todas as metas para cada um dos testes foram classificadas pela autora deste trabalho e por um segundo professor de programação como sendo de fácil, de média ou de difícil implementação. Depois, esta classificação foi utilizada para atribuir uma nota a cada um dos testes realizados pelos estudantes. Para computar quantitativamente os valores indicados pelos dois professores, atribuímos uma escala ordinal a esta classificação, ficando as metas de fácil implementação com o valor 1, de média implementação com o valor 2 e de difícil implementação com o valor 3. Em seguida, foi calculada a média aritmética das notas atribuídas pelos dois professores para cada meta e para cada um dos testes tivemos como pontuação máxima o somatório das médias para as suas metas. Sendo assim, para cada teste os estudantes tiveram a sua pontuação calculada da seguinte maneira:

- (i) para cada meta certa, o estudante pontuou o valor integral atribuído à meta;
- (ii) para cada meta errada ou omitida, o estudante não pontuou;
- (iii) para cada erro e/ou aviso detectado pelo compilador, o estudante tem uma penalidade de 1 ponto por erro e/ou aviso detectado; e
- (iv) a menor nota que o estudante pode obter em cada teste é zero.

Posteriormente, uma análise estatística foi realizada com as notas obtidas nos testes para compararmos os dados dos diferentes participantes dos quase-experimentos.

Para verificar se existe alguma correlação entre as respostas às questões de auto-explicação, o tempo de estudo dos exemplos em vídeo juntamente com as questões de auto-explicação e os resultados obtidos no pós-teste em cada um dos grupos instrucionais, examinamos as respostas dadas às perguntas de auto-explicação durante os momentos de estudo dos exemplos dos grupos instrucionais, bem como o tempo que os estudantes levaram para estudar os exemplos. Para os grupos instrucionais para os quais as perguntas de auto-explicação eram abertas, a análise aconteceu da seguinte maneira:

- (i) as respostas às perguntas relacionadas à atividade de extensão principal e à atividade de refinamento foram classificadas como certas, meio certas ou erradas, de acordo com a proximidade da resposta com uma resposta padrão considerada correta pela autora desta tese. Se a resposta fornecida

pelo estudante tivesse todos os termos considerados essenciais pela autora desta tese, ela considerou a resposta como certa. Se a resposta tivesse parte dos termos, a resposta foi considerada como meio certa. Em caso contrário, ela era considerada errada;

- (ii) as respostas às perguntas relacionadas à extensão intermediária, que incluem antecipar a implementação de um trecho de código foram classificadas como certas ou erradas, de acordo com a classificação de Spohrer et al. (1985) que apresentamos anteriormente para os erros de código cometidos pelos alunos. Se o estudante cometesse algum erro neste tipo de resposta, ela foi considerada errada. Em caso contrário, ela foi considerada certa; e
- (iii) as respostas às perguntas relacionadas à atividade de resolução de problema como um todo foram consideradas certas, meio certas e erradas, de acordo com a proximidade da resposta com a saída esperada a partir da entrada fornecida. Se a saída dada como resposta fosse idêntica à saída esperada, a resposta foi considerada certa. Se a saída dada como resposta fosse parcialmente semelhante à saída esperada, a resposta foi considerada meio certa. Nos outros casos, ela foi considerada errada.

Já para os grupos instrucionais para os quais as perguntas de auto-explicação eram de múltipla escolha, a análise foi realizada comparando as respostas fornecidas pelos estudantes ao gabarito das questões. Se a resposta selecionada pelo estudante fosse a considerada correta de acordo com o gabarito, ela era considerada correta. Ela foi considerada errada, em caso contrário. Em seguida, computamos quantitativamente cada uma das respostas às questões, atribuindo uma escala ordinal à classificação das respostas, ficando as respostas certas com o valor 1, as respostas meio certas com o valor 0.5 e as respostas erradas com o valor 0. Com base nestes valores, o valor máximo que cada estudante.

Por fim, realizamos uma avaliação de uma amostra dos três conjuntos de materiais instrucionais feita por professores das áreas de programação e de educação. A avaliação do material instrucional se justifica pois exemplos trabalhados considerados efetivos para o ensino dos estudantes deve ser compreensível pelo computador, pelos estudantes e deve comunicar efetivamente os conceitos a ser ensinados [BÖRSTLER;

CASPERSEN; NORDSTRÖM, 2015]. Além disso, os exemplos trabalhados e as questões para auto-explicação devem estar alinhadas com os objetivos de aprendizagem que precisam ser alcançados com a disciplina introdutória de programação onde estão sendo utilizados [AURELIANO; TEDESCO; GIRAFFA, 2016].

5.3 Restrições dos experimentos

A principal restrição que existe neste trabalho está relacionada à execução dos experimentos como um todo. Primeiro, houve uma grande dificuldade para o recrutamento de estudantes nos experimentos realizados aqui no Brasil. Uma tentativa sem sucesso foi tentar realizar os experimentos com estudantes do ensino médio matriculados em escolas públicas da rede municipal de ensino na cidade de Recife. Cerca de 100 estudantes foram convidados pessoalmente pela autora desta tese para participar de um minicurso de programação, mas nenhum dos estudantes se voluntariou para participar do minicurso. O recrutamento passou a funcionar apenas quando o experimento passou a contar como atividades extras de uma disciplina de introdução a programação. Porém poucos estudantes se comprometeram de fato com todas as atividades extras durante a execução da disciplina. Portanto, os dados de poucos estudantes puderam ser utilizados como dados para os experimentos realizados. O que resultou na segunda restrição, o número pequeno de estudantes participantes dos experimentos. Por fim, a terceira restrição dos experimentos diz respeito à quantidade de material que precisou ser produzida para cada um deles. Os estudantes participantes dos experimentos estavam matriculados em diferentes disciplinas de introdução a programação e o material produzido precisava estar alinhado aos objetivos de aprendizagem de cada uma destas disciplinas. Por esta razão, foi necessário produzir três conjuntos diferentes de materiais instrucionais para um dos grupos de estudantes participantes dos quase-experimentos, o que gerou um trabalho extra para a produção deste material.

5.4 Implicações

Este quinto capítulo apresentou os procedimentos metodológicos adotados nesta tese. Vimos que foi empregada a visão geral filosófica pragmática e um projeto misto de pesquisa com diferentes casos. Com o objetivo de avaliar a eficácia da primeira versão da *Stepwise Self-explanation*, realizamos um experimento piloto com participantes de uma instituição de ensino médio e técnico localizada na Dinamarca, no segundo

semestre de 2014. Neste experimento utilizamos exemplos trabalhados em vídeo da série *The Joy of Code* que ensinam a linguagem de programação Java utilizando a ferramenta *Greenfoot*. Os participantes foram divididos em dois grupos, o instrucional (GI) e o controle (GC). Enquanto o GI estudou os exemplos trabalhados em vídeo, auto-explicando-os em folhas de papel, o GC estudou os vídeos a sua própria maneira. Por restrições de tempo para realização do estudo, o experimento foi dividido em duas sessões, uma sessão que foi realizada em casa e outra sessão que foi realizada na instituição de ensino. Após o estudo, foi feita uma análise estatística dos programas criados pelos estudantes para resposta às perguntas.

Para avaliar a eficácia da segunda versão da Stepwise Self-explanation, realizamos três quase-experimentos em três diferentes instituições de ensino localizadas no estado de Pernambuco, no primeiro semestre de 2015. O primeiro quase-experimento envolveu participantes de graduação que estavam cursando uma disciplina introdutória de programação pela segunda vez. Nele, utilizamos exemplos trabalhados em vídeo produzidos pela autora desta tese que ensinaram a linguagem de programação C utilizando a ferramenta *CodeBlocks*. Os estudantes foram divididos em três grupos, um grupo controle (GC) e dois grupos instrucionais (GI1, que utilizou as perguntas de AE abertas e o GI2, que utilizou perguntas de AE de múltipla escolha). Este quase-experimento foi realizado em duas sessões de aula extras com cada grupo de participantes.

O segundo quase-experimento envolveu participantes de graduação que estavam cursando uma disciplina introdutória de programação pelo menos pela terceira vez. Nele, também utilizamos exemplos trabalhados em vídeo produzidos pela autora desta tese que ensinaram a linguagem de programação C utilizando a ferramenta *CodeBlocks*. Neste experimento, tivemos apenas um grupo instrucional, que estudou os exemplos trabalhados em vídeo utilizando questões de AE abertas. Este quase-experimento foi realizado em quatro sessões de aula extras com o grupo de participantes.

O terceiro quase-experimento, por sua vez, envolveu participantes do ensino médio-técnico sem experiência em programação e do ensino técnico que estavam cursando uma disciplina introdutória de programação pela primeira vez de uma instituição de ensino de nível médio, técnico e superior do estado de Pernambuco. Nele, utilizamos exemplos trabalhados em vídeo produzidos pela autora desta tese que utilizaram a ferramenta Scratch como ferramenta e como linguagem de programação.

Os participantes foram organizados em dois diferentes grupos, o controle (GC) e o instrucional (GI). Este quase-experimento foi realizado como minicurso de uma semana de tecnologia promovida pela instituição de ensino.

Ao final dos três quase-experimentos, para responder as questões de pesquisa dos três quase-experimentos, examinamos os programas construídos pelos estudantes como respostas aos pré e pós-testes por meio de uma abordagem mista. Para todos os quase-experimentos realizados, cada um dos programas construídos pelos estudantes nos pré e pós-testes passou por uma análise realizada em três etapas, uma etapa para detecção de erros e warnings pelo compilador, uma etapa de análise manual para detecção e categorização dos erros encontrados e, por fim, uma etapa de execução de um conjunto de casos de testes. Somente após esta fase de análise qualitativa, foi atribuída uma nota aos programas construídos pelos estudantes e assim realizou-se uma análise estatística, objetivando comparar os dados dos diferentes participantes dos quase-experimentos. Veremos no próximo capítulo os resultados obtidos do experimento piloto e dos quase-experimentos realizados.

6 Resultados obtidos

Este sexto capítulo tem por objetivo apresentar os resultados obtidos nos experimentos realizados nesta tese. Para tanto, apresentamos na Seção 6.1 os resultados obtidos a partir dos dados coletados durante a execução do experimento piloto. Em seguida, na Seção 6.2 apresentamos os resultados obtidos a partir dos dados coletados na execução dos três quase-experimentos. Por fim, na Seção 6.3 discutimos as considerações finais do capítulo.

6.1 Experimento piloto

Como dissemos anteriormente, o experimento piloto foi dividido em duas etapas, a sessão online e a sessão de sala de aula. Nas seções a seguir apresentaremos os resultados obtidos a partir dos dados coletados durante estas duas sessões.

6.1.1.1 Dados coletados durante a sessão online

Do questionário inicial, coletamos os seguintes dados. O GI tinha 7 participantes do sexo masculino, mas um deles teve problema com o computador, portanto, os seus dados não foram utilizados. O GI restante tinha 6 estudantes com idades entre 16 e 19 anos (média=17.33 anos). Eles não tinham experiência com programação. Todos, com exceção de um, já tinham usado vídeos do Youtube para aprender sobre algum assunto (e.g., usaram vídeos sobre como usar um determinado software ($n=3$) e para aprender sobre como fazer um jogo ($n=3$)). Para estudar os assuntos mencionados, eles utilizaram os vídeos da maneira mostrada na Tabela 6.1.

Tabela 6.1. Como os alunos do GC e do GI utilizaram os vídeos enquanto estudavam algum assunto.

Como você utilizou os vídeos?	#de estudantes	
	GC	GI
Eu pausei, avancei ou voltei para prestar atenção em determinadas partes do vídeo	5	3
Eu assisti mais de uma vez o mesmo vídeo	4	3
Eu assisti mais de uma vez partes do mesmo vídeo	2	1
Eu tentei reproduzir o que eu estava vendo no vídeo	2	4
Eu fazia anotações enquanto assistia o vídeo	1	1
Eu respondi questões enquanto assistia o vídeo	1	1
Eu simplesmente assisti o vídeo sem executar nenhuma outra atividade	1	1

Fonte: própria autora.

O grupo GC também teve 7 estudantes do sexo masculino, mas um deles tinha experiência com programação, portanto os seus dados não foram utilizados. O GC restante tinha estudantes com idade entre 16 e 18 anos (média=17.17 anos) e eles não tinham experiência com programação. De maneira similar ao GI, todos com exceção de um, já tinha usado vídeos do Youtube para aprender sobre algum assunto (e.g., a maioria usou vídeos para aprender como fazer jogos (n=4)). Para estudar os assuntos mencionados, eles utilizaram os vídeos da maneira mostrada na Tabela 6.1.

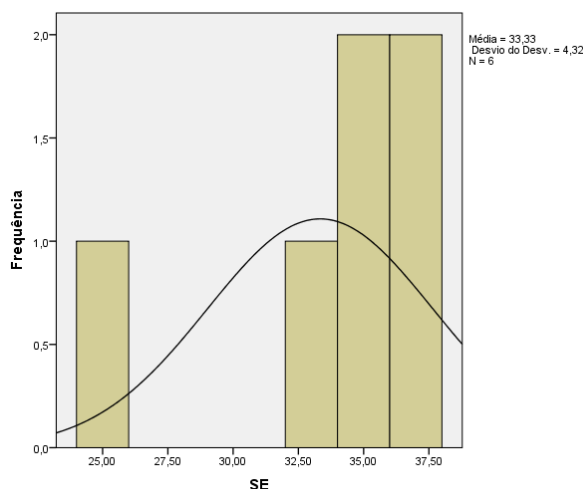
Os vídeos das telas dos computadores, gravadas durante a sessão online mostraram que todos os alunos puderam (i) usar o *Greenfoot* e o software para gravar as suas telas depois de instalá-los; (ii) usaram o Dropbox para carregar os vídeos resultantes; e (iii) finalizaram o exercício prático inicial. Assim, todos eles utilizaram as ferramentas que eram necessárias para a sessão de sala de aula do estudo.

6.1.1.2 *Dados coletados durante a sessão de sala de aula*

Para responder a [QP1] *O aprendizado de programação através da AE de exemplos em vídeo apresenta melhores resultados quando comparado ao aprendizado de programação somente através de exemplos em vídeo?*, examinamos as respostas escritas e os vídeos das telas dos computadores dos estudantes de ambos os grupos enquanto eles estavam realizando o pós-teste. Estes dados complementam um ao outro, assim os consideramos conjuntamente para avaliar as respostas fornecidas pelos estudantes. Para o pós-teste, as notas no GI foram de 25 a 37 pontos, com mediana igual a 34.5 pontos e média 33.33 pontos (DP=4.32). As notas no GC foram de 29 a 35, com mediana igual a 32 pontos e média de 32 pontos (SD=2.76). As estatísticas descritivas mostraram que os participantes no GI tiveram melhores resultados do que os participantes no GC. Entretanto, para saber se essa diferença é significativa ou não, devemos analisar estes dados através de um teste estatístico. Analisando os histogramas dos dois grupos (Figura 6.1 e Figura 6.2), notamos que não possuímos uma distribuição normal em nenhum dos casos. Além disso, o número de participantes no estudo foi pequeno. Por estas duas razões, o teste estatístico mais apropriado é o teste de Mann-Whitney [DANCEY; REIDY, 2013]. Usando o teste estatístico de Mann-Whitney, encontramos $U=11.5$; $z=-1.052$; e $p=0.29$. Por esta razão, não podemos confirmar que a diferença das notas entre o GC e o GI é estatisticamente significativa no que diz respeito

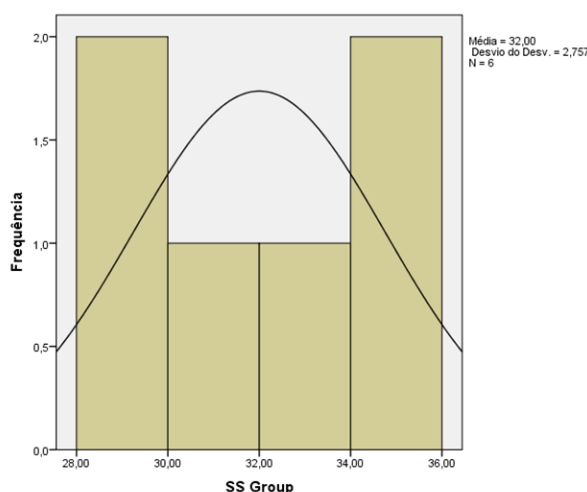
ao pós-teste. Consequentemente, nossos dados não nos permitem distinguir entre as abordagens usadas no GC e no GI, respectivamente.

Figura 6.1. Histograma do GI.



Fonte: própria autora.

Figura 6.2. Histograma do GC.



Fonte: própria autora.

Para responder a [QP2] *Existe alguma correlação entre as respostas às questões de auto-explicação que foram respondidas corretamente e o resultado obtido no pós-teste no grupo instrucional?*, analisamos as respostas escritas às perguntas de estímulo a auto-explicação e as notas dos pós-testes dos estudantes do GI. Propusemos um conjunto de 38 questões para estímulo à auto-explicação. As notas fornecidas às questões de auto-explicação variaram de 16 a 26.5 pontos, com mediana de 22 pontos e média de 21.36 pontos (DP=3.76). Para saber se houve uma relação entre as notas obtidas nas respostas escritas às perguntas de estímulo a auto-explicação e as notas do

pós-teste, utilizamos o coeficiente de correlação de Spearman [DANCEY; REIDY, 2013]. Neste caso, encontramos uma correlação significativa entre as notas obtidas nas respostas escritas às perguntas de estímulo a auto-explicação e as notas do pós-teste, uma vez que o valor de $r=0.899$; e $p=0.007$.

No final do estudo, coletamos o *feedback* dos estudantes através de um questionário final. Quando perguntados sobre a sua experiência estudando a partir dos vídeos, o GI apresentou *feedback* positivo. Estudantes acharam os exemplos em vídeo fáceis ($n=4$), porque eles podiam assistir novamente em caso de dúvidas ($n=2$) ou reproduzir o conteúdo que aprenderam a partir dos vídeos usando o *Greenfoot* ($n=1$). Outros adjetivos utilizados foram: interessante, ótimo e divertido. Quando perguntados sobre a utilização dos vídeos combinados com a ficha de questões para estímulo a auto-explicação, eles também apresentaram *feedback* positivo. Eles acharam útil ($n=4$), porque as questões ajudaram a lembrar as coisas ($n=2$), porque as questões foram importantes para resumir a informação aprendida ($n=1$) e porque um deles pode monitorar o que aprendeu ($n=1$). Apesar desse *feedback* positivo, nossa abordagem de combinar os vídeos com as questões de auto-explicação foi considerada demorada, provavelmente porque eles tiveram atividades sequenciais de estudo praticamente ininterruptas durante toda sessão, enquanto o GC ficou ocioso em alguns momentos apenas esperando o tempo de estudo ser finalizado.

De maneira similar, o GC apresentou *feedback* positivo quando perguntado sobre a experiência de estudar a partir dos vídeos. Os estudantes acharam que estudar a partir dos exemplos em vídeo facilitou o entendimento do assunto abordado ($n=4$). Em particular, eles disseram que os exemplos em vídeos do *The Joy of Code* eram muito bons, simples e mais fáceis de entender do que outros que eles assistiram anteriormente porque o professor apresentava o código como um simples passo a passo e em uma velocidade apropriada. Quando perguntados sobre o que tinham feito enquanto estavam estudando os vídeos, eles disseram que escreveram no papel o que consideraram importante e também o que era difícil de lembrar ($n=2$).

6.1.2 Discussão dos resultados

Examinando as notas obtidas pelos estudantes do GC e do GI no pós-teste, não podemos confirmar que a diferença entre estas notas foi significativa. Assim, não podemos replicar o efeito da auto-explicação através deste estudo por uma combinação

de dois motivos. Primeiro, o conteúdo estudado pode ter sido muito simples para os estudantes. Em geral, os estudos que conseguiram replicar o efeito da auto-explicação na área de programação analisaram o efeito em conteúdos que eram muito mais complexos do que o que nós utilizamos no presente estudo (e.g., Pirolli e Recker (1994) e Bielaczyc et al. (1995) encontraram o efeito da auto-explicação para o ensino de funções, o penúltimo tópico na lista de tópicos para a disciplina Conceitos fundamentais de programação do currículo da ACM [J. T. F. ON COMPUTING CURRICULA, 2013]). Entretanto, devido a restrições de tempo no nosso estudo, estudantes tiveram que estudar apenas comandos muito simples. Consequentemente, os estudantes em ambos os grupos não experimentaram um grande nível de dificuldade ao estudar e ao resolver problemas associados, o que foi confirmado pelo *feedback* dos estudantes ao final do estudo realizado.

Aliado a isso, o segundo motivo é que as explicações instrucionais apresentadas nos exemplos em vídeo podem ter contribuído para a falta do efeito da auto-explicação. Estas explicações instrucionais continham as respostas às perguntas de estímulo a auto-explicação que o GI tinha que responder. Além disso, observamos das notas escritas pelo GC que algumas delas correspondem às respostas às perguntas de auto-explicação. Por exemplo, anotações de cinco estudantes do GC foram relacionadas ao objetivo de escrever um trecho de código que o professor implementou no vídeo. Estas anotações correspondiam a respostas a questão “Qual o principal objetivo do pedaço de código que ele (professor) acabou de escrever?” relacionada à atividade de extensão que o GI deveria responder. Um outro exemplo envolve anotações do código produzido durante a execução de uma atividade de refinamento. Estas anotações eram respostas a seguinte questão: “Qual o código que ele (professor) escreveu para fazer a tartaruga andar?”. Desta maneira, parece que estudantes no GC se engajaram em uma atividade de aprendizado que foi similar àquela do GI durante a fase de aprendizado. Como resultado, obtivemos que o GC teve um desempenho similar quando comparado a estudantes do GI.

Analisando mais de perto os dados do GI, encontramos uma correlação significativa entre as notas das respostas às perguntas de auto-explicação e as notas do pós-teste. Nossos dados mostraram que quase 90% da variação das notas dos pós-testes dos estudantes pode ser explicado pela variação das notas das respostas às perguntas de auto-explicação. Assim, os estudantes do GI que responderam mais corretamente as

perguntas de auto-explicação durante a fase de aprendizado tiveram os melhores desempenhos na fase de programação.

Além disso, estudantes forneceram um *feedback* positivo sobre a nossa abordagem. No que diz respeito ao uso dos exemplos em vídeo como material instrucional, as opiniões dos estudantes sugeriram que eles gostaram da experiência. Alguns dos adjetivos utilizados para expressar as suas opiniões ao utilizar os exemplos em vídeo foram: fácil, divertido e ótimo. No que diz respeito à utilização das perguntas para auto-explicação, a maioria dos estudantes no GI disseram que respondê-las enquanto estavam aprendendo a partir dos exemplos em vídeo, os ajudaram a lembrar o que eles estavam estudando. Além disso, eles disseram também que elas os ajudaram a resumir e monitorar o que estavam aprendendo. Embora eles tenham tido uma boa experiência ao utilizar a nossa abordagem, a maioria deles considerou que ela era demorada, provavelmente porque eles tiveram que desenvolver muitas atividades em um tempo relativamente curto.

6.2 Quase-experimentos

Como dissemos anteriormente, realizamos três quase-experimentos para avaliar a segunda versão da *Stepwise Self-explanation*. Nas seções a seguir, apresentaremos os resultados obtidos a partir dos dados coletados durante estes três quase-experimentos.

6.2.1 Primeiro quase-experimento

6.2.1.1 Perfil dos participantes

O GC contou com 5 participantes ao longo das duas aulas extras, no entanto conseguimos coletar todos os dados para apenas 3 deles. Sendo assim, no GC tivemos 3 estudantes, todos com 19 anos, sendo um do sexo feminino e dois do sexo masculino. Todos eles estavam cursando a disciplina introdutória de programação pela segunda vez. Quando perguntados sobre os seus hábitos de estudo para a disciplina de programação, os estudantes deram as respostas exibidas na Tabela 6.2. Todos, com exceção de 1, já tinham usado vídeos do Youtube para aprender programação, utilizados conforme mostrado na Tabela 6.3.

Tabela 6.2. Como os estudantes dos GC, GI1 e GI2 estudam para a disciplina de IP.

Quando eu estou estudando para a disciplina de IP, eu normalmente:		#de estudantes				
		N ^a	R ^b	AV ^c	F ^d	S ^e

Somente assisto as aulas do professor, não estudo fora de sala de aula.	GC		2		1	
	GI1			2		
	GI2		1	2		
Estudo a partir dos materiais (slides, livros, etc.) fornecidos pelo professor.	GC	1	1		1	
	GI1			1	1	
	GI2			2	1	
Estudo a partir das anotações que eu fiz em sala de aula.	GC		1		1	1
	GI1		1	1		
	GI2	1		2		
Copio somente os programas que o professor faz em sala de aula, sem procurar entender o que foi feito.	GC			1		2
	GI1	1	1			
	GI2	2	1			
Estudo os exemplos e/ou exercícios que foram vistos em sala de aula, tentando reproduzi-los para entendê-los melhor.	GC				2	1
	GI1				1	1
	GI2				1	2
Faço os exercícios recomendados pelo professor.	GC			1	1	1
	GI1				1	1
	GI2				2	1
Pesquisei por novos materiais, exemplos e exercícios que possam me auxiliar no entendimento do assunto.	GC		1		2	
	GI1			1	1	
	GI2		2	1		
Tiro dúvidas com os meus colegas e/ou monitores da disciplina.	GC				1	2
	GI1				1	1
	GI2				1	2

(a) Nunca (b) Raramente (c) As vezes (d) Frequentemente (e) Sempre

Fonte: própria autora.

O GI1 contou com 4 participantes nas duas aulas extras, no entanto conseguimos coletar todos os dados para apenas 2 deles. Sendo assim, no GI1 tivemos 2 estudantes, um com 19 anos e outro com 20 anos, os dois do sexo masculino. Ambos estavam cursando a disciplina introdutória de programação pela segunda vez. Quando perguntados sobre os seus hábitos de estudo para a disciplina de programação, os estudantes deram as respostas exibidas na Tabela 6.2 Além disso, ambos já tinham usado vídeos do Youtube para aprender programação, utilizados conforme mostrado na Tabela 6.3.

Tabela 6.3. Como os estudantes dos GC, GI1 e GI2 lembram de ter utilizado vídeos para estudar para a disciplina de IP.

Como você se lembra de ter utilizado esses vídeos?		#de estudantes				
		N ^a	R ^b	AV ^c	F ^d	S ^e
Eu pausei, avancei e/ou voltei o vídeo para prestar atenção em partes específicas dele.	GC			1	1	
	GI1		1		1	
Eu assisti ao mesmo vídeo (ou partes dele) mais de uma vez.	GC				2	
	GI1				2	
Eu tentei reproduzir o que era feito no	GC					2

vídeo.	GI1				2	
Eu fiz anotações enquanto assistia ao vídeo.	GC		1	1		
	GI1		1	1		
Eu respondi perguntas apresentadas no vídeo enquanto assistia ao vídeo.	GC		1	1		
	GI1			1	1	
Eu assisti ao vídeo somente, sem fazer nenhum tipo de atividade.	GC	1	1			
	GI1		1	1		

(a) Nunca (b) Raramente (c) As vezes (d) Frequentemente (e) Sempre

Fonte: própria autora.

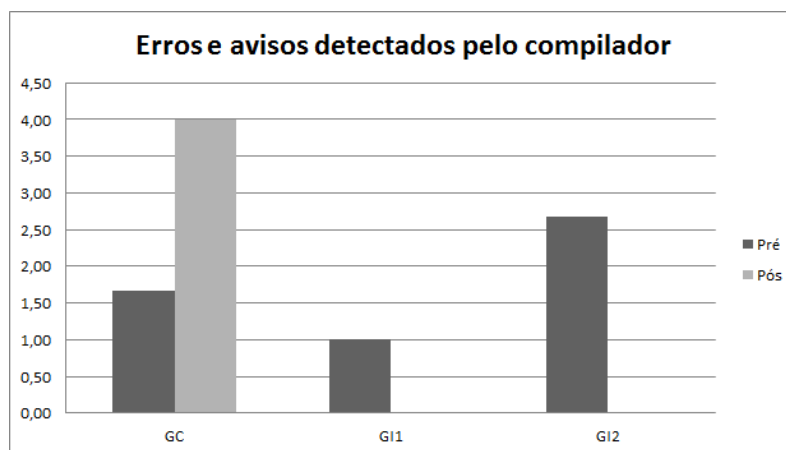
O GI2 contou com 4 participantes nas duas aulas extras, no entanto, conseguimos coletar todos os dados para apenas 3 deles. Sendo assim, no GI2 tivemos 3 estudantes, com idades entre 18 e 20 anos, todos do sexo masculino. Dois deles estavam cursando a disciplina introdutória de programação pela segunda vez e o terceiro estava cursando a disciplina como isolada pela primeira vez. Quando perguntados sobre os seus hábitos de estudo para a disciplina de programação, os estudantes deram as respostas exibidas na Tabela 6.2. Nenhum deles usou vídeos para aprender programação anteriormente.

6.2.1.2 Desempenho dos participantes

Para responder as [QP1] *O aprendizado de programação através da AE de exemplos em vídeo (i.e., dos grupos instrucionais) apresenta melhores resultados quando comparado ao aprendizado de programação somente através de exemplos em vídeo (i.e., do grupo controle)?* e [QP2] *Existe alguma diferença em relação ao aprendizado de programação e os resultados obtidos entre os diferentes grupos instrucionais?*, examinamos os programas construídos pelos participantes do quase-experimento como respostas aos pré e pós-testes.

As médias de erros e avisos detectados pelo compilador para os GC, GI1 e GI2 nos pré e pós-testes são apresentados na Figura 6.3. Nesta figura, podemos perceber que o GC aumentou a média de erros e avisos gerados pelo compilador de 1.67 erros no pré-teste para 4 erros no pós-teste. Ao contrário do GC, o GI1 e o GI2 diminuíram as suas médias de erro do pré para o pós-teste. O GI1 diminuiu de 1 erro no pré-teste para nenhum erro no pós-teste. O GI2, por sua vez, apresentou a maior queda no número de erros, diminuindo de 2.67 erros no pré-teste para nenhum erro no pós-teste.

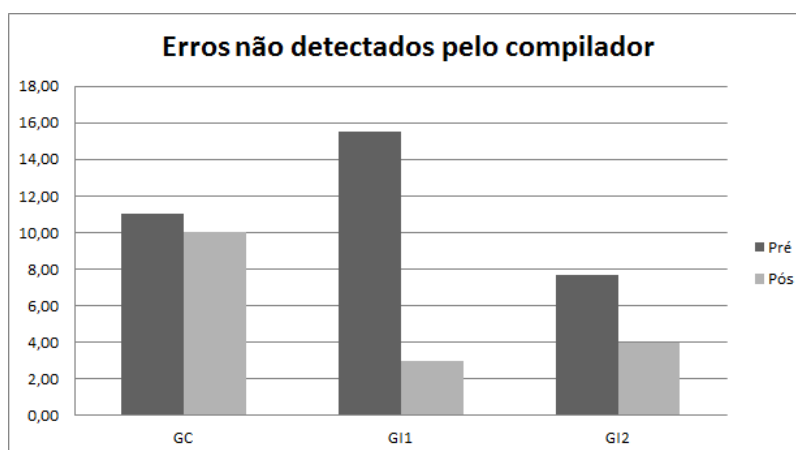
Figura 6.3. Médias de erros e avisos detectados pelo compilador para GC, GI1 e GI2 nos pré e pós-testes.



Fonte: própria autora.

As médias de erros não detectados pelo compilador para os GC, GI1 e GI2 nos pré e pós-testes são apresentados na Figura 6.4. Nesta figura, podemos perceber que os três grupos diminuíram a quantidade de erros deste tipo, no entanto o GI1 e o GI2 tiveram um decréscimo de erros mais acentuado quando comparados ao GC. Enquanto GC diminuiu de 11 erros no pré-teste para 10 erros no pós-teste, o GI1 reduziu de 15,5 erros no pré-teste para somente 3 erros no pós-teste e o GI2 reduziu de 7,67 erros no pré-teste para 4 erros no pós-teste. Sendo assim, verificando a diferença entre a média dos pré e pós-testes de cada um dos grupos para a quantidade média de erros não detectados pelo compilador, verificamos que o GI1 apresentou a maior redução dos três grupos, o GI2 apresentou a segunda maior redução e o GC a menor delas.

Figura 6.4. Médias de erros não detectados pelo compilador para GC, GI1 e GI2 nos pré e pós-testes.

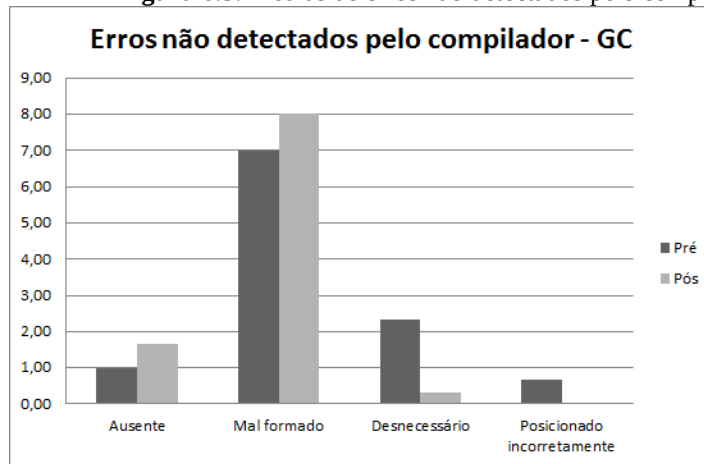


Fonte: própria autora.

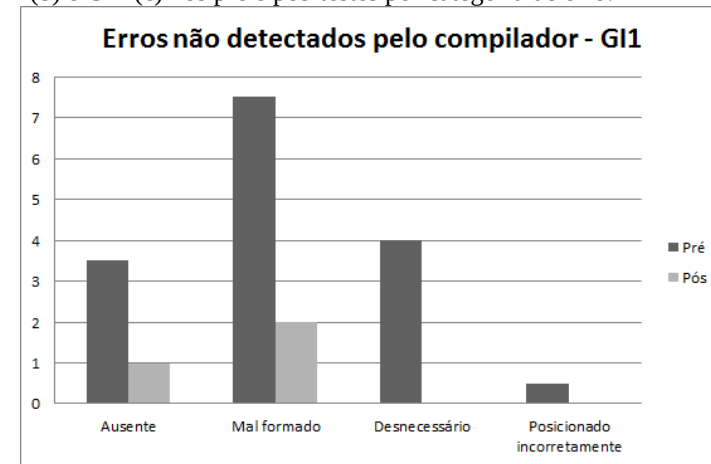
Na Figura 6.5 são apresentadas as médias de erros não detectados pelo compilador para os GC, GI1 e GI2 nos pré e pós-testes organizados pelas categorias de maior

dimensão destes erros (i.e., erros ausentes, mal formados, desnecessários e posicionados incorretamente). Na Figura 6.5 (a), podemos perceber que as médias para os erros ausentes e mal formados aumentaram, respectivamente, de 7 para 8 erros e de 1 para 1.67 erro do pré para o pós-teste no GC. Juntos estes tipos de erro foram os principais responsáveis pela alta média de erros que o GC teve, principalmente no pós-teste. Na Figura 6.5 (b), podemos notar que as médias para todas as categorias de erros tiveram seus valores reduzidos do pré para o pós-teste no GI1. Foi justamente em razão desta redução que o GI1 conseguiu apresentar a maior taxa de diminuição na média de erros não detectados pelo compilador, conforme constatado anteriormente. Finalmente, na Figura 6.5 (c), podemos observar que houve um pequeno aumento nas médias dos erros desnecessários e posicionados incorretamente, respectivamente, de 0 no pré para 0.33 erro no pós-teste e de 1.33 no pré para 1.67 erro no pós-teste no GI2. Entretanto, as médias para os erros ausentes e mal formados tiveram uma alta taxa de redução, respectivamente, de 0.67 no pré para nenhum erro no pós-teste e de 5.67 no pré para 2 erros no pós-teste.

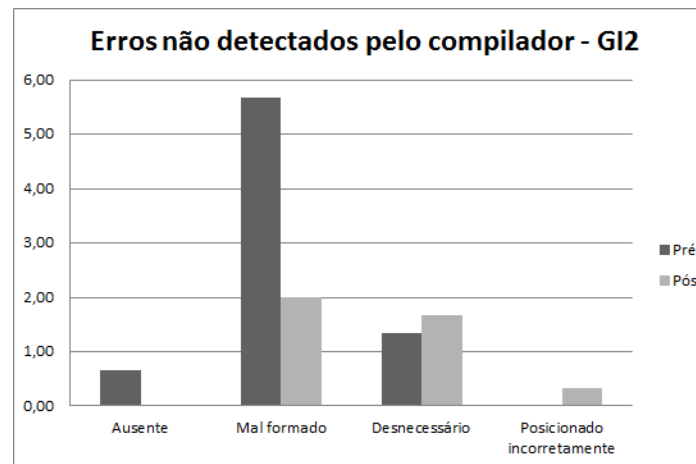
Figura 6.5. Médias de erros não detectados pelo compilador para GC (a), GI1 (b) e GI2 (c) nos pré e pós-testes por categoria de erro.



(a)



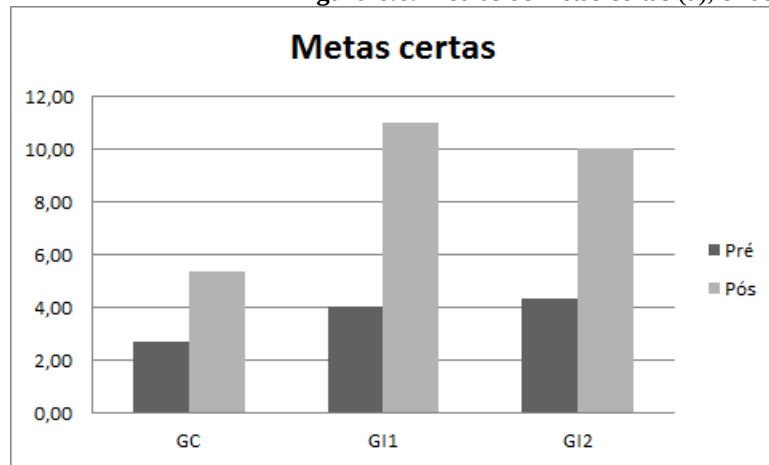
(b)



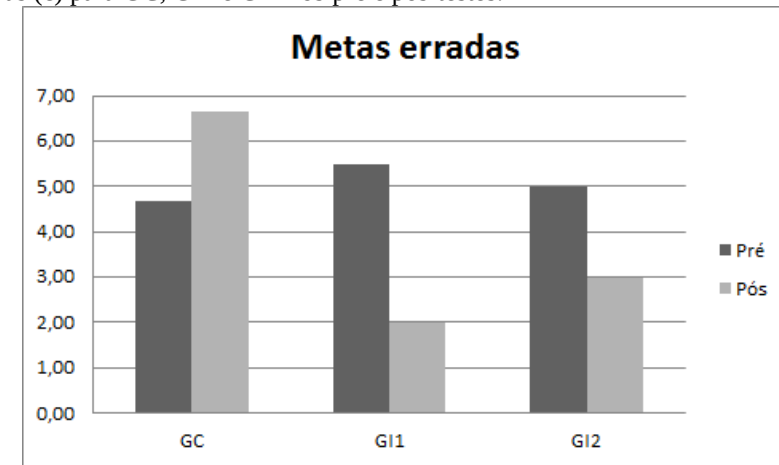
(c)

Fonte: própria autora.

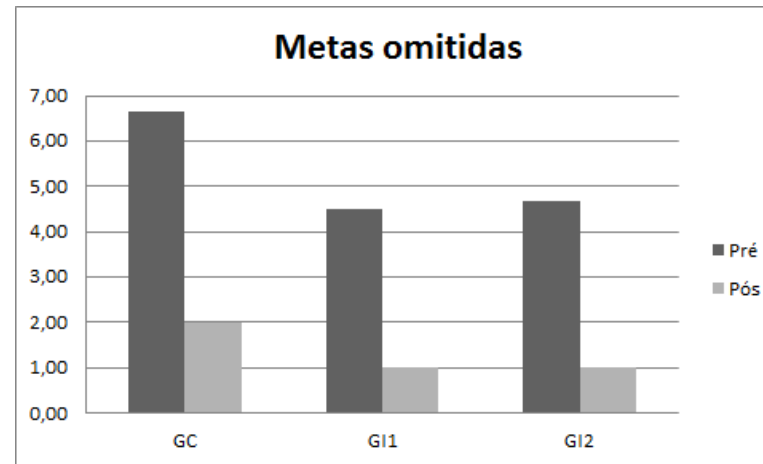
Figura 6.6. Médias de metas certas (a), erradas (b) e omitidas (c) para GC, GI1 e GI2 nos pré e pós-testes.



(a)



(b)



(c)

Fonte: própria autora.

Com base nos erros cometidos pelos estudantes, classificamos as metas como certas, erradas ou omitidas. Na Figura 6.6 (a) são apresentadas as metas certas para cada um dos grupos. Nesta figura, podemos perceber que todos os grupos aumentaram a quantidade de metas certas depois de estudarem os exemplos em vídeo. O GC foi o grupo que teve a menor média de crescimento para as metas corretas, indo de 2.67 no pré para 5.33 metas no pós-teste. O GI2 teve uma média de crescimento um pouco maior, indo de 4.3 metas corretas no pré-teste para 10 metas corretas no pós-teste. O GI1, por sua vez, foi o grupo que teve o maior crescimento indo de 4 metas corretas no pré-teste para 11 metas corretas no pós-teste.

Na Figura 6.6 (b) são apresentadas as metas erradas para cada um dos grupos. O que notamos nesta figura é que os resultados para todos os grupos no pré-teste foram bastante similares. O mesmo não acontece com os resultados do pós-teste. Enquanto o GI1 e GI2 reduziram a quantidade média de metas erradas do pré para o pós-teste, os dados revelaram um aumento na quantidade média de metas erradas para o GC. O GI1 foi o grupo que teve a maior redução na média das metas erradas, indo de 5.5 metas erradas no pré-teste para somente 2 metas erradas no pós-teste. O GI2 teve uma redução moderada, indo de 5 metas erradas no pré-teste para 3 metas erradas no pós-teste. Ao contrário dos GI1 e GI2, o GC aumentou a quantidade média de metas erradas, indo de 4.67 metas erradas no pré-teste para 6.67 metas erradas no pós-teste.

Já na Figura 6.6 (c) são apresentadas as metas omitidas para cada um dos grupos. Neste caso, os GI1 e GI2 tiveram um desempenho muito semelhante, diminuindo de 4.5 e 4.67 metas omitidas no pré-teste, respectivamente, para apenas 1 meta omitida no pós-teste. O GC, por sua vez, diminuiu a sua quantidade de metas omitidas de 6.67 no pré-teste para 2 metas omitidas no pós-teste.

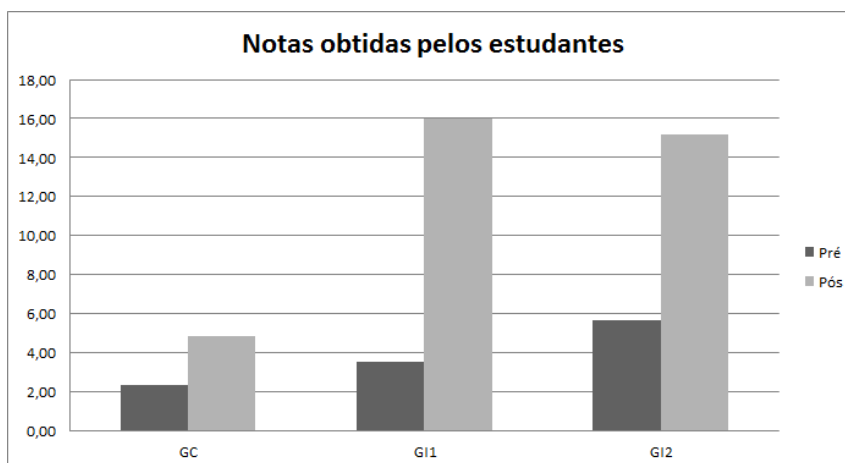
Após classificarmos as metas como erradas, certas ou omitidas, de acordo com o que foi descrito na Seção 5.2.4, computamos quantitativamente os pré e pós-testes para cada um dos quase-experimentos. Ao final deste primeiro quase-experimento, cada estudante poderia ter como nota máxima 21 pontos ao todo para cada um dos testes (i.e., 21 pontos ao máximo para os pré-testes da primeira e segunda aulas extras mais 21 pontos ao máximo para os pós-testes da primeira e segunda aulas extras). Dentro deste contexto, as notas do pré-teste para o GC variaram entre 1 e 3, com mediana igual a 3 e média igual a 2.33 (DP=1.15). As notas do pré-teste para o GI1 variaram entre 1 e 6,

com mediana e média iguais a 3.5 (DP=3.53). As notas do pré-teste para o GI2 variaram entre 0 e 10, com mediana igual a 7 e média igual a 5.67 (DP=5.13). Comparando os dados do GC e do GI1, podemos perceber que GC e GI1 não apresentam uma diferença estatisticamente significativa em relação aos resultados do pré-teste ($U=2.5$; $z=-0.30$; $p=0.80$) e, por isso, eles são comparáveis. Comparando os dados do GC e do GI2, podemos perceber que GC e GI2 não apresentam uma diferença estatisticamente significativa em relação aos resultados do pré-teste ($U=3$; $z=-0.66$; $p=0.70$) e, por isso, eles são comparáveis. Comparando os dados do GI1 e do GI2, podemos perceber que GI1 e GI2 não apresentam uma diferença estatisticamente significativa em relação aos resultados do pré-teste ($U=2$; $z=-0.58$; $p=0.80$) e, por isso, eles são comparáveis.

As notas do pós-teste para o GC variaram entre 0 e 8.5, com mediana igual a 7 e média igual a 5.17 (DP=4.54). As notas do pós-teste para o GI1 variaram entre 14.5 e 17.5, com mediana e média iguais a 16 (DP=2.12). As notas do pós-teste para o GI2 variaram entre 13.5 e 16.5, com mediana igual a 15.5 e média igual a 15.17 (DP=1.53). Comparando os dados do GC e do GI1, podemos perceber que GC e GI não apresentam uma diferença estatisticamente significativa em relação aos resultados do pós-teste ($U=0$; $z=-1.73$; $p=0.20$). Comparando os dados do GC e do GI2, podemos perceber que GC e GI2 não apresentam uma diferença estatisticamente significativa em relação aos resultados do pós-teste ($U=0$; $z=-1.96$; $p=0.10$). Comparando os dados do GI1 e do GI2, podemos perceber que GI1 e GI2 não apresentam uma diferença estatisticamente significativa em relação aos resultados do pré-teste ($U=2$; $z=-0.58$; $p=0.80$).

A média das notas obtidas pelos estudantes no pré-teste e pós-teste aplicados nas duas aulas extras são exibidas na Figura 6.7. Nesta figura, podemos perceber que o GC foi o grupo com o pior desempenho aumentando de 2.33 pontos no pré para 5.17 pontos no pós-teste. Podemos constatar também que o GI1 foi o grupo que apresentou o melhor desempenho aumentando de 3.5 pontos no pré-teste para 16 pontos no pós-teste. Finalmente, verificamos que o GI2 teve um desempenho intermediário aumentando de 5.67 pontos no pré para 15.17 pontos no pós-teste. Embora o resultado do pós-teste para este grupo tenha sido muito próximo do GI1, como a nota que o GI2 tirou no pré-teste foi muito maior do que a do GI1, esse aumento no GI2 foi menor do que o do GI1, justificando assim este grupo ter ocupado a posição de desempenho intermediário.

Figura 6.7. Notas obtidas pelos GC, GI1 e GI2 no pré-teste e no pós-teste.



Fonte: própria autora.

Para responder a [QP3] *Existe alguma correlação entre as respostas às questões de auto-explicação, o tempo de estudo dos exemplos em vídeo juntamente com as questões de auto-explicação e os resultados obtidos no pós-teste em cada um dos grupos instrucionais?*, examinamos as respostas dadas às perguntas de auto-explicação durante os momentos de estudo dos exemplos nas duas aulas extras ministradas aos grupos instrucionais, bem como o tempo que os estudantes levaram para estudar os exemplos nestas aulas. De acordo com a pontuação que foi definida na Seção 5.2.4, o valor máximo que cada estudante poderia obter em seus momentos de estudo em cada aula deste experimento foi de 15 pontos.

As notas fornecidas às questões de auto-explicação em cada aula para o GI1 variaram de 2 a 8.5 pontos, com mediana de 7.75 e média de 6.5 pontos (DP=3.03). Os tempos de estudo em cada aula para o GI1 variaram de 31m9s a 41m53s, com mediana de 38m33s e média de 37m32s (DP=4m56s). Para saber se houve uma relação entre as notas obtidas nas respostas escritas às perguntas de estímulo a auto-explicação, o tempo de estudo dos exemplos e as notas do pós-teste no GI1, utilizamos o coeficiente de correlação de Spearman. Neste caso, não encontramos uma correlação significativa entre: (i) as notas obtidas nas respostas escritas às perguntas de estímulo a auto-explicação e o tempo de estudo dos exemplos ($r\hat{=}-0.8$; $p=0.1$); (ii) as notas obtidas nas respostas escritas às perguntas de estímulo a auto-explicação e as notas do pós-teste ($r\hat{=}0.4$; $p=0.3$); e (iii) tempo de estudo dos exemplos e as notas do pós-teste ($r\hat{=}-0.8$; $p=0.1$).

Para o GI2, as notas fornecidas às questões de auto-explicação em cada aula variaram de 12 a 14 pontos, com mediana de 12 e média de 12.5 pontos ($DP=0.84$). Os tempos de estudo em cada aula para o GI2 variaram de 26m12s a 40m13s, com mediana de 35m33s e média de 32m19s ($DP=4m55s$). De maneira idêntica ao que fizemos para o GI1, utilizamos o coeficiente de correlação de Spearman para saber se houve uma relação entre as notas obtidas nas respostas escritas às perguntas de estímulo a auto-explicação, o tempo de estudo dos exemplos e as notas do pós-teste no GI2. Neste caso, também não encontramos uma correlação significativa entre: (i) as notas obtidas nas respostas escritas às perguntas de estímulo a auto-explicação e o tempo de estudo dos exemplos ($r=0.07$; $p=0.45$); (ii) as notas obtidas nas respostas escritas às perguntas de estímulo a auto-explicação e as notas do pós-teste ($r=0.11$; $p=0.42$); e (iii) o tempo de estudo dos exemplos e as notas do pós-teste ($r=-0.52$; $p=0.15$).

6.2.1.3 Avaliação do material instrucional

O material instrucional foi avaliado pelos estudantes participantes do experimento e por professores de diferentes áreas. Os resultados obtidos a partir destas avaliações são apresentados nas seções a seguir.

6.2.1.3.1 Avaliação dos participantes

Na fase final do experimento, coletamos o *feedback* dos estudantes através de um questionário final (adaptado de Renkl (2002)). Na primeira parte deste questionário, os estudantes dos GC, GI1 e GI2 foram perguntados sobre as suas opiniões a respeito da utilidade e qualidade dos exemplos trabalhados e das explicações apresentados pela professora nos vídeos. Analisando os dados apresentados na Tabela 6.4, percebemos que os participantes de todos os grupos, GC, GI1 e GI2, tiveram uma opinião positiva sobre a qualidade e utilidade dos exemplos e das explicações que foram fornecidas nos exemplos em vídeo.

Tabela 6.4. Opiniões dos GC, GI1 e GI2 sobre os exemplos em vídeo.

		#de estudantes				
		DP ^a	D ^b	NN ^c	C ^d	CP ^e
Os exemplos e as explicações apresentados nos vídeos foram úteis para o meu aprendizado.	GC				1	2
	GI1				1	1
	GI2				1	2
Os exemplos e as explicações apresentados nos vídeos não me ajudaram no entendimento do assunto.	GC	1	2			
	GI1	2				
	GI2	1	2			
As explicações apresentadas nos vídeos foram	GC				1	2

detalhadas o suficiente para o entendimento do exemplo.	GI1				1	1
	GI2				1	2
As explicações apresentadas nos vídeos foram insuficientes para o entendimento do exemplo.	GC	2	1			
	GI1	1	1			
	GI2	2	1			

(a) Discordo plenamente (b) Discordo (c) Nem concordo, nem discordo (d) Concordo (e) Concordo plenamente

Fonte: própria autora.

Além disso, na segunda parte deste questionário, apresentada somente para os grupos instrucionais, os estudantes opinaram sobre a utilidade e qualidade das questões de auto-explicação presentes nos exemplos trabalhados em vídeo. Analisando os dados apresentados na Tabela 6.5, percebemos que a maioria dos participantes do GI1 e GI2 teve uma opinião positiva a respeito da qualidade e utilidade das questões empregadas para promover a auto-explicação dos vídeos.

Tabela 6.5. Opiniões dos GI1 e GI2 sobre as questões para AE nos exemplos em vídeo.

		#de estudantes				
		DP^a	D^b	NN^c	C^d	CP^e
As questões propostas eram claras e objetivas.	GI1					2
	GI2			1	2	
As questões propostas eram ambíguas e eu não consegui entendê-las por completo.	GI1	2				
	GI2	1	1	1		
As questões foram úteis durante o estudo do exemplo que foi apresentado no vídeo.	GI1					2
	GI2				1	2
As questões me ajudaram a entender o exemplo que foi apresentado no vídeo.	GI1				1	1
	GI2				2	1
As questões me atrapalharam no estudo do exemplo apresentado no vídeo.	GI1	2				
	GI2	2	1			

(a) Discordo plenamente (b) Discordo (c) Nem concordo, nem discordo (d) Concordo (e) Concordo plenamente

Fonte: própria autora.

6.2.1.3.2 Avaliação dos professores

Além da avaliação com os estudantes, realizamos também uma avaliação de uma amostra do material instrucional com dois professores. A amostra avaliada consistiu de metade do material instrucional, ou seja, de um dos dois exemplos em vídeo juntamente com as perguntas para auto-explicação correspondentes. O primeiro professor (P1) tem doutorado em Ciência da Computação e experiência de 19 anos de ensino em disciplinas como Algoritmos, Introdução à lógica, Introdução à programação e Projeto e análise de algoritmos. O segundo professor (P2) tem doutorado na área de Tecnologias Educacionais e experiência de 2.5 anos de ensino em disciplinas de Interação humano-computador e Informática na educação. As respostas fornecidas pelos professores são apresentadas na Tabela 6.6. Nesta tabela podemos perceber que os exemplos em vídeo e as perguntas para auto-explicação foram avaliadas positivamente pelos professores.

Tabela 6.6. Opiniões dos professores sobre o material instrucional.

	DP ^a	D ^b	NN ^c	C ^d	CP ^e
O exemplo é útil para o aprendizado de alunos iniciantes em programação, no contexto do objetivo de aprendizagem em questão.					P1, P2
O enunciado e a especificação do exemplo foram apresentados de maneira clara e objetiva.				P1, P2	
A sequência de passos da solução do exemplo é adequada ao problema em questão.					P1, P2
As explicações apresentadas para os passos de solução são claras e objetivas.				P1, P2	
A utilização dos aspectos visuais de vídeo (i.e., ênfase, seleção, destaque e zoom) durante a apresentação da solução são adequadas aos momentos em que elas ocorrem.				P1, P2	
O som e a imagem do vídeo são de boa qualidade.				P1, P2	
As questões incluídas no vídeo possuem um enunciado claro e objetivo.			P2		P1
As questões incluídas nos vídeos são úteis para o entendimento dos exemplos apresentados.			P2		P1
As questões são apresentadas em uma sequência adequada às diferentes seções do vídeo.					P1, P2
As questões são adequadamente alinhadas ao conteúdo apresentado.				P1	P2

(a) Discordo plenamente (b) Discordo (c) Nem concordo, nem discordo (d) Concordo (e) Concordo plenamente

Fonte: própria autora.

6.2.2 Segundo quase-experimento

Os resultados obtidos a partir dos dados coletados durante as 4 aulas extras são apresentados nas sub-seções a seguir.

6.2.2.1 Perfil dos participantes

O GI contou com 5 participantes ao longo das 4 aulas extras, no entanto conseguimos coletar todos os dados para apenas 2 deles. Sendo assim, tivemos para análise dos dados de 2 estudantes um com idade de 21 anos (E1) e outro com idade de 31 anos (E2), ambos do sexo masculino. E1 estava cursando a disciplina introdutória de programação pela quarta vez e E2 pela terceira vez. Quando perguntados sobre os seus hábitos de estudo para a disciplina de programação, os estudantes deram as respostas exibidas na Tabela 6.7.

Tabela 6.7. Como os estudantes do GI estudam para a disciplina de IP.

Quando eu estou estudando para a disciplina de IP, eu normalmente:	N ^a	R ^b	AV ^c	F ^d	S ^e
Somente assisto as aulas do professor, não estudo fora de sala de aula.		E2	E1		
Estudo a partir dos materiais (slides, livros, etc.) fornecidos pelo professor.				E1, E2	

Estudo a partir das anotações que eu fiz em sala de aula.			E2	E1	
Copio somente os programas que o professor faz em sala de aula, sem procurar entender o que foi feito.		E1, E2			
Estudo os exemplos e/ou exercícios que foram vistos em sala de aula, tentando reproduzi-los para entendê-los melhor.				E1	E2
Faço os exercícios recomendados pelo professor.			E1	E2	
Pesquiso por novos materiais, exemplos e exercícios que possam me auxiliar no entendimento do assunto.				E1, E2	
Tiro dúvidas com os meus colegas e/ou monitores da disciplina.			E1	E2	

(a) Nunca (b) Raramente (c) As vezes (d) Frequentemente (e) Sempre

Fonte: própria autora.

Quando perguntados sobre a utilização de vídeos para estudar para a disciplina de IP, apenas um deles relatou ter usado vídeos do Youtube para este fim. Ele utilizou os vídeos conforme apresentado na Tabela 6.8.

Tabela 6.8. Como um dos estudantes do GI lembra de ter utilizado vídeos para estudar para a disciplina de IP.

Como você se lembra de ter utilizado esses vídeos?	N ^a	R ^b	AV ^c	F ^d	S ^e
Eu pausei, avancei e/ou voltei o vídeo para prestar atenção em partes específicas dele.				E1	
Eu assisti ao mesmo vídeo (ou partes dele) mais de uma vez.				E1	
Eu tentei reproduzir o que era feito no vídeo.				E1	
Eu fiz anotações enquanto assistia ao vídeo.		E1			
Eu respondi perguntas apresentadas no vídeo enquanto assistia ao vídeo.			E1		
Eu assisti ao vídeo somente, sem fazer nenhum tipo de atividade.		E1			

(a) Nunca (b) Raramente (c) As vezes (d) Frequentemente (e) Sempre

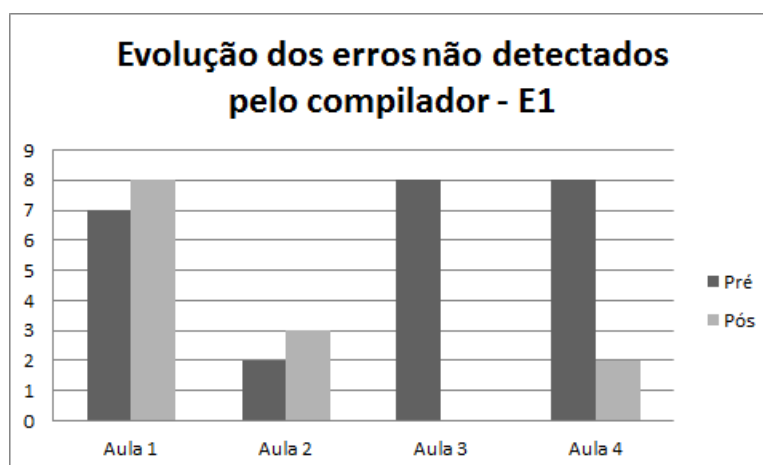
Fonte: própria autora.

6.2.2.2 Desempenho dos participantes

Para responder a [QP1] *Como foi a evolução do aprendizado de programação dos estudantes a partir da AE de exemplos trabalhados em vídeo ao longo dos momentos de estudo durante a execução deste experimento?*, examinamos os programas construídos pelos estudantes como respostas aos pré e pós-teste aplicados nas 4 aulas extras. Para análise destes programas, utilizamos a análise apresentada na Seção 5.2.4. Na Figura 6.8, a evolução dos erros não gerados pelo compilador para o E1 para os pré-testes e pós-testes realizados ao longo das 4 aulas extras. Nesta figura, podemos perceber que nas 2 primeiras aulas, o E1 teve o acréscimo de 1 erro do pré para o pós-

teste realizado. Podemos perceber também que, nas aulas 3 e 4, ele teve um decréscimo na quantidade de erros, saindo de 8 erros no pré-teste nestas duas aulas para 0 e apenas 2 erros no pós-teste, respectivamente. Sendo assim, na média das quatro aulas extras, o E1 teve um decréscimo de 6.25 erros por pré-teste para 3.25 erros por pós-teste, o que representa uma melhoria dos pré-testes para os pós-testes.

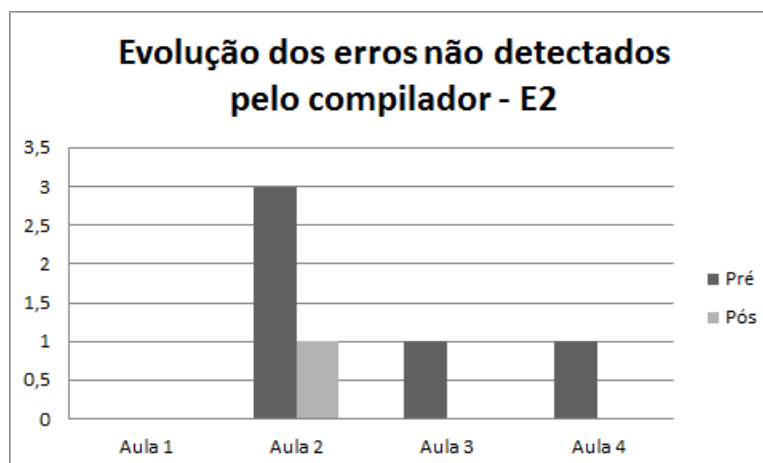
Figura 6.8. Evolução dos erros não gerados pelo compilador pelo E1 nas 4 aulas extras.



Fonte: própria autora.

Similarmente, apresentamos na Figura 6.9, a evolução dos erros não gerados pelo compilador para o E2 para os pré e pós-testes realizados ao longo das 4 aulas extras. Nesta figura, podemos constatar que em todas as aulas, o E2 teve desempenho do pós-teste sempre melhor ou igual ao pré-teste. Na primeira aula, vemos que o E2 não cometeu nenhum erro no pré-teste e no pós-teste da primeira aula extra. Nas outras três aulas, vemos que o número de erros do pós-teste é sempre menor do que o do pré-teste. Na aula 2, percebemos uma redução de 3 erros no pré-teste para 1 erro no pós-teste, enquanto nas aulas 3 e 4, constatamos uma redução de 2 erros no pré-teste para nenhum erro no pós-teste. Sendo assim, na média das quatro aulas extras, o E2 teve um decréscimo de 1.25 erros por pré-teste para 0.25 erros por pós-teste, o que também representa uma melhoria dos pré-testes para os pós-testes.

Figura 6.9. Evolução dos erros não gerados pelo compilador pelo E2 nas 4 aulas extras.



Fonte: própria autora.

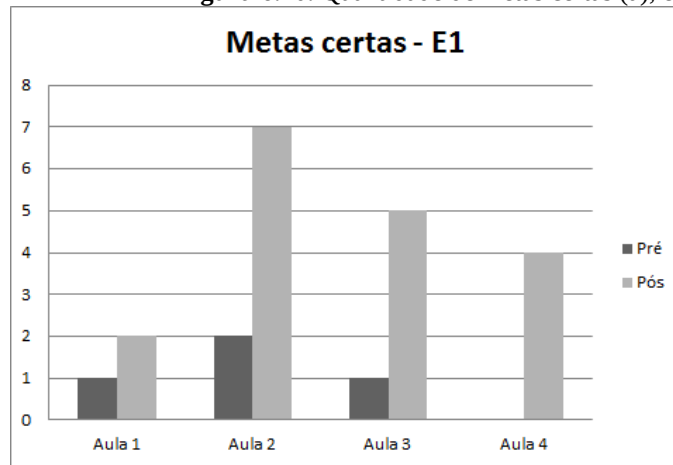
Como dito anteriormente, com base nos erros que foram cometidos pelos estudantes, classificamos as metas como certas, erradas e omitidas. Nas Figura 6.8 e Figura 6.9 apresentamos a quantidade destas metas para E1 e E2, respectivamente. Na Figura 6.10 (a) podemos perceber que E1 teve uma quantidade de metas certas maior para os pós-testes do que para os pré-testes em todas as aulas. A maior diferença no número de metas certas entre o pré e o pós-teste pode ser percebida na aula 2, quando E1 acerta apenas 2 metas no pré-teste e passa a acertar 7 metas no pós-teste. A menor diferença é percebida na aula 1, quando E1 acerta apenas 1 meta no pré-teste e passa a acertar 2 no pós-teste. Na Figura 6.10 (b) constatamos que E1 teve uma quantidade de metas erradas para os pós-testes sempre menor ou igual do que para os pré-testes em todas as aulas. A maior diferença pode ser notada na aula 4 onde ele errou 5 metas no pré-teste e passou a errar apenas 1 no pós-teste, enquanto a menor diferença foi na aula 1 onde ele errou 4 metas tanto no pré-teste quanto no pós-teste. Na Figura 6.10 (c) notamos que E1 deixou de omitir metas do pré-teste para o pós-teste. A maior diferença de metas omitidas é na aula 2, quando no pré-teste ele não implementa 4 metas e a menor diferença é na aula 4, quando ele não omite metas tanto no pré quanto no pós-teste. Em suma, o E1 teve na média dos testes: (i) um acréscimo de 1 meta certa no pré-teste para 4.5 metas certas no pós-teste; (ii) um decréscimo de 3.5 metas erradas no pré-teste para 1.5 meta errada no pós-teste; e (iii) uma redução de 1.5 meta omitida no pré-teste para nenhuma meta omitida no pós-teste.

Na Figura 6.11 (a) podemos notar que E2 teve uma quantidade de metas certas para os pós-testes sempre maior ou igual do que para os pré-testes em todas as aulas. A

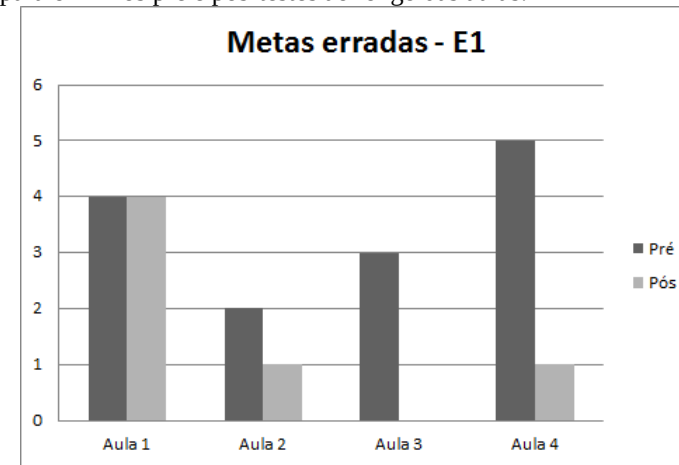
menor diferença na quantidade de metas certas ocorreu na aula 1, quando ele acertou as 6 metas do programa tanto no pré quanto no pós-teste. Já a maior diferença foi na aula 2, quando ele acertou 4 metas no pré-teste e passou a acertar 7 metas do programa no pós-teste. Ao contrário do que aconteceu anteriormente, na Figura 6.11 (b) constatamos que E2 teve uma quantidade de metas erradas para os pós-testes sempre menor ou igual do que para os pré-testes em todas as aulas. A menor diferença na quantidade de metas erradas ocorreu na aula 1, quando ele não errou nenhuma das metas do programa tanto no pré quanto no pós-teste. Já a maior diferença foi na aula 2, quando ele errou 3 metas no pré-teste e passou a errar apenas 1 meta no pós-teste. Na Figura 6.11 (c) podemos notar que E2 omitiu muito poucas metas. A única omissão ocorrida foi na aula 2, quando ele omitiu 1 meta no pré-teste. Em suma, o E2 teve na média dos testes: (i) um acréscimo de 4.5 metas certas no pré-teste para 5.75 metas certas no pós-teste; (ii) um decréscimo de 1.25 metas erradas no pré-teste para 0.25 meta errada no pós-teste; e (iii) um decréscimo de 0.25 meta omitida no pré-teste para nenhuma meta omitida no pós-teste.

Após classificarmos as metas como erradas, certas ou omitidas, de acordo com o que foi descrito na Seção 5.2.4, computamos quantitativamente os pré e pós-testes para cada um dos quase-experimentos. Ao final deste segundo quase-experimento, cada estudante poderia para o pré e o pós-teste da primeira, segunda, terceira e quarta aulas, uma pontuação máxima de 9.5, 12, 10.5 e 9.5 pontos cada, respectivamente. Tomando por base esta pontuação, as notas obtidas pelos estudantes no pré-teste e pós-teste em cada uma das 4 aulas são exibidas na Figura 6.12. Nesta figura, podemos perceber que ambos os estudantes obtiveram notas nos pós-testes que foram iguais ou superiores às notas obtidas no pré-teste, indicando que houve ganho de aprendizado em todas as aulas após o estudo dos exemplos em vídeo. Podemos notar também que esse ganho foi mais expressivo para o E1 que chegou a obter uma diferença máxima de 9.5 pontos do pré para o pós-teste na aula 3 e uma diferença mínima de 1 ponto do pré para o pós-teste na aula 1. Por outro lado, a maior diferença para o E2 foi de 5 pontos do pré para o pós-teste na aula 2 e a menor de 0 pontos na aula 1. Na média dos testes, o E1 saiu de uma média de 1.12 ponto no pré-teste para uma média de 7.37 pontos no pós-teste. O E2 teve uma melhora na média de seu desempenho, saindo de uma média de 7.62 pontos no pré-teste para uma média de 10 pontos no pós-teste.

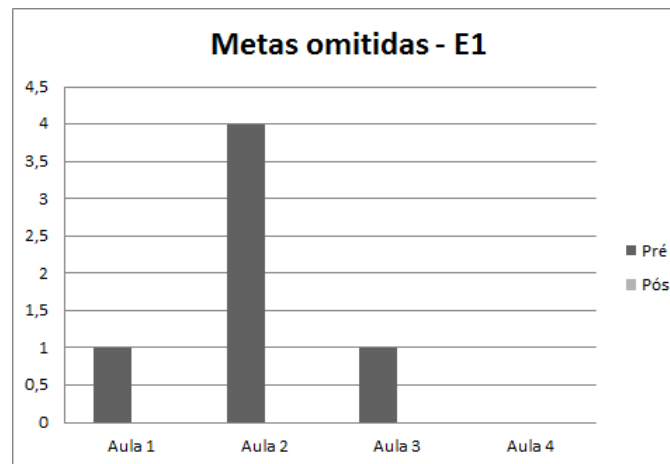
Figura 6.10. Quantidade de metas certas (a), erradas (b) e omitidas (c) para o E1 nos pré e pós-testes ao longo das aulas.



(a)



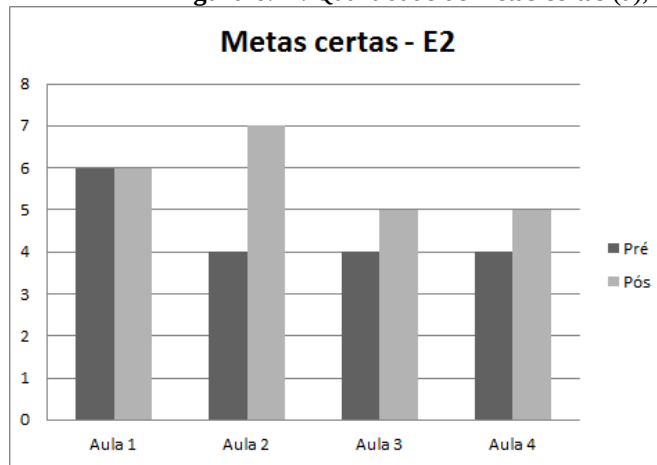
(b)



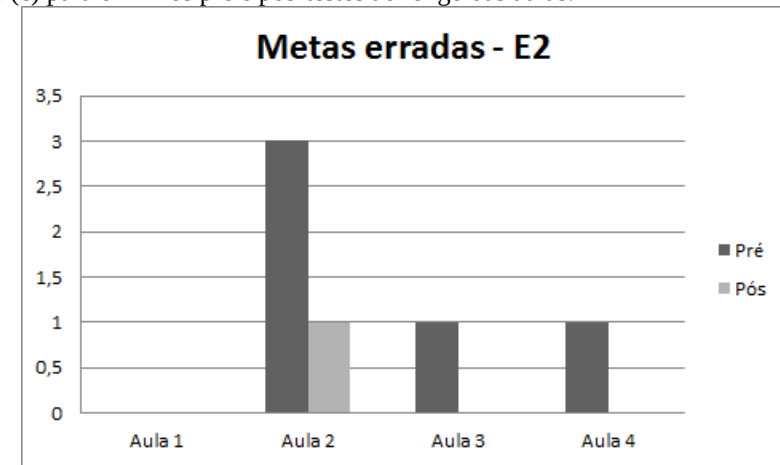
(c)

Fonte: própria autora.

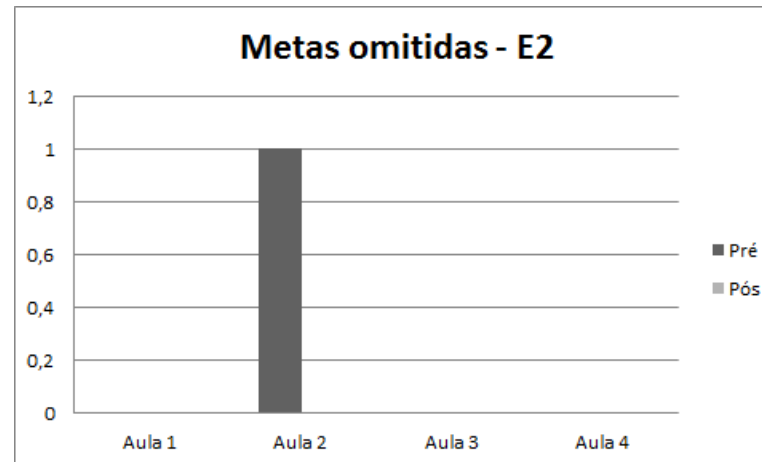
Figura 6.11. Quantidade de metas certas (a), erradas (b) e omitidas (c) para o E2 nos pré e pós-testes ao longo das aulas.



(a)



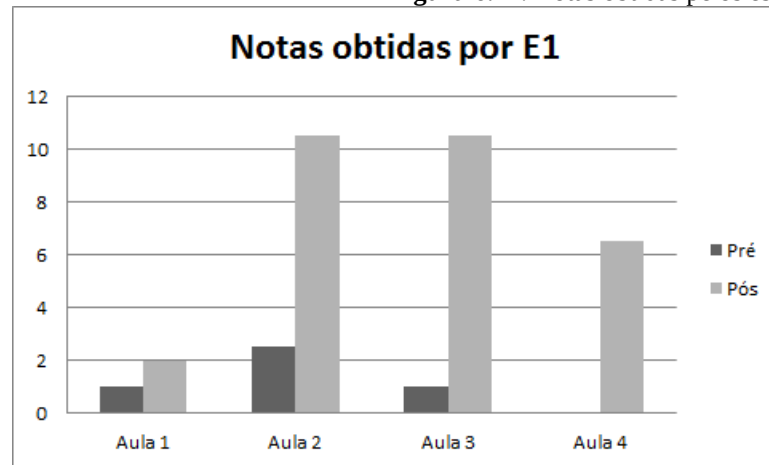
(b)



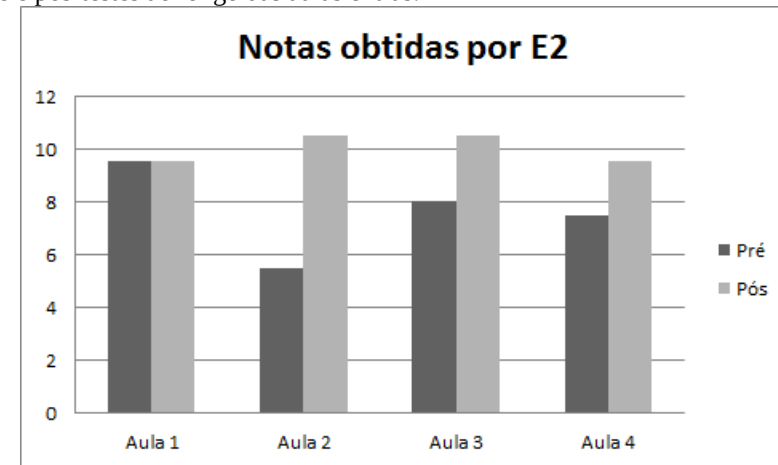
(c)

Fonte: própria autora.

Figura 6.12. Notas obtidas pelos estudantes nos pré e pós-testes ao longo das aulas extras.



(a)



(b)

Fonte: própria autora.

Para responder a [QP2] *Existe alguma correlação entre as respostas às questões de auto-explicação, o tempo de estudo dos exemplos em vídeos juntamente com as questões de auto-explicação e os resultados obtidos no pós-teste em cada um dos grupos instrucionais?*, examinamos as respostas dadas às perguntas de auto-explicação durante os momentos de estudo dos exemplos nas 4 aulas extras ministradas ao GI, bem como o tempo que os estudantes levaram para estudar os exemplos nestas aulas. Com base nos valores dados a cada uma das respostas, o valor máximo que cada estudante poderia obter em seus momentos de estudo foi de 12, 14, 13 e 15 pontos na primeira, segunda, terceira e quarta aulas, respectivamente. As notas obtidas pelos estudantes no estudo dos vídeos foram normalizadas, isto é, divididas pela pontuação máxima que os estudantes podiam obter nas respostas às perguntas de cada um dos exemplos em vídeo e depois multiplicadas por 10 para que ficassem compreendidas entre 0 e 10 pontos. São estas notas normalizadas que serão utilizadas daqui por diante nesta análise. As notas fornecidas às questões de auto-explicação em cada aula para o GI variaram de 4 a 8.2 pontos, com mediana de 6.42 e média de 6.55 pontos (DP=1.47). Os tempos de estudo em cada aula para o GI1 variaram de 31m43s a 50m29s, com mediana de 41m53s e média de 42m18s (DP=6m55s). Para saber se houve uma relação entre as notas obtidas nas respostas escritas às perguntas de estímulo a auto-explicação, o tempo de estudo dos exemplos e as notas do pós-teste no GI, utilizamos o coeficiente de correlação de Spearman. Neste caso, encontramos uma correlação significativa forte entre o tempo de estudo dos exemplos e as notas do pós-teste ($r=0.73$; $p=0.02$). No entanto, não encontramos uma correlação significativa entre: (i) as notas obtidas nas respostas escritas às perguntas de estímulo a auto-explicação e o tempo de estudo dos exemplos ($r=-0.31$; $p=0.23$); e (ii) as notas obtidas nas respostas escritas às perguntas de estímulo a auto-explicação e as notas do pós-teste ($r=0.13$; $p=0.49$).

6.2.2.3 Avaliação do material instrucional

O material instrucional deste experimento também foi avaliado pelos participantes do experimento e por professores de diferentes áreas. Os resultados obtidos a partir destas avaliações são apresentados nas seções a seguir.

6.2.2.3.1 Avaliação dos estudantes

De maneira similar ao primeiro experimento, na fase final do segundo experimento, coletamos o *feedback* dos estudantes utilizando o mesmo um questionário

final (adaptado de Renkl (2002)) utilizado anteriormente. Na primeira parte deste questionário, os estudantes do GI foram perguntados sobre as suas opiniões a respeito da utilidade e qualidade dos exemplos trabalhados e das explicações apresentados pelo professor nos vídeos. Analisando os dados apresentados na Tabela 6.9, percebemos que os participantes tiveram uma opinião positiva a respeito da qualidade e utilidade dos exemplos e das explicações que foram fornecidas nos vídeos.

Tabela 6.9. Opiniões do GI sobre os exemplos em vídeo.

	DP^a	D^b	NN^c	C^d	CP^e
Os exemplos e as explicações apresentados nos vídeos foram úteis para o meu aprendizado.					E1, E2
Os exemplos e as explicações apresentados nos vídeos não me ajudaram no entendimento do assunto.	E1, E2				
As explicações apresentadas nos vídeos foram detalhadas o suficiente para o entendimento do exemplo.					E1, E2
As explicações apresentadas nos vídeos foram insuficientes para o entendimento do exemplo.	E1, E2				

(a) Discordo plenamente (b) Discordo (c) Nem concordo, nem discordo (d) Concordo (e) Concordo plenamente

Fonte: própria autora.

Além disso, na segunda parte deste questionário, os estudantes opinaram sobre a utilidade e qualidade das questões de auto-explicação presentes nos exemplos trabalhados em vídeo. Analisando os dados apresentados na Tabela 6.10, percebemos que E1 e E2 tiveram uma opinião positiva a respeito da qualidade e utilidade das questões empregadas para promover a auto-explicação dos vídeos.

Tabela 6.10. Opiniões do GI sobre as questões para AE nos exemplos em vídeo.

	DP^a	D^b	NN^c	C^d	CP^e
As questões propostas eram claras e objetivas.				E1	E2
As questões propostas eram ambíguas e eu não consegui entendê-las por completo.	E2	E1			
As questões foram úteis durante o estudo do exemplo que foi apresentado no vídeo.					E1, E2
As questões me ajudaram a entender o exemplo que foi apresentado no vídeo.					E1, E2
As questões me atrapalharam no estudo do exemplo apresentado no vídeo.	E1, E2				

(a) Discordo plenamente (b) Discordo (c) Nem concordo, nem discordo (d) Concordo (e) Concordo plenamente

Fonte: própria autora.

6.2.2.3.2 Avaliação dos professores

Além da avaliação com os estudantes, realizamos também uma avaliação de uma amostra do material instrucional com dois professores. Esta amostra consistiu de metade do material instrucional, ou seja, de dois dos quatro exemplos em vídeos juntamente

com as perguntas para auto-explicação correspondentes utilizados nas aulas extras. O primeiro professor (P1) tem doutorado em Ciência da Computação com ênfase em Visão Computacional e experiência de 2 anos de ensino em disciplinas como Introdução à programação e Programação paralela. O segundo professor (P2) tem mestrado em Ciência da Computação com ênfase em Pensamento Computacional e experiência de 1.5 ano de ensino em disciplinas de Informática para o ensino médio. As respostas fornecidas pelos professores são apresentadas na Tabela 6.11. Nesta tabela, podemos perceber que os exemplos em vídeo e as perguntas para auto-explicação foram avaliadas positivamente por ambos os professores.

Tabela 6.11. Opiniões dos professores a respeito do material instrucional.

	DP^a	D^b	NN^c	C^d	CP^e
O exemplo é útil para o aprendizado de alunos iniciantes em programação, no contexto do objetivo de aprendizagem em questão.					P1, P2
O enunciado e a especificação do exemplo foram apresentados de maneira clara e objetiva.				P2	P1
A sequência de passos da solução do exemplo é adequada ao problema em questão.					P1, P2
As explicações apresentadas para os passos de solução são claras e objetivas.					P1, P2
A utilização dos aspectos visuais de vídeo (i.e., ênfase, seleção, destaque e zoom) durante a apresentação da solução são adequadas aos momentos em que elas ocorrem.					P1, P2
O som e a imagem do vídeo são de boa qualidade.				P2	P1
As questões incluídas no vídeo possuem um enunciado claro e objetivo.					P1, P2
As questões incluídas nos vídeos são úteis para o entendimento dos exemplos apresentados.					P1, P2
As questões são apresentadas em uma sequência adequada às diferentes seções do vídeo.					P1, P2
As questões são adequadamente alinhadas ao conteúdo apresentado.					P1, P2

(a) Discordo plenamente (b) Discordo (c) Nem concordo, nem discordo (d) Concordo (e) Concordo plenamente

Fonte: própria autora.

6.2.3 Terceiro quase-experimento

Os resultados³³ obtidos a partir dos dados coletados durante as duas aulas extras são apresentados nas seções a seguir.

³³ Os resultados apresentados aqui nesta seção são um refinamento dos resultados apresentados anteriormente em Aureliano e Tedesco (2015).

6.2.3.1 Perfil dos participantes

Dentre os 12 participantes, 9 haviam concluído o ensino médio e cursavam o ensino técnico profissionalizante e 3 cursavam o ensino médio integrado ao técnico profissionalizante. Além disso, os 9 participantes que concluíram o ensino médio eram todos da mesma turma e estavam cursando uma disciplina introdutória de LP em C. O fato da maioria dos participantes estar cursando uma disciplina introdutória de LP não foi um impedimento à realização do experimento, visto que todos estes 9 participantes relataram sentir muitas dificuldades no conteúdo que foi ministrado até o momento da realização do experimento (o que foi comprovado posteriormente com o pré-teste realizado). Os outros 3 participantes, que ainda cursavam o ensino médio, não haviam cursado nenhuma disciplina relacionada à LP.

O GC contou inicialmente com 6 participantes, no entanto, por problemas técnicos na coleta de dados de um deles, esta análise conta com os dados de somente 5 participantes. As idades deles variavam de 16 a 18 anos (média=17.2 anos), 3 do sexo masculino e 2 do sexo feminino. Todos eles já haviam concluído o ensino médio e estavam cursando o ensino técnico profissionalizante. Além disso, eles estavam cursando uma disciplina introdutória de programação em C, porém nenhum deles tinha experiência com o *Scratch*. Por estarem matriculados em uma disciplina de programação, os participantes foram perguntados sobre os seus hábitos de estudo para esta disciplina. As respostas obtidas são apresentadas na Tabela 6.12. Além disso, todos os participantes relataram ter utilizado vídeos disponíveis no Youtube para estudar para a disciplina de LP. Estes vídeos foram utilizados da maneira descrita na Tabela 6.13.

Tabela 6.12. Como os estudantes dos GC e GI1 estudam para a disciplina de IP e como os estudantes do GI2 estudam para as disciplinas em geral.

Quando eu estou estudando para a disciplina de IP [para as disciplinas em geral], eu normalmente:		#de estudantes				
		N ^a	R ^b	AV ^c	F ^d	S ^e
Somente assisto as aulas do professor, não estudo fora de sala de aula.	GC			3	1	1
	GI1			3		1
	GI2		1		2	
Estudo a partir dos materiais (slides, livros, etc.) fornecidos pelo professor.	GC	1	1	1	2	
	GI1			1	3	
	GI2			2		1
Estudo a partir das anotações que eu fiz em sala de aula.	GC	2	1			2
	GI1			1	2	1
	GI2			1	1	1
Copio somente os exemplos e/ou exercícios que o professor faz em sala de aula, sem procurar	GC		3	1	1	
	GI1	2	1	1		

entender o que foi feito.	GI2	3				
Estudo os exemplos e/ou exercícios que foram vistos em sala de aula, tentando reproduzi-los para entendê-los melhor.	GC		3		2	
	GI1			1	2	1
	GI2				1	2
Faço os exercícios recomendados pelo professor.	GC	2			2	1
	GI1		1	1	2	
	GI2			2		1
Pesquisa por novos materiais, exemplos e exercícios que possam me auxiliar no entendimento do assunto.	GC		1		2	2
	GI1				3	1
	GI2			2	1	
Tiro dúvidas com os meus colegas e/ou monitores da disciplina.	GC			1	3	1
	GI1					4
	GI2					3

(a) Nunca (b) Raramente (c) As vezes (d) Frequentemente (e) Sempre

Fonte: própria autora.

O GI contou com 8 participantes, no entanto, por problemas técnicos na coleta de dados de um deles, os dados de somente 7 participantes puderam ser aproveitados nesta análise. As idades deles variavam de 15 a 19 anos (média=17 anos), 6 do sexo masculino e 1 do sexo feminino. Este grupo englobou participantes de dois perfis diferentes, o GI1 com participantes que já haviam concluído o ensino médio e estavam matriculados na disciplina introdutória de programação em C, e o GI2 com alunos que estavam cursando o ensino médio integrado ao ensino técnico e não haviam cursado nenhuma disciplina de programação. Nenhum dos participantes tinha experiência com o *Scratch*. Todos os participantes foram perguntados sobre os seus hábitos de estudo; os 4 participantes do GI1 responderam como estudavam para a disciplina de programação na qual estavam matriculados, enquanto os do GI2 responderam como estudavam para as disciplinas em geral. As respostas obtidas são apresentadas na Tabela 6.12. Além disso, 5 participantes relataram ter utilizado vídeos disponíveis no Youtube para estudar para a disciplina de programação (GI1) ou para estudar quaisquer outras disciplinas (GI2). Estes vídeos foram utilizados da maneira descrita na Tabela 6.13.

Tabela 6.13. Como os estudantes dos GC e GI1 lembram de ter utilizado vídeos para estudar para a disciplina de IP e como os estudantes do GI2 lembram de ter utilizado vídeos para as disciplinas em geral.

Como você se lembra de ter utilizado esses vídeos?		#de estudantes				
		N ^a	R ^b	AV ^c	F ^d	S ^e
Eu pausei, avancei e/ou voltei o vídeo para prestar atenção em partes específicas dele.	GC			1	2	2
	GI1				1	1
	GI2			2	1	
Eu assisti ao mesmo vídeo (ou partes dele) mais de uma vez.	GC		2		1	2
	GI1				2	
	GI2		1	2		
Eu tentei reproduzir o que era feito no vídeo.	GC	1			2	2
	GI1		1	1		

	GI2				3	
Eu fiz anotações enquanto assistia ao vídeo.	GC		3		1	1
	GI1				2	
	GI2			1	1	1
Eu respondi perguntas apresentadas no vídeo enquanto assistia ao vídeo.	GC	1		2	1	1
	GI1			1		1
	GI2				1	2
Eu assisti ao vídeo somente, sem fazer nenhum tipo de atividade.	GC	3		1	1	
	GI1			2		
	GI2		3			

(a) Nunca (b) Raramente (c) Às vezes (d) Frequentemente (e) Sempre

Fonte: própria autora.

6.2.3.2 Desempenho dos participantes

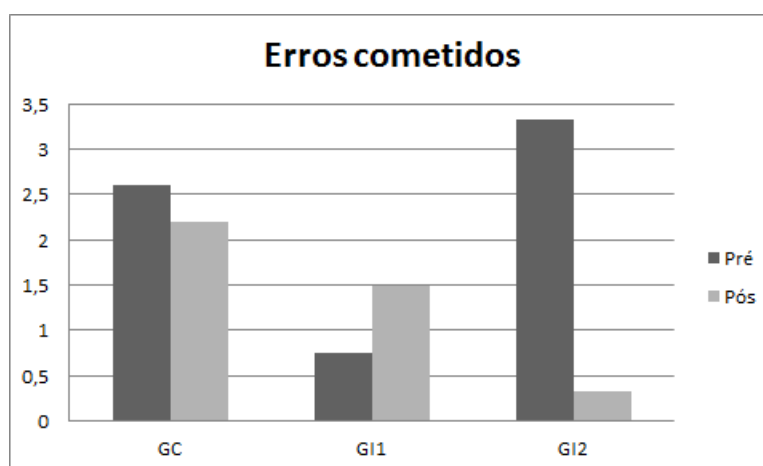
Para responder a [QP1] *O aprendizado de programação através da AE de exemplos em vídeo (i.e., do grupo instrucional) apresenta melhores resultados quando comparado ao aprendizado de programação somente através de exemplos em vídeo (i.e., do grupo controle)?*, examinamos os programas construídos pelos estudantes como respostas aos pré e pós-teste aplicados aos grupos durante o experimento. De forma análoga aos experimentos anteriores, para implementação de cada programa, procuramos por trechos de implementação que correspondiam a metas intermediárias identificadas para o programa. Em cada um destes trechos de implementação, procuramos por erros não sintáticos³⁴ que foram categorizados com uma versão estendida da classificação de erros cometidos por iniciantes em programação e proposta por Spohrer et al. (1985). Às categorias de erros já existentes (ausente, mal formado, desnecessário e posicionado incorretamente), foi acrescentada a categoria “inadequado” para ser utilizada quando uma parte do trecho de implementação para a meta foi utilizado no lugar de outro trecho de implementação. Como estamos utilizando o *Scratch* neste experimento, um erro inadequado ocorre normalmente quando um comando é utilizado no lugar de outro. Por exemplo, um erro inadequado é utilizar o comando para atribuir um valor a uma variável declarada no programa [`mude variável para valor`] no lugar do comando [`adicione a variável valor`] que acrescenta um valor à variável. Além disso, na segunda dimensão de classificação para os erros, adicionamos a categoria “comando” às categorias já existentes. Esta categoria comando é utilizada quando o erro é cometido com um

³⁴ Como utilizamos o Scratch, nos programas produzidos pelos estudantes neste experimento, não há erros ou avisos detectados pelo compilador. Por esta razão, começamos a análise destes programas da segunda etapa do processo de análise descrito na Seção 5.2.4.

comando por inteiro. Voltando ao exemplo do erro inadequado cometido anteriormente, este erro é classificado como erro de comando inadequado já que a inadequação se refere ao comando que não foi escolhido corretamente. Por fim, a terceira etapa da análise ocorreu normalmente.

Agora que já explicamos as adaptações ocorridas na análise dos programas construídos pelos estudantes para este experimento, vamos apresentar os resultados obtidos. As médias de erros cometidos pelos GC, GI1 e GI2 nos pré e pós-testes são apresentados na Figura 6.13. Nesta figura, podemos perceber que o GC e o GI2 diminuíram a quantidade de erros cometidos do pré para o pós-teste, sendo este último o que apresentou a maior queda no número de erros. O GC diminuiu a quantidade média de erros de 2.6 para 2.2 erros, já o GI2 diminuiu a sua quantidade média de erros de 3.33 para 0.33 erros. Ao contrário destes dois grupos, notamos que o GI1 teve um aumento na sua quantidade média de erros de 0.75 para 1.5, dobrando assim a sua quantidade de erros do pré para o pós-teste. Veremos mais a frente, no entanto, que o aumento na quantidade média de erros no GI1 não influenciou o desempenho geral deste grupo ao analisarmos a sua quantidade média de metas certas, erradas e omitidas, uma vez que esta quantidade de erros ficou concentrada em apenas poucas metas.

Figura 6.13. Médias dos erros cometidos pelos GC, GI1 e GI2 nos pré e pós-testes.



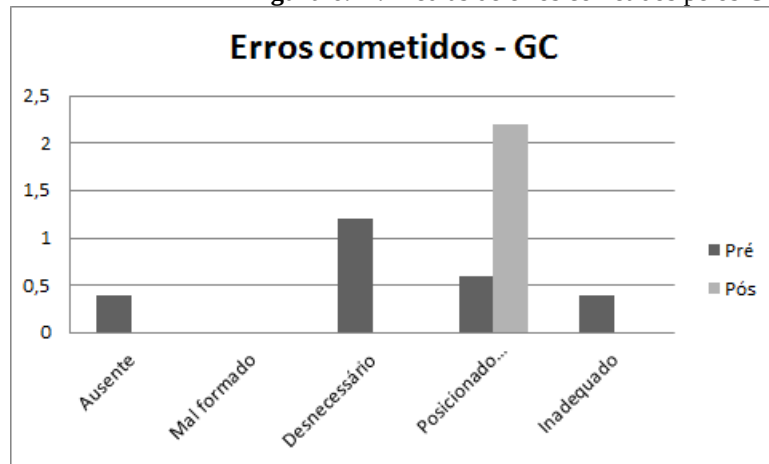
Fonte: própria autora.

Na Figura 6.14 são apresentadas as médias de erros cometidos pelos GC, GI1 e GI2 no pré e no pós-teste organizados pelas categorias de erros. Na Figura 6.14 (a) podemos perceber que as médias para os erros ausentes, desnecessários e inadequados diminuíram, respectivamente, de 0.4, 1.2 e 0.4 para 0 do pré para o pós-teste no GC. A média de erros posicionados incorretamente cresceu de 0.6 para 2.2 do pré para o pós-

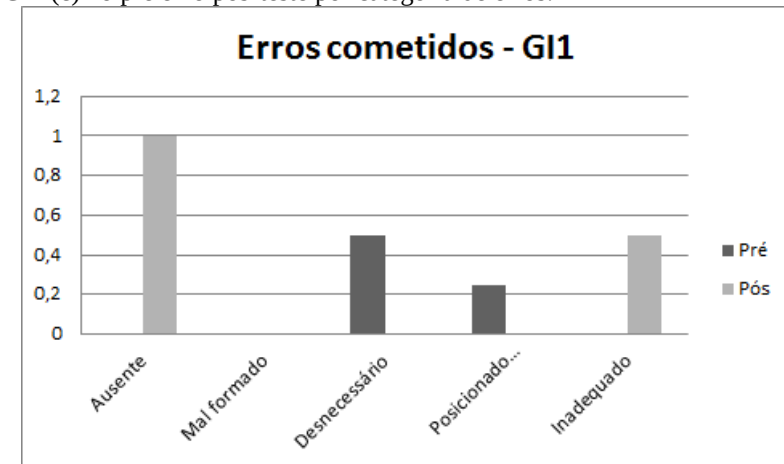
teste. Portanto, temos que os erros mais cometidos pelo GC no pré-teste foram os erros desnecessários, enquanto que no pós-teste foram os erros posicionados incorretamente. Na Figura 6.14 (b) notamos que a quantidade média para os erros ausentes e mal formados no GI1 aumentou de 0 para 1 e 0.5 do pré para o pós-teste, respectivamente. Notamos também que a quantidade média dos erros desnecessários e posicionados incorretamente diminuiu, respectivamente, de 0.5 e 0.25 para 0 do pré para o pós-teste. Sendo assim, temos que os erros mais cometidos pelo GC no pré-teste foram os erros inadequados, enquanto que no pós-teste foram os erros ausentes. Por fim, Na Figura 6.14 (c) constatamos que o GI2 cometeu apenas erros desnecessários e inadequados. A quantidade média dos erros desnecessários foi reduzida de 0.67 para 0, enquanto a quantidade média dos erros inadequados foi reduzida de 2.67 para 0.33 do pré para o pós-teste. Desse modo, tivemos que no GI2 os erros mais cometidos tanto no pré quanto no pós-teste foram os inadequados. Em nenhum dos grupos, houve alteração na quantidade média de erros mal formados.

Com base nos erros identificados, classificamos as metas como certas, erradas ou omitidas. Na Figura 6.15 (a) apresentamos a quantidade média de metas certas para cada um dos grupos. Nesta figura, percebemos que todos os grupos aumentaram a quantidade média de metas certas do pré para o pós-teste. O GI1 foi o grupo que teve a maior quantidade média de metas certas no pós-teste com 5.75 metas certas por estudante. O GI2 teve uma quantidade intermediária, com 4.67 metas certas por estudante no pós-teste. Já o GC foi o grupo que teve a menor quantidade de metas certas com 4.2 metas certas por estudante no pós-teste. Na Figura 6.15 (b) apresentamos a quantidade média de metas erradas para cada um dos grupos. Nela, percebemos que o GI2 foi o único grupo que teve a sua quantidade média de metas erradas reduzida de 1 para 0.33 do pré para o pós-teste. Ao contrário do GI2, o GC e o GI1 tiveram um acréscimo nas suas quantidades médias de metas erradas do pré para o pós-teste. O primeiro grupo de 0.8 para 1 meta errada, já o segundo grupo de 0.25 para 0.75 meta errada. Na Figura 6.15 (c) apresentamos a quantidade média de metas omitidas para cada um dos grupos. Nela, constatamos que os três grupos tiveram uma redução na quantidade média de metas omitidas do pré para o pós-teste. O GI1 foi o grupo que apresentou a maior redução, indo de 6.75 para 0.5 meta omitida. O GC teve uma redução intermediária, indo de 6.2 para 1.8 meta omitida por estudante. Por fim, o GI2 teve a menor redução, indo de 6 para 2 metas omitidas por estudante.

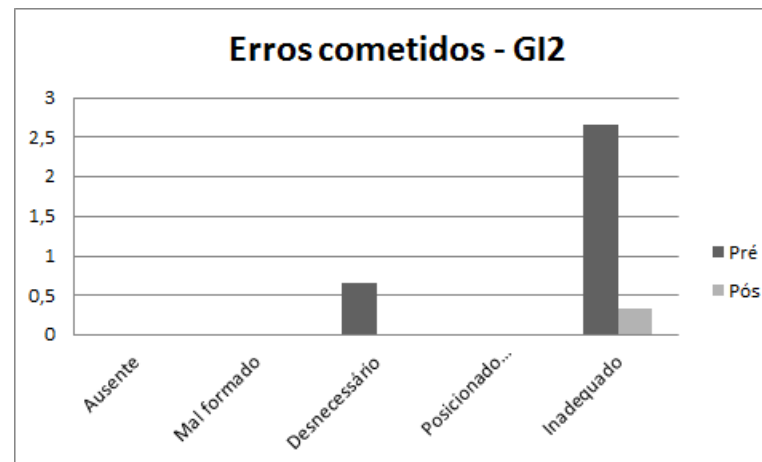
Figura 6.14. Médias de erros cometidos pelos GC (a), GI1 (b) e GI2 (c) no pré e no pós-teste por categoria de erros.



(a)



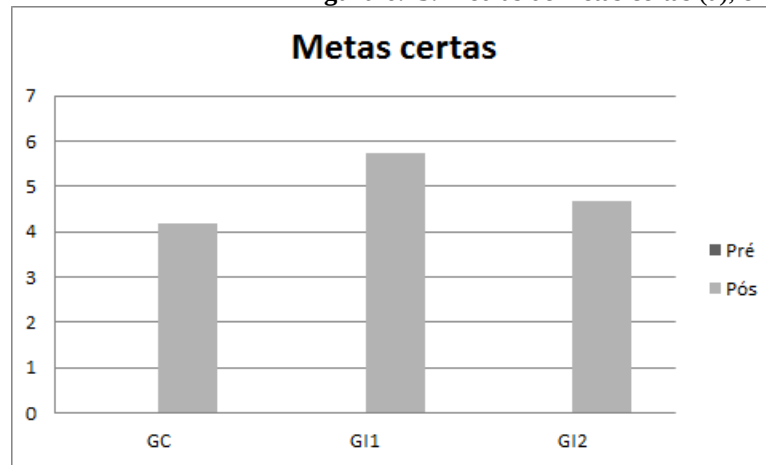
(b)



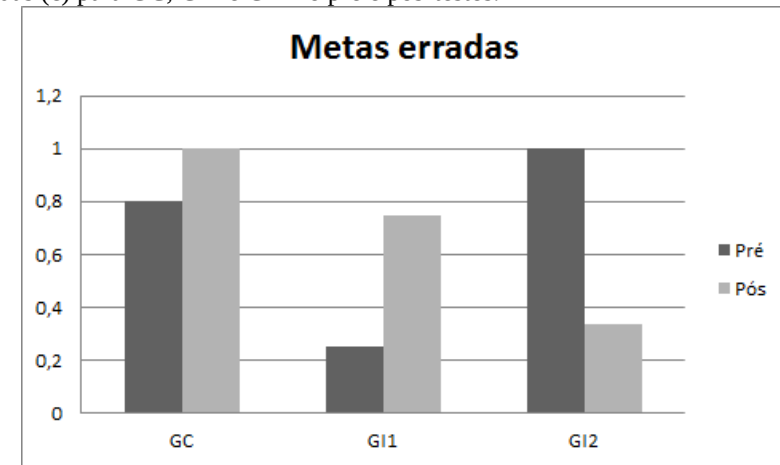
(c)

Fonte: própria autora.

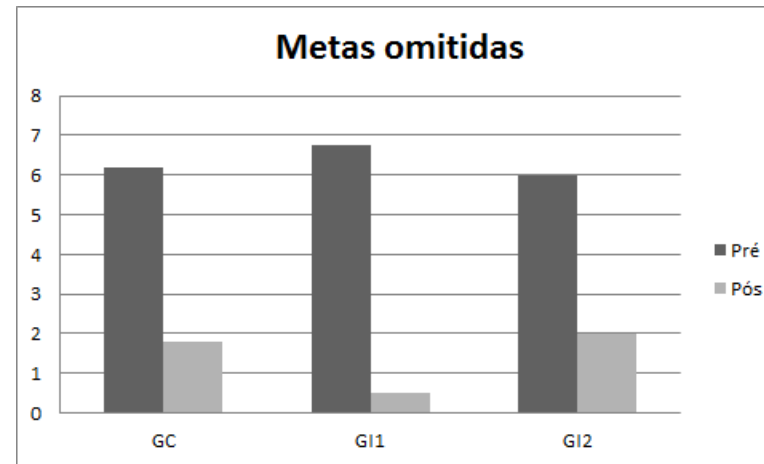
Figura 6.15. Médias de metas certas (a), erradas (b) e omitidas (c) para GC, GI1 e GI2 no pré e pós-testes.



(a)



(b)

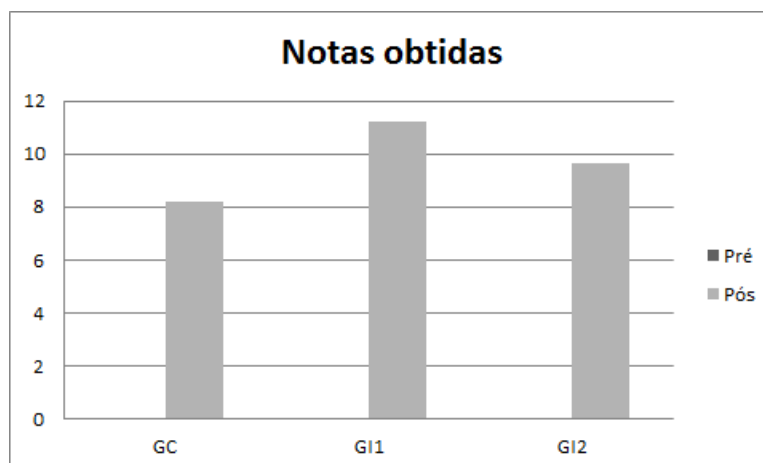


(c)

Fonte: própria autora.

Após classificarmos as metas como erradas, certas ou omitidas, de acordo com o que foi descrito na Seção 5.2.4, computamos quantitativamente os pré e pós-testes para cada um dos quase-experimentos. Ao final deste terceiro quase-experimento, cada estudante poderia para o pré-teste e o pós-teste tivemos uma pontuação máxima de 13.5 pontos cada. Tomando por base esta pontuação, as notas do pós-teste para o GC variaram entre 1 e 13.5, com mediana igual a 7.5 e média igual a 8 (DP=4.99). As notas do pós-teste para o GI1 variaram entre 8 e 13.5, com mediana igual a 11 e média igual a 10.87 (DP=3.04). As notas do pós-teste para o GI2 variaram entre 7.5 e 12.5, com mediana igual a 7.5 e média igual a 9.17 (DP=2.89). Comparando os dados do GC e do GI1, podemos perceber que GC e GI não apresentam uma diferença estatisticamente significativa em relação aos resultados do pós-teste ($U=5$; $z=-1.25$; $p=0.29$). Comparando os dados do GC e do GI2, podemos perceber que GC e GI2 não apresentam uma diferença estatisticamente significativa em relação aos resultados do pós-teste ($U=6$; $z=-0.46$; $p=0.79$). Comparando os dados do GI1 e do GI2, podemos perceber que GI1 e GI2 não apresentam uma diferença estatisticamente significativa em relação aos resultados do pré-teste ($U=2$; $z=-1.44$; $p=0.23$).

A média das notas obtidas pelos estudantes no pré-teste e pós-teste aplicados no decorrer do minicurso são exibidas na Figura 6.16. Nesta figura, podemos perceber que todos os grupos tiveram nota zero no pré-teste, uma vez que os estudantes não acertaram nenhuma meta antes do minicurso ser ministrado. Podemos perceber também que o grupo com a menor nota no pós-teste foi o GC que tirou 8 pontos. O GI2 teve uma nota intermediária com 9.17 pontos, uma nota quase 15% maior do que a nota do GC. Por fim, o grupo com a maior nota foi o GI1 que tirou 10.86 pontos, uma nota quase 36% maior que a nota do GC e um pouco mais de 18% maior do que a nota do GI2.

Figura 6.16. Notas obtidas pelos GC, GI1 e GI2 no terceiro experimento.

Fonte: própria autora.

Para responder a [QP2] *Existe alguma correlação entre as respostas às questões de auto-explicação, o tempo de estudo dos exemplos em vídeos juntamente com as questões de auto-explicação e os resultados obtidos no pós-teste em cada um dos grupos instrucionais?*, examinamos as respostas dadas às perguntas de auto-explicação durante os momentos de estudo dos 4 exemplos estudados pelo grupo instrucional, bem como o tempo que os estudantes levaram para estudar os exemplos durante o minicurso. Com base nos valores dados a cada uma das respostas, o valor máximo que cada estudante poderia obter em seus momentos de estudo foi de 42 pontos. As notas obtidas pelos estudantes no estudo dos vídeos foram normalizadas, isto é, divididas pela pontuação máxima que os estudantes podiam obter nas respostas às perguntas dos exemplos em vídeo e depois multiplicadas por 10 para que ficassem compreendidas entre 0 e 10 pontos. São estas notas normalizadas que serão utilizadas daqui por diante nesta análise. As notas fornecidas às questões de auto-explicação para o GI variaram de 8.57 a 9.52 pontos, com mediana de 9.29 e média de 9.14 pontos (DP=0.43). O tempo total de estudo dos exemplos em vídeo para o GI1 variaram de 49m06s a 1h08m26s, com mediana de 59m27s e média de 57m58s (DP=6m54s). Para saber se houve uma relação entre as notas obtidas nas respostas escritas às perguntas de estímulo a auto-explicação, o tempo de estudo dos exemplos e as notas do pós-teste no GI, utilizamos o coeficiente de correlação de Spearman. Neste caso, não encontramos uma correlação significativa entre: (i) as notas obtidas nas respostas às perguntas de estímulo a auto-explicação e o tempo de estudo dos exemplos ($r\hat{o}=0.11$; $p=0.4$); (ii) as notas obtidas nas respostas às perguntas de estímulo a auto-explicação e as notas do pós-teste ($r\hat{o}=-0.48$;

$p=0.4$); e (iii) o tempo de estudo dos exemplos e as notas do pós-teste ($r=0.33$; $p=0.24$).

6.2.3.3 Avaliação do material instrucional

Da mesma maneira que para os experimentos anteriores, o material instrucional deste experimento foi avaliado pelos participantes do experimento e por professores de diferentes áreas. Os resultados obtidos a partir destas avaliações são apresentados nas seções a seguir.

6.2.3.3.1 Avaliação dos estudantes

Na fase final deste experimento, coletamos o feedback dos estudantes através de um questionário final (adaptado de Renkl (2002)). Na primeira parte deste questionário, os estudantes dos GC, GI1 e GI2 foram perguntados sobre as suas opiniões a respeito da utilidade e qualidade dos exemplos trabalhados e das explicações apresentados pelo professor nos vídeos. Analisando os dados apresentados na Tabela 6.14, percebemos que os participantes de todos os grupos, GC, GI1 e GI2, tiveram uma opinião positiva a sobre a qualidade e utilidade dos exemplos e das explicações que foram fornecidas nos vídeos.

Tabela 6.14. Opiniões dos GC, GI1 e GI2 sobre os exemplos em vídeo.

		#de estudantes				
		DP ^a	D ^b	NN ^c	C ^d	CP ^e
Os exemplos e as explicações apresentados nos vídeos foram úteis para o meu aprendizado.	GC					5
	GI1					4
	GI2					3
Os exemplos e as explicações apresentados nos vídeos não me ajudaram no entendimento do assunto.	GC	4	1			
	GI1	3	1			
	GI2	3				
As explicações apresentadas nos vídeos foram detalhadas o suficiente para o entendimento do exemplo.	GC				1	4
	GI1					4
	GI2					3
As explicações apresentadas nos vídeos foram insuficientes para o entendimento do exemplo.	GC	4	1			
	GI1	3	1			
	GI2	3				

(a) Discordo plenamente (b) Discordo (c) Nem concordo, nem discordo (d) Concordo (e) Concordo plenamente

Fonte: própria autora.

Além disso, na segunda parte deste questionário, apresentada somente para os grupos instrucionais, os estudantes opinaram sobre a utilidade e qualidade das questões de auto-explicação presentes nos exemplos trabalhados em vídeo. Analisando os dados apresentados na Tabela 6.15, percebemos que a maioria dos participantes do GI1 e GI2

teve uma opinião positiva sobre a qualidade e utilidade das questões empregadas para promover a auto-explicação dos vídeos. As avaliações neutras também se fizeram presentes em algumas das respostas, porém elas foram em menor número.

Tabela 6.15. Opiniões dos GI1 e GI2 sobre as questões para AE nos exemplos em vídeo.

		#de estudantes				
		DP ^a	D ^b	NN ^c	C ^d	CP ^e
As questões propostas eram claras e objetivas.	GI1				1	3
	GI2				1	2
As questões propostas eram ambíguas e eu não consegui entendê-las por completo.	GI1	1	2	1		
	GI2	1	2			
As questões foram úteis durante o estudo do exemplo que foi apresentado no vídeo.	GI1				1	3
	GI2				2	1
As questões me ajudaram a entender o exemplo que foi apresentado no vídeo.	GI1			1	1	2
	GI2			1	1	1
As questões me atrapalharam no estudo do exemplo apresentado no vídeo.	GI1	2	1	1		
	GI2	1	2			

(a) Discordo plenamente (b) Discordo (c) Nem concordo, nem discordo (d) Concordo (e) Concordo plenamente

Fonte: própria autora.

6.2.3.3.2 Avaliação dos professores

Similarmente aos experimentos anteriores, além da avaliação com os estudantes, realizamos também uma avaliação de uma amostra do material instrucional com dois professores. Esta amostra consistiu de metade do material instrucional, ou seja, de dois dos quatro exemplos em vídeos juntamente com as perguntas para auto-explicação correspondentes utilizados nas aulas extras. O primeiro professor (P1) tem mestrado em Ciência da Computação com ênfase em Inteligência Artificial e experiência de 5 anos de ensino em disciplinas de Introdução à programação para o ensino médio. O segundo professor (P2) tem mestrado em Ciência da Computação com ênfase em Neurociência Computacional e experiência de 2 anos de ensino em disciplinas de Introdução à programação para o ensino médio. As respostas fornecidas pelos professores são apresentadas na Tabela 6.16. Nesta tabela, podemos perceber que os exemplos em vídeo e as perguntas para auto-explicação foram avaliadas positivamente por ambos os professores.

Tabela 6.16. Opiniões dos professores sobre o material instrucional.

	DP ^a	D ^b	NN ^c	C ^d	CP ^e
O exemplo é útil para o aprendizado de alunos iniciantes em programação, no contexto do objetivo de aprendizagem em questão.				P2	P1
O enunciado e a especificação do exemplo foram apresentados de maneira clara e objetiva.				P2	P1
A sequência de passos da solução do exemplo é adequada ao problema em questão.				P2	P1

As explicações apresentadas para os passos de solução são claras e objetivas.				P2	P1, P2
A utilização dos aspectos visuais de vídeo (i.e., ênfase, seleção, destaque e zoom) durante a apresentação da solução são adequadas aos momentos em que elas ocorrem.				P1, P2	
O som e a imagem do vídeo são de boa qualidade.				P2	P1
As questões incluídas no vídeo possuem um enunciado claro e objetivo.				P2	P1
As questões incluídas nos vídeos são úteis para o entendimento dos exemplos apresentados.				P2	P1
As questões são apresentadas em uma sequência adequada às diferentes seções do vídeo.				P2	P1
As questões são adequadamente alinhadas ao conteúdo apresentado.				P2	P1

(a) Discordo plenamente (b) Discordo (c) Nem concordo, nem discordo (d) Concordo (e) Concordo plenamente

Fonte: própria autora.

6.2.4 Discussão dos resultados

No início desta Seção, pressupomos que aprender programação através da auto-explicação de exemplos trabalhados em vídeo apresentaria resultados superiores quando comparados à condição de controle. Com base nos resultados obtidos da análise descritiva apresentada para os três quase-experimentos e no âmbito dos quase-experimentos realizados, podemos confirmar que aprender programação através da auto-explicação de exemplos trabalhados em vídeo trouxe resultados superiores quando comparados à condição controle dos quase-experimentos. Em relação à diferença de erros detectados pelo compilador do pré para o pós-teste, vimos que:

- no primeiro quase-experimento, GC teve o pior resultado do que GI1 e GI2;
- no segundo quase-experimento, os estudantes não cometeram erros deste tipo;
- por conta da ferramenta utilizada, não há erros deste tipo no terceiro quase-experimento.

Em relação à diferença de erros não detectados pelo compilador do pré para o pós-teste, vimos que:

- no primeiro quase-experimento, GC teve o pior resultado do que GI1 e GI2;
- no segundo quase-experimento, os estudantes em geral passaram a errar menos no pós do que erravam no pré-teste e a quantidade de erros foi diminuindo ao longo das aulas extras;
- no terceiro quase-experimento, GC teve o pior resultado, enquanto GI1 teve um resultado intermediário e GI2 apresentou o melhor resultado.

Em relação à quantidade de metas certas do pré para o pós-teste, podemos afirmar que:

- no primeiro quase-experimento, GC teve o menor taxa de aumento no número de metas certas, enquanto GI2 teve a segunda maior taxa e o GI1 a maior taxa de aumento neste número;
- no segundo quase-experimento, os estudantes tiveram uma melhoria na quantidade de metas certas;
- no terceiro quase-experimento, GC foi o grupo com o menor aumento no número de metas certas, enquanto GI1 foi o grupo com o maior aumento e GI2 o grupo com aumento intermediário.

Em relação à quantidade de metas erradas do pré para o pós-teste, podemos afirmar que:

- no primeiro quase-experimento, GC foi o único grupo que teve aumento para estas metas, enquanto GI1 e GI2 tiveram uma redução significativa;
- no segundo quase-experimento, os estudantes apresentaram redução para estas metas;
- no terceiro experimento, enquanto GC e GI2 aumentaram o número para estas metas, o GI1 reduziu esse número.

Em relação a quantidade de metas omitidas do pré para o pós-teste, podemos afirmar que:

- no primeiro quase-experimento, GC teve a menor redução, enquanto GI1 e GI2 tiveram uma redução maior para estas metas;
- no segundo quase-experimento, os estudantes apresentaram uma redução para estas metas;
- no terceiro quase-experimento, GI1 teve a maior redução, GC uma redução intermediária e GI2 a menor redução.

Por fim, em relação às notas do pré e do pós-teste, podemos afirmar que:

- no primeiro quase-experimento, em ordem crescente de resultados, temos GC, GI2 e GI1, onde GC obteve o menor aumento nas notas do pré para o pós-teste e GI1 o maior;

- no segundo quase-experimento, ambos os estudantes obtiveram aumento nas notas do pré para o pós-teste;
- no terceiro quase-experimento, em ordem crescente de resultados, temos GC, GI2 e GI1, onde GC obteve o pior resultado e GI1 o melhor.

Das diferenças entre os iniciantes e experientes (Seção 2.3), vimos que os experientes têm um melhor conhecimento sintático e semântico do que os iniciantes e, por isso, eles são mais precisos no que diz respeito à quantidade de erros cometidos. Vimos também que os iniciantes possuem o conhecimento frágil que é aquele conhecimento que os estudantes possuem, mas têm dificuldade em aplicar. Com base nos resultados apresentados, vimos que nos três quase experimentos os grupos controles tiveram um desempenho pior quando comparados aos grupos instrucionais para a maioria dos parâmetros avaliados. Os nossos resultados apontam que os estudantes dos grupos controles, após o estudo dos exemplos em vídeo, continuam cometendo mais erros sintáticos, semânticos e lógicos do que os estudantes dos grupos instrucionais, tanto na quantidade de erros individuais quanto na quantidade de metas erradas. Além disso, os grupos instrucionais apresentam um melhor resultado tanto na diminuição dos erros semânticos e lógicos, quanto na quantidade de metas certas, o que evidencia uma diferença entre estes grupos.

A partir da comparação dos resultados obtidos nos três quase-experimentos com as diferenças entre os iniciantes e os experientes relatadas pela literatura, não estamos querendo dizer que, após o estudo dos exemplos em vídeo, os estudantes dos grupos instrucionais viraram experientes em programação, enquanto os estudantes do GC continuaram como iniciantes. Contudo, o que os nossos resultados sugerem é que no caminho de iniciantes a experientes que estes estudantes estão percorrendo, os estudantes dos grupos instrucionais parecem estar mais distantes do ponto de partida, onde são puramente iniciantes, do que os dos grupos controles após o estudo dos exemplos em vídeo.

De maneira geral, os resultados obtidos não conseguiram mostrar uma correlação entre as respostas às questões de auto-explicação, o tempo de estudo dos exemplos em vídeos juntamente com as questões de auto-explicação e os resultados obtidos no pós-teste para nenhum dos grupos instrucionais. Estes resultados, no entanto, não podem ser julgados como definitivos, visto que, ao terminar de ver o exemplo em vídeo e responder as questões de AE, os estudantes poderiam rever o exemplo e manipular o

código resultante do exemplo até que o momento de estudo finalizasse. Os tempos de estudo dos exemplos em vídeo registrados são os tempos que os estudantes levaram para terminar de assistir o vídeo até que submetessem a resposta à última questão do vídeo e não contabiliza estes momentos adicionais de estudo. Durante estes momentos, os estudantes podem refinar o que aprenderam sobre os exemplos em vídeo, modificando ainda mais os seus esquemas a respeito dos exemplos que acabaram de estudar. Registramos em nossas observações que os estudantes que terminaram de assistir os vídeos assistiram partes específicas dele novamente depois que terminavam de assisti-lo pela primeira vez ou os estudantes ficaram manipulando o código resultante do exemplo em vídeo. Estes registros observacionais a respeito do momento de estudo nos trazem importantes indicativos que sugerem a realização de novas investigações.

Por fim, os participantes dos três quase-experimentos forneceram *feedback* positivo ao avaliar os exemplos em vídeo e as perguntas para AE utilizados neste experimento. De maneira geral, os estudantes concordaram que os exemplos e as explicações foram úteis e detalhadas o suficiente para o entendimento e aprendizado do conteúdo apresentado. A grande maioria também concordou que as questões para AE eram claras e objetivas, além de terem ajudado a estudar e compreender o conteúdo apresentado nos exemplos. Os nossos resultados também demonstram que os professores que participaram da avaliação do material instrucional dos quase-experimentos tiveram uma opinião positiva sobre o material. Em relação aos exemplos em vídeo, eles concordaram que os exemplos avaliados foram úteis para o aprendizado dos estudantes, que os problemas eram enunciados de maneira clara e objetiva, que a solução apresentada era adequada ao problema e que as explicações da solução também eram claras e objetivas. Além disso, recursos visuais do vídeo, som e imagem eram de boa qualidade. Em relação às questões, os professores concordaram que as questões incluídas no vídeo estavam clara e objetivamente enunciadas e que elas foram úteis para o entendimento dos exemplos apresentados. Eles também concordaram que as perguntas foram apresentadas em uma sequência adequada aos exemplos em vídeo e foram alinhadas apropriadamente ao conteúdo apresentado. Sendo assim, estes resultados somados aos já apresentados previamente nesta seção, são encorajadores e sugerem que prossigamos com a nossa pesquisa.

6.3 Implicações

Este sexto capítulo apresentou os resultados obtidos com os experimentos realizados nesta tese. Para avaliar a primeira versão da *Stepwise Self-explanation*, realizamos um experimento piloto. A partir dos dados coletados para este experimento, não conseguimos encontrar uma diferença estatisticamente significativa entre os participantes do grupo controle e do grupo instrucional. No entanto, obtivemos uma correlação significativa entre as notas das respostas às perguntas de auto-explicação e as notas do pós-teste. Assim, os estudantes do grupo instrucional que responderam mais corretamente as perguntas de auto-explicação durante a fase de aprendizado tiveram os melhores desempenhos na fase de programação. Além disso, os estudantes do grupo instrucional forneceram um *feedback* positivo sobre a nossa abordagem.

Para avaliar a segunda versão da *Stepwise Self-explanation*, realizamos três quase-experimentos com estudantes com diferentes níveis na iniciação à programação utilizando diferentes paradigmas e linguagens de programação. Os diferentes grupos nos quase-experimentos incluíram estudantes completamente iniciantes, iniciantes cursando introdução à programação pela primeira vez, iniciantes cursando introdução à programação pela segunda vez e iniciantes cursando introdução à programação pelo menos pela terceira vez. A partir dos dados coletados nestes quase-experimentos, encontramos uma diferença entre os estudantes dos grupos controles e dos grupos instrucionais. No pós-teste, os resultados apontam que os estudantes dos grupos controles apresentam os piores resultados, cometendo mais erros sintáticos, semânticos e lógicos do que os estudantes dos grupos instrucionais. Além disso, os estudantes dos grupos controles também implementam menos metas certas do que os estudantes dos grupos instrucionais. Com base nas quantidades médias de erros, de metas certas, erradas e omitidas, e das notas obtidas pelos estudantes, pudemos confirmar que aprender programação através da abordagem *Stepwise self-explanation* trouxe benefícios para os estudantes que dela fizeram uso, fazendo com que eles passassem a escrever programas que tinham menos erros sintáticos, semânticos e lógicos. Sendo assim, os nossos resultados sugerem que os estudantes que utilizaram a nossa abordagem estão mais distantes do ponto inicial da trajetória que eles devem percorrer de iniciantes a experientes em programação do que aqueles estudantes que não utilizaram a nossa abordagem.

De maneira geral, os nossos resultados não conseguiram mostrar uma correlação entre as respostas às questões de auto-explicação, o tempo de estudo dos exemplos em vídeos juntamente com as questões de auto-explicação e os resultados obtidos no pós-teste para nenhum dos grupos instrucionais. No entanto, em função dos bons resultados encontrados nas métricas apresentadas anteriormente, acreditamos que estes resultados não devem ser julgados como definitivos e novas investigações a respeito do momento de estudo dos estudantes precisam ser realizadas. Elas podem levar em consideração questões que ainda não sabemos responder com este trabalho, tais como o nível de engajamento e atenção dos estudantes com o material instrucional proporcionado pela abordagem ou o efeito das pausas proporcionadas pelas perguntas nos exemplos em vídeo para o aprendizagem de programação do estudante. Veremos mais adiante no sexto capítulo quais serão os trabalhos futuros decorrentes dos resultados obtidos neste capítulo.

7 Considerações finais

Este sétimo capítulo apresenta na Seção 7.1 as contribuições desta tese e na Seção 7.2 os principais trabalhos que podem dar continuidade à pesquisa nela apresentada.

7.1 Contribuições

A questão de pesquisa que norteou esta tese foi a seguinte **QP**: *como estimular e orientar o processo de aprendizagem dos estudantes iniciantes em programação enquanto eles estudam o processo de construção de programas a partir de exemplos trabalhados em vídeo?* Para tanto, propomos a definição de uma abordagem a qual nomeamos *Stepwise self-explanation* que visou contribuir com as áreas de ensino e aprendizagem de programação para iniciantes. Esta abordagem foi proposta para minimizar dois problemas destacados da literatura: (i) a dificuldade de combinar as diferentes estruturas de uma linguagem de programação para construir programas sintática, semântica e logicamente corretos; (ii) a maneira inadequada (ou insuficiente) que os estudantes estudam para uma disciplina introdutória de programação. Para fazer isso, lançamos mão da auto-explicação como atividade de aprendizagem para estimular e orientar os estudantes iniciantes enquanto eles aprendem o processo de programação descrito pelo FSI a partir de exemplos trabalhados em vídeo.

Propusemos duas versões para a abordagem *Stepwise self-explanation*, que se constituem na primeira contribuição desta tese. Na versão piloto da abordagem (Seção 4.2), foram utilizados exemplos trabalhados em vídeo da série *The Joy of Code* que contém as atividades de extensão e refinamento do processo de programação descrito pelo FSI. Os momentos de auto-explicação ocorreram logo após sequências de atividades de extensão e refinamento apresentadas nos exemplos. Para que os estudantes refletissem sobre as atividades de extensão, foram propostas questões que auxiliavam na reflexão sobre a identificação do objetivo ou da proposta do código sendo estendido. Já para que os estudantes refletissem sobre as atividades de refinamento, foram propostas questões que auxiliam na reflexão sobre a aplicação dos conceitos de uma linguagem de programação para alcançar o objetivo identificado.

Com o objetivo de avaliar a eficácia da primeira versão da abordagem, realizamos um experimento piloto com estudantes do ensino médio e técnico. Este experimento piloto constitui-se na segunda contribuição desta tese. Embora com este experimento não tenhamos conseguido encontrar uma diferença estatisticamente significativa entre

os grupos instrucional e controle, verificamos a necessidade de aprimorar o projeto do nosso estudo em dois pontos: (i) em relação ao nível de dificuldade dos tópicos de estudo escolhidos para serem estudados; e (ii) em relação ao número de atividades realizadas por sessão de estudo do experimento. Além disso, verificamos a necessidade de refinamento da própria abordagem, com relação às questões propostas e aos momentos em que elas são oferecidas aos estudantes para que reflitam sobre as atividades de programação desenvolvidas.

Na versão refinada da Stepwise Self-explanation (Seção 4.3), os exemplos em vídeo foram todos produzidos pela autora desta tese de acordo com o processo de programação definido pelo FSI e constituem-se na terceira contribuição desta tese. Similarmente aos exemplos utilizados na versão anterior da abordagem, os exemplos utilizados nesta segunda versão contém apenas as atividades de extensão e refinamento. Além disso, os momentos de auto-explicação ocorreram em quatro momentos diferentes dos exemplos em vídeo. Logo após uma extensão principal, para que os estudantes refletissem sobre a o objetivo geral do programa sendo estendido. Logo após uma extensão intermediária, para que os estudantes refletissem sobre uma possível implementação de um trecho de código que atenda à meta intermediária que será alcançada com o extensão em questão. Logo após o refinamento, para que os estudantes refletissem sobre a aplicação dos conceitos de uma linguagem de programação para alcançar o objetivo do passo em questão e o funcionamento do trecho de código que foi implementado. Por fim, logo após a finalização da implementação do problema, para que os estudantes refletissem sobre o funcionamento global do programa depois que toda a implementação para ele foi fornecida.

Com o objetivo de avaliar a eficácia da versão refinada da abordagem, realizamos três quase-experimentos com estudantes do ensino médio, técnico e superior. Estes experimentos e o material instrucional produzido para eles constituem-se na quarta e quinta contribuições desta tese. O primeiro quase-experimento foi realizado no formato de aulas extras, ministradas para estudantes do ensino superior matriculados pelo menos pela segunda vez em uma disciplina introdutória de programação em C. Estes estudantes foram divididos em três grupos, dois grupos instrucionais, para o primeiro as questões de auto-explicação eram abertas (GI1) e para o segundo as questões de auto-explicação eram de múltipla escolha (GI2), e um grupo controle (GC). Ao analisar as médias de erros, de metas certas, erradas e omitidas e das notas obtidas pelos diferentes

grupos, vimos que o GC foi o grupo que apresentou os piores resultados, enquanto o GI1 e o GI2, especialmente o GI1, apresentaram os melhores resultados quando comparados aos apresentados pelo GC.

De maneira similar, o segundo quase-experimento foi realizado no formato de aulas extras e contou com a participação de estudantes do ensino superior matriculados pelo menos pela terceira vez em uma disciplina introdutória de programação em C. Estes estudantes estudaram os exemplos em vídeo que continham questão de auto-explicação que eram abertas. Ao analisar as diferenças entre as quantidades médias de erros, de metas certas, erradas e omitidas e das notas obtidas pelos estudantes do pré para o pós-teste, vimos que os participantes deste experimento tiveram ganhos de aprendizagem significativos após o momento de estudo dos exemplos a partir das questões de auto-explicação.

Por fim, o terceiro quase-experimento foi realizado como um minicurso para estudantes do ensino médio e técnico. Os estudantes que cursavam apenas o ensino técnico estavam matriculados pela primeira vez em uma disciplina introdutória de programação em C e relataram estar sentindo muitas dificuldades no aprendizado da disciplina. Já os estudantes que cursavam o ensino médio juntamente com o técnico não tinham nenhum tipo de experiência com a disciplina de programação. Estes estudantes foram divididos em dois grupos, um grupo instrucional (GI), com questões de auto-explicação de múltipla escolha, e um grupo controle (GC). Posteriormente, para a análise, o GI foi sub-dividido em dois grupos, o GI1 com estudantes matriculados na disciplina introdutória de programação e o GI2 com estudantes que não estavam matriculados nesta disciplina. Ao analisar as médias de erros, de metas certas, erradas e omitidas e das notas obtidas pelos diferentes grupos, vimos que o GC foi o grupo que teve os piores resultados, enquanto o GI1 e o GI2, especialmente o GI1, apresentaram resultados melhores quando comparados aos apresentados pelo GC.

Na fase final de todos os experimentos, coletamos o *feedback* dos participantes através de um questionário final. A maioria deles avaliou positivamente os exemplos em vídeo juntamente com as questões propostas para a auto-explicação. Além disso, uma amostra dos exemplos em vídeo e das questões de auto-explicação utilizada em cada um dos experimentos também foi avaliada por dois professores com experiência na área de programação e na área de educação. De maneira similar, a maioria dos professores teve

uma opinião positiva a respeito do material avaliado por eles. Esta avaliação do material por estudantes e por professores se constitui na sexta contribuição desta tese.

Além disso, antes de definir esta abordagem, realizamos uma revisão da literatura onde identificamos que aprender e ensinar a programar são duas atividades consideradas difíceis e que os estudantes iniciantes em programação não vão se tornar bons programadores em poucas horas. Realizamos também um mapeamento sistemático da literatura de 13 anos de publicações realizadas pela comunidade científica brasileira sobre os processos de ensino e aprendizagem de programação para iniciantes. Este mapeamento revelou que o interesse dos pesquisadores sobre o assunto vem crescendo ao longo dos anos, uma vez que as dificuldades encontradas no Brasil são as mesmas encontradas pela comunidade como um todo. Além disso, vimos que as publicações apresentam, em sua maioria, ferramentas e, em segundo lugar e em um número muito menor, técnicas para o ensino-aprendizagem de programação. Estes resultados estão em desacordo com os encontrados pela revisão apresentada por Sheard et al. (2009), que apresentam um grande número de técnicas seguido por um menor número de ferramentas para o ensino-aprendizagem de programação. Este mapeamento se constitui na sétima e última contribuição desta tese.

7.2 Trabalhos futuros

Durante a realização deste trabalho, várias questões foram levantadas. Algumas destas questões serão apresentadas nesta seção para que possam guiar o futuro da pesquisa aqui iniciada.

7.2.1 Refinamento dos procedimentos metodológicos para avaliação da *Stepwise Self-explanation*

Pretendemos refinar os procedimentos metodológicos em trabalhos futuros a presente tese. Por exemplo, a abordagem *Stepwise Self-explanation* foi utilizada em experimentos realizados como aula extra por 2 ou 4 encontros semanais ou em um minicurso de 4 horas com resultados promissores. No entanto, pretendemos incluir a abordagem como parte de uma disciplina introdutória de programação ministrada durante todo um semestre (ou ano) letivo para que possamos avaliar também a sua efetividade por um período mais longo do que os que foram realizados nesta tese. Pretendemos também incluir a técnica de grupos focais [GONDIM, 2002] para coleta de

informações mais detalhadas sobre a experiência dos estudantes sobre a utilização da abordagem e os seus momentos de estudo a partir dela. Melhorias como estas mostram-se extremamente importantes, pois visam fornecer dados mais aprofundados, necessários à evolução da abordagem.

7.2.2 Introdução de um outro momento na abordagem para discussão em pares e/ou grupos

A abordagem *Stepwise self-explanation* como está definida agora prevê que os estudantes estudem e reflitam sobre o processo de programação sozinhos. Acreditamos que este processo de estudo pode ser aprimorado com a introdução de um momento adicional de estudo e reflexão, onde os estudantes podem discutir o que aprenderam com seus pares, seja em dupla e/ou em grupos. De acordo com Chi e Wylie (2014), quando o estudante estuda através da auto-explicação sozinho e depois passa a estudar com outro(s) estudante(s), o aprendizado deste estudante aumenta. Sendo assim, este momento de estudo interativo, seja em pares ou em grupos, após o estudo individual se mostra extremamente importante, pois pode enriquecer ainda mais o processo de estudo dos iniciantes em programação.

7.2.3 Auto-explicação de exemplos trabalhados alternativos

À medida que os estudantes vão avançando na disciplina introdutória de programação é importante que eles aprendam que um mesmo problema pode ser solucionado de diferentes maneiras. Portanto, pode se pensar na gravação de exemplos trabalhados alternativos para um mesmo problema. Estes exemplos alternativos poderiam ser estudados em conjunto e, após o estudo deles, poderiam ser propostas perguntas adicionais que estimulassem a auto-explicação dos estudantes com foco na comparação entre as diferentes alternativas estudadas. Sendo assim, poderiam ser feitas perguntas que solicitassem uma análise das vantagens e desvantagens da utilização de um determinado construto de uma linguagem de programação, por exemplo. Esta gravação de soluções alternativas poderia ser realizada por um professor da disciplina individualmente ou em colaboração com outros professores de uma mesma ou de várias instituições.

7.2.4 Auto-explicação de exemplos trabalhados com erros

À medida que os estudantes vão avançando na disciplina introdutória de programação também é importante que eles aprendam a depurar e resolver erros nos exemplos trabalhados. Sendo assim, pode se pensar na gravação de exemplos trabalhados com erros para os quais os estudantes precisam propor soluções. Ao propor as soluções, eles devem também auto-explicar os erros encontrados e como as implementações das soluções vão resolver os erros encontrados.

7.2.5 Envolvimento dos estudantes no processo de gravação dos exemplos trabalhados em vídeo

Além de fazer com que os estudantes auto-expliquem os exemplos em vídeo através da utilização da abordagem *Stepwise self-explanation*, podemos envolver os estudantes no processo de gravação de exemplos em vídeos, fazendo com que eles gravem e compartilhem soluções para problemas similares aos estudados. Gravar as soluções em vídeo estimula os estudantes a explicar as suas próprias soluções e pode contribuir para o aprendizado de programação. Aliado a isso, os estudantes podem analisar diferentes exemplos trabalhados alternativos, comparando as vantagens e as desvantagens de cada um deles. Lui, Cheung e Li (2008) estimularam seus estudantes a compartilharem os exemplos trabalhados estáticos que eles construíram em sua disciplina de programação e esta atividade se mostrou benéfica para aqueles estudantes que compartilharam suas soluções.

7.2.6 Sistema tutor inteligente para propor perguntas de acordo com o nível de conhecimento e do contexto da atividade de aprendizado do estudante

Atualmente a abordagem provê que o mesmo conjunto de questões para auto-explicação seja utilizado por todos os estudantes. Um dos mecanismos do auxílio fornecido pelo *scaffolding* é que este auxílio vá sendo gradualmente diminuído à medida que o estudante for ganhando mais experiência. Sendo assim, um dos desdobramentos possíveis desta tese é a implementação de um sistema tutor inteligente que avalie o conhecimento do estudante continuamente, fornecendo *feedback* para as auto-explicações fornecidas e adaptando as questões para auto-explicação de acordo com o seu nível de conhecimento. Por exemplo, se o estudante tiver um completo entendimento sobre como declarar variáveis, não é necessário fazer perguntas relacionadas a esta atividade. Além disso, durante a reprodução do exemplo em vídeo

pelo estudante, caso ele esteja apresentando dificuldades, o tutor pode fornecer dicas de como ele vai resolver o problema. Por fim, o tutor pode também sugerir novas atividades relacionadas, tais como o estudo de novos exemplos trabalhados equivalentes, mais simples ou mais complexos do que o que ele acabou de assistir, por exemplo.

7.2.7 Relatório diagnóstico dos estudantes como forma de auxílio ao processo de avaliação pelo professor

O estudo dos exemplos trabalhados através da auto-explicação por escrito gera um conjunto de informações sobre o estudante que pode ser utilizado para auxiliar o professor em sua avaliação dos estudantes. Para tanto, os professores podem ter um acompanhamento mais regular do desempenho dos estudantes através de diferentes tipos de relatórios, tais como o tempo de estudo dos exemplos em vídeo, a quantidade e quais as respostas que eles mais acertaram e mais erraram, no caso de ele ser associado a uma ferramenta de avaliação automática, quantidade e quais os erros gerados ao reproduzir o código dos exemplos, entre outros. Estes relatórios podem guiar não somente a prática pedagógica do professor em sala de aula, mas também a seleção e produção de novos exemplos trabalhados em vídeo que devem ser estudados pelos estudantes utilizando a *Stepwise self-explanation*.

7.2.8 Material instrucional para utilização da abordagem por outros professores de programação

Até o presente momento, a abordagem *Stepwise self-explanation* foi utilizada apenas pela autora desta tese. Para que ela seja utilizada por outros professores de programação, é importante que seja criado um material instrucional que os auxilie no processo de aprendizagem de construção de exemplos trabalhados em vídeo de acordo com o FSI e também na proposição de questões correspondentes para auxílio aos estudantes em seus processos de auto-explicação destes exemplos em vídeo.

7.2.9 Generalizar a abordagem para utilização em outras disciplinas

A *Stepwise self-explanation* foi pensada para disciplinas introdutórias de programação, mas a abordagem pode ser generalizada com poucas adaptações para que possa ser utilizada em outras disciplinas. Por este motivo, pretendemos generalizar a abordagem para que ela possa ser utilizada em outras disciplinas da área de computação.

Além disso, para avaliar a abordagem em outras disciplinas também será necessário a realização de novos experimentos.

Referências

- ABDUL-RAHMAN, S.-S.; DU BOULAY, B. Learning programming via worked-examples: Relation of learning styles to cognitive load. **Computers in Human Behavior**, v. 30, p. 286-298, 1// 2014.
- ADAMS, J. C. Alice, middle schoolers & the imaginary worlds camps. **ACM SIGCSE Bulletin**, v. 39, n. 1, p. 307-311, 2007.
- ALEVEN, V. A. W. M. M.; KOEDINGER, K. R. An effective metacognitive strategy: learning by doing and explaining with a computer-based Cognitive Tutor. **Cognitive Science**, v. 26, n. 2, p. 147-179, 2002.
- ASCENCIO, A. F. G.; DE CAMPOS, E. A. V. **Fundamentos da programação de computadores: algoritmos, Pascal, C/C++ e Java**. LONGMAN DO BRASIL, 2008. ISBN 9788576051480.
- ATKINSON, R. K.; RENKL, A.; MERRILL, M. M. Transitioning From Studying Examples to Solving Problems: Effects of Self-Explanation Prompts and Fading Worked-Out Steps. **Journal of Educational Psychology**, v. 95, n. 4, p. 774-783, 2003.
- ATKINSON, R. K. et al. Learning from Examples: Instructional Principles from the Worked Examples Research. **Review of Educational Research**, v. 70, n. 2, p. 181-214, 2000.
- AURELIANO, V. C. O.; TEDESCO, P. C. D. A. R. Avaliando o uso do Scratch como abordagem alternativa para o processo de ensino-aprendizagem de programação. CONGRESSO DA SOCIEDADE BRASILEIRA DE COMPUTAÇÃO, 2012a. p.1-10.
- _____. Ensino-aprendizagem de Programação para Iniciantes: uma Revisão Sistemática da Literatura focada no SBIE e WIE. Anais do Simpósio Brasileiro de Informática na Educação, 2012b.
- _____. **Utilizando o Scratch para Apoiar o Processo de Ensino-aprendizagem de Programação para Iniciantes na EaD**. XXXIII CONGRESSO DA SOCIEDADE BRASILEIRA DE COMPUTAÇÃO 2013.
- AURELIANO, V. C. O.; TEDESCO, P. C. D. A. R. **Aprendendo linguagem de programação através da auto-explicação de exemplos em vídeo**. Proceedings of the IV Congresso Brasileiro de Informática na Educação. Maceió - Alagoas - Brasil 2015.
- AURELIANO, V. C. O.; TEDESCO, P. C. D. A. R.; GIRAFFA, L. M. M. **Desafios e oportunidades aos processos de ensino e de aprendizagem de programação para iniciantes**. 24º WEI - Workshop sobre Educação em Computação. Porto Alegre. Anais do CSBC 2016 2016.

BENASSI, V. et al. **Applying Science of Learning in Education: Infusing Psychological Science Into the Curriculum**. 2014. ISBN 9781941804292.

BENNEDSEN, J.; CASPERSEN, M. E. Failure rates in introductory programming. **SIGCSE Bull.**, v. 39, n. 2, p. 32-36, 2007.

_____. Exposing the Programming Process. In: BENNEDSEN, J.; CASPERSEN, M., *et al* (Ed.). **Reflections on the Teaching of Programming**: Springer Berlin Heidelberg, v.4821, 2008. cap. 2, p.6-16. (Lecture Notes in Computer Science). ISBN 978-3-540-77933-9.

BERGIN, S.; REILLY, R. **Programming: factors that influence success**. Proceedings of the 36th SIGCSE technical symposium on Computer science education. St. Louis, Missouri, USA: ACM: 411-415 p. 2005.

BERTHOLD, K.; RENKL, A. Fostering the understanding of multi-representational examples by self-explanation prompts. In: BARA, B. G.; BARSALOU, L., *et al*, Proceedings of the CogSci 2005, 2005a. Erlbaum. p.250-255.

_____. **Fostering the understanding of multi-representational examples by self-explanation prompts**. Erlbaum. 2005b

BERTHOLD, K.; RENKL, A. How to Foster Active Processing of Explanations in Instructional Communication. **Educational Psychology Review**, v. 22, n. 1, p. 25-40, 2010/03/01 2010.

BIELACZYK, K.; PIROLI, P. L.; BROWN, A. L. Training in Self-Explanation and Self-Regulation Strategies: Investigating the Effects of Knowledge Acquisition Activities on Problem Solving. **Cognition and Instruction**, v. 13, n. 2, p. 221-252, 1995/06/01 1995.

BIGGS, J.; TANG, C. **Teaching For Quality Learning At University**. McGraw-Hill Education, 2011. ISBN 9780335242757.

BÖRSTLER, J.; CASPERSEN, M.; NORDSTRÖM, M. Beauty and the Beast: on the readability of object-oriented example programs. **Software Quality Journal**, p. 1-16, 2015/02/14 2015.

BRANSFORD, J. D.; BROWN, A. L.; COCKING, R. R. **How People Learn: Brain, Mind, Experience and School**. Washington, DC: {N.R. Council} National Academy Press, 2000.

BRENNAN, K.; CHUNG, M.; HAWSON, J. Creative computing: A design-based introduction to computational thinking. **Retrieved May**, v. 9, p. 2012, 2011.

BROWN, J. S.; COLLINS, A.; NEWMAN, S. Cognitive apprenticeship: Teaching the crafts of reading, writing, and mathematics. **Knowing, learning, and instruction: Essays in honor of Robert Glaser**, p. 487, 1989.

CARLISLE, M. C. **Using You Tube to enhance student class preparation in an introductory Java course**. Proceedings of the 41st ACM technical symposium on Computer science education. Milwaukee, Wisconsin, USA: ACM: 470-474 p. 2010.

CASPERSEN, M. E. **Educating Novices in the Skills of Programming**. 2007. 323 DAIMI Aarhus University

CASPERSEN, M. E.; KÖLLING, M. STREAM: A First Programming Process. **Trans. Comput. Educ.**, v. 9, n. 1, p. 1-29, 2009.

CHASE, W. G.; SIMON, H. A. Perception in chess. **Cognitive Psychology**, v. 4, n. 1, p. 55-81, 1// 1973.

CHERENKOVA, Y.; ZINGARO, D.; PETERSEN, A. **Identifying challenging CS1 concepts in a large problem dataset**. Proceedings of the 45th ACM technical symposium on Computer science education. Atlanta, Georgia, USA: ACM: 695-700 p. 2014.

CHI, M. T. Self-explaining expository texts: The dual processes of generating inferences and repairing mental models. **Advances in instructional psychology**, v. 5, p. 161-238, 2000.

CHI, M. T. H. et al. Self-explanations: How students study and use examples in learning to solve problems. **Cognitive Science**, v. 13, p. 145-182+, 1989.

CHI, M. T. H. et al. Eliciting Self-Explanations Improves Understanding. **Cognitive Science**, v. 18, n. 3, p. 439-477, 1994.

CHIU, J. L.; CHI, M. T. **Supporting self-explanation in the classroom**. Applying science of learning in education: Infusing psychological science into the curriculum. BENASSI, V. A.; OVERSON, C. E., et al: 91-103 p. 2014.

CLARK, R. C.; NGUYEN, F.; SWELLER, J. **Efficiency in Learning: Evidence-Based Guidelines to Manage Cognitive Load**. Wiley, 2005. ISBN 978-0-7879-7728-3.

COLLINS, A.; BROWN, J. S.; HOLUM, A. Cognitive apprenticeship: Making thinking visible. **American educator**, v. 15, n. 3, p. 6-11, 1991.

COMPUTAÇÃO, S. B. D. **Currículo de Referência para Cursos de Graduação em Bacharelado em Ciência da Computação e Engenharia de Computação** 2005.

COOPER, S.; DANN, W.; PAUSCH, R. Alice: a 3-D tool for introductory programming concepts. **J. Comput. Sci. Coll.**, v. 15, n. 5, p. 107-116, 2000.

CRESWELL, J. W. **Research Design: Qualitative, Quantitative, and Mixed Methods Approaches**. SAGE Publications, 2013. ISBN 9781452226101.

CRESWELL, J. W.; GARRETT, A. L. The "movement" of mixed methods research and the role of educators. **South African Journal of Education**, v. 28, p. 321-333, 2008.

CRIPPEN, K. J.; EARL, B. L. The impact of web-based worked examples and self-explanation on performance, problem solving, and self-efficacy. **Comput. Educ.**, v. 49, n. 3, p. 809-821, 2007.

DANCEY, C. P.; REIDY, J. **Estatística sem Matemática para Psicologia - 5ed.** Penso Editora, 2013. ISBN 9788565848114.

DAVIES, S. P. Models and theories of programming strategy. **International Journal of Man-Machine Studies**, v. 39, n. 2, p. 237-267, 8// 1993.

DE ARAUJO, E. C. et al. O papel do hábito de estudo no desempenho do aluno de programação. 2013.

DE JESUS, E. A.; RAABE, A. L. Avaliação Empírica da Utilização de um Jogo para Auxiliar a Aprendizagem de Programação. Anais do Simpósio Brasileiro de Informática na Educação, 2010.

DE RAADT, M. et al. Approaches to learning in computer programming students and their effect on success. Proceedings of the 28th HERDSA Annual Conference: Higher Education in a Changing World (HERDSA 2005), 2005. Higher Education Research and Development Society of Australasia (HERDSA). p.407-414.

DOTTA, S. C. et al. Análise das Preferências dos Estudantes no uso de Videoaulas: Uma experiência na Educação a Distância. **Anais do Workshop de Informática na Escola**, v. 1, n. 1, 2013.

DU BOULAY, B. Some difficulties of learning to program. In: (Ed.). **Studying the Novice Programmer**, 1989. p.431-446. ISBN 1317786203.

EDUCAÇÃO, M. D. **Catálogo Nacional dos Cursos Superiores de Tecnologia** 2010.

FERREIRA, C.; GONZAGA, F.; SANTOS, R. Um Estudo sobre a Aprendizagem de Lógica de Programação Utilizando Programação por Demonstração. **Faculdade Governador Ozanam Coelho/Universidade Federal de Alfenas/Universidade Federal do Rio de Janeiro**. Disponível em:< http://www.inf.pucminas.br/sbc2010/anais/pdf/wei/st_06_03.pdf>. Acesso em, v. 23, 2009.

FOWLER, M.; BECK, K. **Refactoring: Improving the Design of Existing Code**. Addison-Wesley, 1999. ISBN 9780201485677.

GOMES, A.; MENDES, A. Learning to program-difficulties and solutions. **International Conference on Engineering Education–ICEE**, v. 2007, 2007.

GOMES, A. J.; SANTOS, A. N.; MENDES, A. J. A study on students' behaviours and attitudes towards learning to program. Proceedings of the 17th ACM annual conference on Innovation and technology in computer science education, 2012. ACM. p.132-137.

GONDIM, H. W. A.; AMBRÓSIO, A. P.; COSTA, F. M. Uma Experiência no Ensino de Algoritmos utilizando Ambientes Visuais de Programação 3D. Workshop sobre Educação em Computação (WEI), Bento Gonçalves-RS, 2009.

GONDIM, S. M. G. Grupos focais como técnica de investigação qualitativa: desafios metodológicos. **Paidéia (Ribeirão Preto)**, v. 12, p. 149-161, 2002.

GRAY, S. et al. **Suggestions for graduated exposure to programming concepts using fading worked examples**. Proceedings of the third international workshop on Computing education research. Atlanta, Georgia, USA: ACM: 99-110 p. 2007.

GROSS, S. et al. Towards Deeper Understanding of Syntactic Concepts in Programming. The First Workshop on AI-supported Education for Computer Science (AIEDCS 2013), 2013. p.11.

GUEDES, E. B. Um Estudo Observacional sobre a Disciplina Introdutória de Programação. Anais do Workshop de Informática na Escola, 2014.

HENRIKSEN, P.; KÖLLING, M. Greenfoot: combining object visualisation with interaction. Companion to the 19th annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications, 2004. ACM. p.73-82.

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

J. T. F. ON COMPUTING CURRICULA, A., IEEE CS. **Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science**. ACM, 2013. 518 ISBN 978-1-4503-2309-3.

JAMES, S. et al. Use and perceptions of worked example videos for first-year students studying mathematics in a primary education degree. Lighthouse Delta 2013: The 9th Delta Conference on teaching and learning of undergraduate mathematics and statistics, 2013. p.59-69.

JOHNSON, C. I.; MAYER, R. E. Applying the self-explanation principle to multimedia learning in a computer-based game-like environment. **Computers in Human Behavior**, v. 26, n. 6, p. 1246-1252, 2010.

KALYUGA, S. et al. The Expertise Reversal Effect. **Educational Psychologist**, v. 38, n. 1, p. 23-31, 2003/03/01 2003.

KALYUGA, S. et al. When problem solving is superior to studying worked examples. **Journal of Educational Psychology**, v. 93, n. 3, p. 579-588, 2001.

KINNUNEN, P.; MALMI, L. Why students drop out CS1 course? , Proceedings of the second international workshop on Computing education research, 2006. ACM. p.97-108.

KIRSCHNER, P. A.; SWELLER, J.; CLARK, R. E. Why Minimal Guidance During Instruction Does Not Work: An Analysis of the Failure of Constructivist, Discovery, Problem-Based, Experiential, and Inquiry-Based Teaching. **Educational Psychologist**, v. 41, n. 2, p. 75-86, 2006/06/01 2006.

KÖLLING, M. **Greenfoot: a highly graphical ide for learning object-oriented programming**. Proceedings of the 13th annual conference on Innovation and technology in computer science education. Madrid, Spain: ACM: 327-327 p. 2008.

KÖLLING, M. **Introduction to Programming with Greenfoot: Object-Oriented Programming in Java with Games and Simulations**. Prentice Hall, 2009. ISBN 9780136037538.

_____. **Introduction to Programming with Greenfoot: Object-oriented Programming in Java with Games and Simulations**. Prentice Hall, 2010. ISBN 9780136037538.

KÖLLING, M. et al. The BlueJ System and its Pedagogy. **Computer Science Education**, v. 13, n. 4, p. 249-268, 2003/12/01 2003.

KUMAR, A. N. An Evaluation of Self-explanation in a Programming Tutor. In: TRAUSAN-MATU, S.;BOYER, K., *et al* (Ed.). **Intelligent Tutoring Systems**: Springer International Publishing, v.8474, 2014. cap. 30, p.248-253. (Lecture Notes in Computer Science). ISBN 978-3-319-07220-3.

KURLAND, D. M. et al. A study of the development of programming ability and thinking skills in high school students. **Journal of Educational Computing Research**, v. 2, n. 4, p. 429-458, 1986.

LAHTINEN, E. et al. **A study of the difficulties of novice programmers**. Proceedings of the 10th annual SIGCSE conference on Innovation and technology in computer science education. Caparica, Portugal: ACM: 14-18 p. 2005.

LOPES, A. P. B.; DE MENEZES, C. S. Avaliando o EpA—um estudo de caso na aprendizagem de computação. Anais do Workshop de Informática na Escola, 2006.

MALONEY, J. et al. The Scratch Programming Language and Environment. **Trans. Comput. Educ.**, v. 10, n. 4, p. 1-15, 2010.

MALONEY, J. H. et al. Programming by choice: urban youth learning programming with scratch. **SIGCSE Bull.**, v. 40, n. 1, p. 367-371, 2008.

MARGULIEUX, L. E.; GUZDIAL, M.; CATRAMBONE, R. **Subgoal-labeled instructional material improves performance and transfer in learning to develop mobile applications**. Proceedings of the ninth annual international conference on International computing education research. Auckland, New Zealand: ACM: 71-78 p. 2012.

MCCAULEY, R. et al. Debugging: a review of the literature from an educational perspective. **Computer Science Education**, v. 18, n. 2, p. 67-92, 2008.

MCCRACKEN, M. et al. **A multi-national, multi-institutional study of assessment of programming skills of first-year CS students**. Working group reports from ITiCSE on Innovation and technology in computer science education. Canterbury, UK: ACM: 125-180 p. 2001.

MCNAMARA, D.; LEVINSTEIN, I.; BOONTHUM, C. iSTART: Interactive strategy training for active reading and thinking. **Behavior Research Methods, Instruments, & Computers**, v. 36, n. 2, p. 222-233, 2004/05/01 2004.

MCNAMARA, D. S. SERT: Self-Explanation Reading Training. **Discourse Processes**, v. 38, n. 1, p. 1-30, 2004/07/01 2004.

MILLER, G. A. The magical number seven, plus or minus two. **Psychological Review**, v. 63, n. 2, p. 81-97, 1956.

MILNE, I.; ROWE, G. Difficulties in learning and teaching programming—views of students and tutors. **Education and Information technologies**, v. 7, n. 1, p. 55-66, 2002.

MIZRAHI, V. V. **Treinamento em linguagem C**. PRENTICE HALL BRASIL, 2008. ISBN 9788576051916.

MORRISON, B. B.; MARGULIEUX, L. E.; GUZDIAL, M. **Subgoals, Context, and Worked Examples in Learning Computing Problem Solving**. Proceedings of the eleventh annual International Conference on International Computing Education Research. Omaha, Nebraska, USA: ACM: 21-29 p. 2015.

MOSKAL, B.; LURIE, D.; COOPER, S. **Evaluating the effectiveness of a new instructional approach**. Proceedings of the 35th SIGCSE technical symposium on Computer science education. Norfolk, Virginia, USA: ACM: 75-79 p. 2004.

OMG. **Business Process Model and Notation (BPMN), Version 2.0** 2011.

PAAS, F. G. Training strategies for attaining transfer of problem-solving skill in statistics: A cognitive-load approach. **Journal of Educational Psychology**, v. 84, n. 4, p. 429-434, 1992.

PASHLER, H. et al. **Organizing instruction and study to improve student learning [electronic resource]**. Washington, D.C: U.S. Dept. of Education, National Center for Education Research, Institute of Education Sciences, 2007.

PEARS, A. et al. A survey of literature on the teaching of introductory programming. **SIGCSE Bull.**, v. 39, n. 4, p. 204-223, 2007.

PERKINS, D. N.; MARTIN, F. **Fragile knowledge and neglected strategies in novice programmers**. Papers presented at the first workshop on empirical studies of programmers on Empirical studies of programmers. Washington, D.C., USA: Ablex Publishing Corp.: 213-229 p. 1986.

PIROLI, P.; RECKER, M. Learning Strategies and Transfer in the Domain of Programming. **Cognition and Instruction**, v. 12, n. 3, p. 235-275, 1994/09/01 1994.

POLYA, G. **How to Solve It: A New Aspect of Mathematical Method: A New Aspect of Mathematical Method**. Princeton university press, 1945. ISBN 1400828678.

PUTNAM, R. T. et al. A summary of misconceptions of high school Basic programmers. In: (Ed.). **Studying the novice programmer**, 1989. p.301-314. ISBN 1317786203.

RAABE, A.; BERNARDES, A.; JUNIOR, R. G. A. **Produção e Avaliação de Videoaulas: Um Estudo de Caso no Ensino de Programação**. Anais do Workshop de Informática na Escola, 2014.

RAABE, A. L. A.; GIRAFFA, L. M. M. Uma arquitetura de tutor para promover experiências de aprendizagem mediadas. **Simpósio Brasileiro de Informática na Educação, SBIE**, 2006.

RENKL, A. Learning from Worked-Out Examples: A Study on Individual Differences. **Cognitive Science**, v. 21, n. 1, p. 1-29, 1997.

RENKL, A. Worked-out examples: instructional explanations support learning by self-explanations. **Learning and Instruction**, v. 12, n. 5, p. 529-556, 10// 2002.

RISSOLI, V. R. V.; SANTOS, G. A. O Agente Pedagógico Animado MInA. Anais do Simpósio Brasileiro de Informática na Educação, 2013.

RITTLE-JOHNSON, B. Promoting transfer: Effects of self-explanation and direct instruction. **Child development**, v. 77, n. 1, p. 1-15, 2006.

ROBINS, A. V.; ROUNTREE, J.; ROUNTREE, N. Learning and Teaching Programming: A Review and Discussion. **Computer Science Education**, v. 13, n. 2, p. 137-172, / 2003.

ROGALSKI, J.; SAMURÇAY, R. Acquisition of programming knowledge and skills. **Psychology of programming**, v. 18, p. 157-174, 1990.

SAMURÇAY, R. The concept of variable in programming: Its meaning and use in problem-solving by novice programmers. **Studying the novice programmer**, v. 9, p. 161-178, 1989.

SCHOENFELD, A. H. **Problem solving in the mathematics curriculum: a report recommendations and annotated bibliography**. MAA, 1983.

_____. **Mathematical problem solving**. Academic press Orlando, FL, 1985. ISBN 0126288704.

SHEARD, J. et al. **Analysis of research into the teaching and learning of programming**. Proceedings of the fifth international workshop on Computing education research workshop. Berkeley, CA, USA: ACM: 93-104 p. 2009.

SILVA FILHO, R. L. L. E. et al. A evasão no ensino superior brasileiro. **Cadernos de Pesquisa**, v. 37, p. 641-659, 2007.

SILVA, M. T.; CAVALCANTE, M. C. T. C.; DE BARROS COSTA, E. Explorando correlações em Programação: um estudo focado no processo seletivo e em disciplinas correlatas. Anais do Simpósio Brasileiro de Informática na Educação, 2013.

SILVA, M. T. et al. Um Arcabouço para Construção de Mecanismos de Análise de Códigos de Programação Introdutória. Anais do Simpósio Brasileiro de Informática na Educação, 2014. p.1083-1092.

SIMON. A Classification of Recent Australasian Computing Education Publications. **Computer Science Education**, v. 17, n. 3, p. 155-169, 2007/09/01 2007.

SIMON et al. **Predictors of success in a first programming course**. Proceedings of the 8th Australasian Conference on Computing Education - Volume 52. Hobart, Australia: Australian Computer Society, Inc.: 189-196 p. 2006.

SKUDDER, B.; LUXTON-REILLY, A. **Worked examples in computer science**. Proceedings of the Sixteenth Australasian Computing Education Conference - Volume 148. Auckland, New Zealand: Australian Computer Society, Inc.: 59-64 p. 2014.

SOLOWAY, E. Learning to program = learning to construct mechanisms and explanations. **Commun. ACM**, v. 29, n. 9, p. 850-858, 1986.

SOLOWAY, E.; SPOHRER, J. C. **Studying the novice programmer**. Psychology Press, 1989. ISBN 0805800034.

SORVA, J.; SEPPÄLÄ, O. **Research-based design of the first weeks of CS1**. Proceedings of the 14th Koli Calling International Conference on Computing Education Research. Koli, Finland: ACM: 71-80 p. 2014.

SPOHRER, J. C.; SOLOWAY, E. Novice mistakes: are the folk wisdoms correct? In: (Ed.). **Studying the novice programmer**, 1989. p.401-416. ISBN 0805800034.

SPOHRER, J. C.; SOLOWAY, E.; POPE, E. A goal/plan analysis of buggy pascal programs. **Hum.-Comput. Interact.**, v. 1, n. 2, p. 163-207, 1985.

SWELLER, J. Cognitive load theory, learning difficulty, and instructional design. **Learning and Instruction**, v. 4, n. 4, p. 295-312, // 1994.

SWELLER, J.; COOPER, G. A. The Use of Worked Examples as a Substitute for Problem Solving in Learning Algebra. **Cognition and Instruction**, v. 2, n. 1, p. 59-89, 1985.

SWELLER, J.; VAN MERRIENBOER, J. G.; PAAS, F. W. C. Cognitive Architecture and Instructional Design. **Educational Psychology Review**, v. 10, n. 3, p. 251-296, 1998/09/01 1998.

TEW, A. E.; MCCracken, W. M.; GUZDIAL, M. Impact of alternative introductory courses on programming concept understanding. Proceedings of the first international workshop on Computing education research, 2005. ACM. p.25-35.

TOBAR, C. M.; OLIVEIRA, C. L.; DE FREITAS, R. L. An explanation tool to support learning of basic programming. Frontiers in Education Conference, 36th Annual, 2006. IEEE. p.21-26.

VANLEHN, K. Cognitive skill acquisition. **Annu Rev Psychol**, v. 47, p. 513-39, 1996.

VIHAVAINEN, A.; MILLER, C. S.; SETTLE, A. **Benefits of Self-explanation in Introductory Programming**. Proceedings of the 46th ACM Technical Symposium on Computer Science Education. Kansas City, Missouri, USA: ACM: 284-289 p. 2015.

VILNER, T.; ZUR, E.; SAGI, R. **Integrating video components in CS1**. Proceedings of the 43rd ACM technical symposium on Computer Science Education. Raleigh, North Carolina, USA: ACM: 123-128 p. 2012.

WANG, T.-C. et al. Teaching programming concepts to high school students with alice. Frontiers in Education Conference, 2009. FIE'09. 39th IEEE, 2009. IEEE. p.1-6.

WATSON, C.; LI, F. W. B. **Failure rates in introductory programming revisited**. Proceedings of the 2014 conference on Innovation & technology in computer science education. Uppsala, Sweden: ACM: 39-44 p. 2014.

WATSON, C.; LI, F. W. B.; GODWIN, J. L. **No tests required: comparing traditional and dynamic predictors of programming success**. Proceedings of the 45th ACM technical symposium on Computer science education. Atlanta, Georgia, USA: ACM: 469-474 p. 2014.

WIEDENBECK, S. **Factors affecting the success of non-majors in learning to program**. Proceedings of the first international workshop on Computing education research. Seattle, WA, USA: ACM: 13-24 p. 2005.

WINSLOW, L. E. Programming pedagogy—a psychological overview. **SIGCSE Bull.**, v. 28, n. 3, p. 17-22, 1996.

WIRTH, N. Program development by stepwise refinement. **Commun. ACM**, v. 14, n. 4, p. 221-227, 1971.

WITTWER, J.; RENKL, A. Why Instructional Explanations Often Do Not Work: A Framework for Understanding the Effectiveness of Instructional Explanations. **Educational Psychologist**, v. 43, n. 1, p. 49-64, 2008/01/28 2008.

WONG, R. M. F.; LAWSON, M. J.; KEEVES, J. The effects of self-explanation training on students' problem solving in high-school mathematics. **Learning and Instruction**, v. 12, n. 2, p. 233-262, 4// 2002.

WYLIE, R.; KOEDINGER, K.; MITAMURA, T. **Extending the self-explanation effect to second language grammar learning**. Proceedings of the 9th International Conference of the Learning Sciences - Volume 1. Chicago, Illinois: International Society of the Learning Sciences: 57-64 p. 2010.

ZHU, X.; SIMON, H. A. Learning mathematics from examples and by doing. **Cognition and Instruction**, v. 4, n. 3, p. 137--166, 1987.

Apêndice A – Termo de consentimento

Em sua tese de doutorado, a aluna Viviane Aureliano tem por objetivo investigar como alunos iniciantes em programação podem estudar para uma disciplina de programação de uma maneira mais efetiva através de gravações em vídeos. Os vídeos são dinâmicos e, por esta razão, eles tem sido propostos como um tipo ideal de material instrucional a partir do qual iniciantes em programação podem aprender as habilidades em programação. Mesmo com as vantagens apresentadas, os professores não podem garantir que os alunos estão estudando apropriadamente para alcançar as necessidades de um curso introdutório de programação. Uma maneira de assegurar que os alunos vão estudar apropriadamente é através da utilização de estratégias que envolvam os alunos enquanto eles estão estudando através de vídeos.

Para avaliar a efetividade dessas estratégias, faz-se necessário voluntários iniciantes em programação que estudem de acordo com o que as estratégias propõem através da realização de um experimento. Tal experimento contará com a realização de 5 atividades em sequência. Na primeira atividade serão coletadas informações demográficas e de experiência de estudo do usuário. Na segunda atividade, o voluntário realizará uma atividade de programação. A terceira atividade consistirá em um momento de estudo de linguagens de programação através de vídeos. Na quarta atividade, o voluntário realizará uma outra atividade de programação. Por último, na quinta atividade coletaremos as opiniões do voluntário a respeito do experimento.

Como você se encaixa no perfil de aluno iniciante em programação, a aluna Viviane Aureliano ficará imensamente grata caso queira participar desse experimento. Por estas razões, solicitamos seu consentimento para a realização deste experimento, bem como para a coleta de dados. Para tanto, é importante que você tenha algumas informações adicionais:

- a) Os dados coletados durante o estudo destinam-se **estritamente** a atividades de pesquisa e desenvolvimento.
- b) A divulgação dos resultados provenientes do estudo pauta-se no **respeito a sua privacidade**, e o **anonimato** dos mesmos é preservado em quaisquer documentos que venham a ser elaborados.
- c) O consentimento para a participação neste estudo é uma **escolha livre**, feita mediante a prestação de todos os esclarecimentos necessários sobre a pesquisa realizada. Você tem **toda** liberdade para interromper a sua participação no momento em que desejar.
- d) A aluna Viviane Aureliano encontra-se disponível para contato através do e-mail vcoa@cin.ufpe.br.

De posse das informações acima, gostaríamos que você se pronunciasse acerca da realização do estudo apresentado.

- () Dou meu consentimento para sua realização.
 () Não autorizo sua realização.

Recife, _____

Participante	Pesquisadora
Nome: _____	Nome: Viviane Cristina Oliveira Aureliano
Assinatura: _____	Assinatura: _____

Apêndice B – Pré-testes e pós-testes utilizados

A seguir são apresentados os pré-testes (exercícios iniciais) e os pós-testes (exercícios finais) utilizados nos três quase-experimentos utilizados na versão refinada da *Stepwise Self-explanation*.

Primeiro quase-experimento (2 aulas extras)

Nome: _____

Exercício inicial

Instruções para realização do exercício inicial:

- todas as questões devem ser escritas na linguagem de programação C;
 - as questões podem ser implementadas utilizando-se o Code Blocks / Dev-C++;
 - nenhum material além desse caderno de exercícios ou o Code Blocks / Dev-C++ poderá ser utilizado;
 - não esqueça de explicar por escrito neste papel, para cada exercício, como o código que você implementou realiza a tarefa proposta em cada questão;
 - ao finalizar o exercício:
 - salve as questões com o seguinte formato de nome `ei_<seuNome>.c` (por exemplo, o arquivo de um aluno cujo nome é Miguel Oliveira deve ser `ei_MiguelOliveira.c`);
 - envie todas as questões feitas para o e-mail: vcoa@cin.ufpe.br;
 - o assunto do e-mail deverá ser Exercício Inicial – Grupo <N>, onde N vai ser fornecido pelo professor que estiver com você em sala de aula.
1. Construa um programa para decodificar uma frase onde cada palavra está escrita de com o primeiro caracter na sua última posição e acrescentada com o sufixo `ay`. Considere que a frase poderá armazenar até 1000 caracteres e que entre cada palavra escrita existe apenas um espaço. Ao final, exiba a frase decodificada. O programa deverá funcionar da seguinte maneira:

Entrada	Saída
rogramaçãopay	programação
asacay aday adeiramay	casa da madeira
oay atoray oeuray aay ouparay oday eiray	o rato roeu a roupa do rei

Nome: _____

Exercício final

Instruções para realização do exercício final:

- todas as questões devem ser escritas na linguagem de programação C;
 - as questões podem ser implementadas utilizando-se o Code Blocks / Dev-C++;
 - nenhum material além desse caderno de exercícios ou o Code Blocks / Dev-C++ poderá ser utilizado;
 - ao finalizar o exercício:
 - salve as questões com o seguinte formato de nome ef_<seuNome>.c (por exemplo, o arquivo de um aluno cujo nome é Miguel Oliveira deve ser ef_MiguelOliveira.c);
 - envie todas as questões feitas para o e-mail: vcoa@cin.ufpe.br;
 - o assunto do e-mail deverá ser Exercício Final – Grupo <N>, onde N vai ser fornecido pelo professor que estiver com você em sala de aula.
1. Construa um programa para decodificar uma frase onde cada palavra está escrita de maneira invertida e tendo uma letra p antes de cada letra da palavra. Considere que a frase poderá armazenar até 1000 caracteres e que entre cada palavra escrita existe apenas um espaço. Ao final, exiba a frase decodificada. O programa deverá funcionar da seguinte maneira:

Entrada	Saída
opãpçpapmpaprpqpoprppp	programação
apsapcp apdp aprpipepdpapmp	casa da madeira
op optpaprp upepoprp ap apppupoprp opd ipeprp	o rato roeu a roupa do rei

Nome: _____

Exercício inicial

Instruções para realização do exercício inicial:

- todas as questões devem ser escritas na linguagem de programação C;
- as questões podem ser implementadas utilizando-se o Code Blocks / Dev-C++;
- nenhum material além desse caderno de exercícios ou o Code Blocks / Dev-C++ poderá ser utilizado;
- ao finalizar o exercício:
 - salve as questões com o seguinte formato de nome ei_<seuNome>.c (por exemplo, o arquivo de um aluno cujo nome é Miguel Oliveira deve ser ei_MiguelOliveira.c);
 - envie todas as questões feitas para o e-mail: vcoa@cin.ufpe.br;
 - o assunto do e-mail deverá ser Exercício Inicial – Grupo <N>, onde N vai ser fornecido pelo professor que estiver com você em sala de aula.

1. Construa um programa que é uma versão simplificada do jogo SuperMario Bros. O programa deve receber um valor inteiro que representa as dimensões de um tabuleiro quadrado e o próprio tabuleiro que é uma matriz de caracteres, onde:

- O caracter ‘.’ representa um espaço vazio;
- O caracter ‘m’ representa uma moeda; e
- O caracter ‘C’ representa um cogumelo.

Considere que:

- a dimensão do tabuleiro poderá ser até 100 x 100 caracteres;
- que o Mario começa no canto superior esquerdo do tabuleiro;
- ele percorre toda a linha da direita para a esquerda até chegar ao lado direito do tabuleiro;
- depois ele desce uma posição e percorre toda linha, desta vez da direita para a esquerda; e
- vai repetindo esses dois últimos passos até acabar de percorrer o tabuleiro.

O Mário percorre o tabuleiro por completo e a medida que vai percorrendo ele vai coletando as moedas. Quando ele encontra um cogumelo, ele perde todas as moedas coletadas. O programa deve imprimir como resultado a quantidade máxima de moedas que o Mário pode coletar antes de encontrar um cogumelo. O programa deverá funcionar da seguinte maneira:

Entrada	Saída
4 .mm. ...C ..mm C.mm	4
3 .mC mmC ..m	3

Nome: _____

Exercício final

Instruções para realização do exercício final:

- todas as questões devem ser escritas na linguagem de programação C;
- as questões podem ser implementadas utilizando-se o Code Blocks / Dev-C++;
- nenhum material além desse caderno de exercícios ou o Code Blocks / Dev-C++ poderá ser utilizado;
- ao finalizar o exercício:
 - salve as questões com o seguinte formato de nome ef_<seuNome>.c (por exemplo, o arquivo de um aluno cujo nome é Miguel Oliveira deve ser ef_MiguelOliveira.c);
 - envie todas as questões feitas para o e-mail: vcoa@cin.ufpe.br;
 - o assunto do e-mail deverá ser Exercício Final – Grupo <N>, onde N vai ser fornecido pelo professor que estiver com você em sala de aula.

1. Construa um programa que é uma versão simplificada do jogo Sonic. O programa deve receber um valor inteiro que representa as dimensões de um tabuleiro quadrado e o próprio tabuleiro que é uma matriz de caracteres, onde:

- O caracter ‘.’ representa um espaço vazio;
- O caracter ‘o’ representa um anel de ouro; e
- O caracter ‘E’ representa um espinho.

Considere que:

- a dimensão do tabuleiro poderá ser até 100 x 100 caracteres;
- que o Sonic começa no canto superior esquerdo do tabuleiro;
- ele percorre toda a linha da esquerda para a direita até chegar ao lado direito do tabuleiro;
- depois ele desce uma posição e percorre toda linha, desta vez da direita para a esquerda; e
- vai repetindo esses dois últimos passos até acabar de percorrer o tabuleiro.

O Sonic percorre o tabuleiro por completo e a medida que vai percorrendo ele vai coletando os anéis de ouro. Quando ele encontra um espinho, ele perde todos os anéis de ouro coletados. O programa deve imprimir como resultado a quantidade máxima de anéis de ouro que o Sonic pode coletar antes de encontrar um fantasma. O programa deverá funcionar da seguinte maneira:

Entrada	Saída
4 .oo. ..oo ..oo Eooo	9
3 .Eo ooE oo.	4

Segundo quase-experimento (4 aulas extras)

Nome: _____

Exercício inicial

Instruções para realização do exercício inicial:

- todas as questões devem ser escritas na linguagem de programação C;
 - as questões podem ser implementadas utilizando-se o Code Blocks / Dev-C++;
 - nenhum material além desse caderno de exercícios ou o Code Blocks / Dev-C++ poderá ser utilizado;
 - não esqueça de explicar por escrito neste papel, para cada exercício, como o código que você implementou realiza a tarefa proposta em cada questão;
 - ao finalizar o exercício:
 - salve as questões com o seguinte formato de nome ei_<seuNome>.c (por exemplo, o arquivo de um aluno cujo nome é Miguel Oliveira deve ser ei_MiguelOliveira.c);
 - envie todas as questões feitas para o e-mail: vcoa@cin.ufpe.br;
 - o assunto do e-mail deverá ser Exercício Inicial – Grupo <N>, onde N vai ser fornecido pelo professor que estiver com você em sala de aula.
1. Construa um programa que recebe um valor e executa repetidas vezes de 1 até esse valor inclusive. No decorrer da execução do programa, ele vai solicitando um valor ao usuário a cada iteração. Se o valor digitado pelo usuário adicionado ao índice corrente do laço for divisível por 5, imprima o valor resultante. Ao final da sua execução, o programa deve imprimir o somatório dos valores impressos para o usuário. O programa deverá funcionar da seguinte maneira:

Entrada	Saída
Digite um número: 3	
Digite o número: 9	O valor resultante é 10.
Digite o número: -5	-
Digite o número: 2	O valor resultante é 5.
	O somatório desses números é: 15.

Nome: _____

Exercício final

Instruções para realização do exercício final:

- todas as questões devem ser escritas na linguagem de programação C;
 - as questões podem ser implementadas utilizando-se o Code Blocks / Dev-C++;
 - nenhum material além desse caderno de exercícios ou o Code Blocks / Dev-C++ poderá ser utilizado;
 - não esqueça de explicar por escrito neste papel, para cada exercício, como o código que você implementou realiza a tarefa proposta em cada questão;
 - ao finalizar o exercício:
 - salve as questões com o seguinte formato de nome ef_<seuNome>.c (por exemplo, o arquivo de um aluno cujo nome é Miguel Oliveira deve ser ef_MiguelOliveira.c);
 - envie todas as questões feitas para o e-mail: vcoa@cin.ufpe.br;
 - o assunto do e-mail deverá ser Exercício Final – Grupo <N>, onde N vai ser fornecido pelo professor que estiver com você em sala de aula.
1. Construa um programa que recebe um valor e executa repetidas vezes de 1 até esse valor inclusive. No decorrer da execução do programa, ele vai solicitando um valor ao usuário a cada iteração. Se o valor digitado adicionado ao índice corrente do laço for divisível por 7, imprima o valor resultante. Ao final da sua execução, o programa deve imprimir o somatório dos valores apresentados aos usuário. O programa deverá funcionar da seguinte maneira:

Entrada	Saída
Digite um número: 3	
Digite o número: 6	O valor resultante é 7.
Digite o número: -5	-
Digite o número: 12	O valor resultante é 14.
	O somatório desses números é: 21.

Nome: _____

Exercício inicial

Instruções para realização do exercício inicial:

- todas as questões devem ser escritas na linguagem de programação C;
- as questões podem ser implementadas utilizando-se o Code Blocks / Dev-C++;
- nenhum material além desse caderno de exercícios ou o Code Blocks / Dev-C++ poderá ser utilizado;
- ao finalizar o exercício:
 - salve as questões com o seguinte formato de nome ei_<seuNome>.c (por exemplo, o arquivo de um aluno cujo nome é Miguel Oliveira deve ser ei_MiguelOliveira.c);
 - envie todas as questões feitas para o e-mail: vcoa@cin.ufpe.br;
 - o assunto do e-mail deverá ser Exercício Inicial – Grupo <N>, onde N vai ser fornecido pelo professor que estiver com você em sala de aula.

1. Construa um programa que receba a média de um número arbitrário de alunos (quando alguém digitar -1 é a condição de parada). A cada iteração do programa, ele mostra a mensagem indicada na tabela abaixo.

Média	Mensagem
Entre 0.0 e 3.0	Você foi reprovado.
Entre 3.0 e 7.0	Você vai para final.
Entre 7.0 e 10.0	Você foi aprovado.

Aos alunos que foram para a final, calcule a nota que deverão tirar para serem aprovados, considerando que a média exigida é 6.0. Além disso, o programa deverá informar também a quantidade de alunos e a média das médias das notas que eles tiraram. O programa deverá funcionar da seguinte maneira:

Entrada	Saída
Digite a média do aluno 1: 5.0	Você vai para final. Deve tirar nota 7.0 para ser aprovado.
Digite a média do aluno 2: 7.5	Você foi aprovado.
Digite a média do aluno 3: 2.0	Você foi reprovado.
Digite a média do aluno 4: -1	Quantidade de alunos: 3 Média das médias: 4,83

Nome: _____

Exercício final

Instruções para realização do exercício final:

- todas as questões devem ser escritas na linguagem de programação C;
 - as questões podem ser implementadas utilizando-se o Code Blocks / Dev-C++;
 - nenhum material além desse caderno de exercícios ou o Code Blocks / Dev-C++ poderá ser utilizado;
 - ao finalizar o exercício:
 - salve as questões com o seguinte formato de nome ef_<seuNome>.c (por exemplo, o arquivo de um aluno cujo nome é Miguel Oliveira deve ser ef_MiguelOliveira.c);
 - envie todas as questões feitas para o e-mail: vcoa@cin.ufpe.br;
 - o assunto do e-mail deverá ser Exercício Final – Grupo <N>, onde N vai ser fornecido pelo professor que estiver com você em sala de aula.
1. Construa um programa que receba o salário de um número arbitrário de funcionários (quando alguém digitar -1 é a condição de parada). A cada iteração do programa, ele deve receber o salário, calcular e mostrar o novo salário de acordo com a tabela abaixo.

Faixa Salarial	% de aumento
Até R\$ 800,00	50%
De R\$ 800,00 até R\$ 1000,00	40%
De R\$ 1000,00 até R\$ 1200,00	30%
De R\$ 1200,00 até R\$ 1400,00	20%
De R\$ 1400,00 até R\$ 1600,00	10%
Acima de R\$ 1600,00	5%

Além disso, o programa deverá também informar a quantidade de funcionários e a média salarial dos funcionários da empresa. O programa deverá funcionar da seguinte maneira:

Entrada	Saída
Digite o salário do funcionário 1: R\$ 800,00	Seu aumento será de 50% Seu novo salário será: R\$ 1200,00
Digite o salário do funcionário 2: R\$ 1300,00	Seu aumento será de 20% Seu novo salário será: R\$ 1560,00
Digite o salário do funcionário 3: R\$ 1700,00	Seu aumento será de 5% Seu novo salário será: R\$ 1785,00
Digite o salário do funcionário 4: R\$ -1	Quantidade de funcionários: 3 Média salarial: R\$ 1515,00

Nome: _____

Exercício inicial

Instruções para realização do exercício inicial:

- todas as questões devem ser escritas na linguagem de programação C;
 - as questões podem ser implementadas utilizando-se o Code Blocks / Dev-C++;
 - nenhum material além desse caderno de exercícios ou o Code Blocks / Dev-C++ poderá ser utilizado;
 - ao finalizar o exercício:
 - salve as questões com o seguinte formato de nome ei_<seuNome>.c (por exemplo, o arquivo de um aluno cujo nome é Miguel Oliveira deve ser ei_MiguelOliveira.c);
 - envie todas as questões feitas para o e-mail: vcoa@cin.ufpe.br;
 - o assunto do e-mail deverá ser Exercício Inicial – Grupo <N>, onde N vai ser fornecido pelo professor que estiver com você em sala de aula.
1. Construa um programa que receba a quantidade recebe a quantidade de números inteiros que devem ser armazenados em um vetor (máximo de 150 números inteiros). Depois, o programa percorre esse vetor, copiando os valores armazenados no vetor da seguinte maneira: se o número for divisível por 3 armazene no segundo vetor o resultado da divisão do número por 3; se o número não for divisível por 3 armazene o resto de sua divisão por 3. Por último, o programa exibe os valores armazenados no segundo vetor na ordem inversa de armazenamento. O programa deverá funcionar da seguinte maneira:

Entrada	Saída
Digite a quantidade de valores: 3 Digite o valor: 1 Digite o valor: 2 Digite o valor: 3	O vetor resultante é: 1 2 1
Digite a quantidade de valores: 4 Digite o valor: 4 Digite o valor: 8 Digite o valor: 5 Digite o valor: 6	O vetor resultante é: 2 2 2 1

Nome: _____

Exercício final

Instruções para realização do exercício final:

- todas as questões devem ser escritas na linguagem de programação C;
 - as questões podem ser implementadas utilizando-se o Code Blocks / Dev-C++;
 - nenhum material além desse caderno de exercícios ou o Code Blocks / Dev-C++ poderá ser utilizado;
 - ao finalizar o exercício:
 - salve as questões com o seguinte formato de nome ef_<seuNome>.c (por exemplo, o arquivo de um aluno cujo nome é Miguel Oliveira deve ser ef_MiguelOliveira.c);
 - envie todas as questões feitas para o e-mail: vcoa@cin.ufpe.br;
 - o assunto do e-mail deverá ser Exercício Final – Grupo <N>, onde N vai ser fornecido pelo professor que estiver com você em sala de aula.
1. Construa um programa que receba a quantidade recebe a quantidade de números inteiros que devem ser armazenados em um vetor (máximo de 50 números inteiros). Depois, o programa percorre esse vetor, copiando os valores armazenados no vetor da seguinte maneira: se o índice da posição do número no vetor for par armazene no segundo vetor o quadrado do número armazenado; se o índice da posição do número no vetor for ímpar armazene o cubo do número armazenado. Por último, o programa exibe os valores armazenados no segundo vetor na ordem inversa de armazenamento. O programa deverá funcionar da seguinte maneira:

Entrada	Saída
Digite a quantidade de valores: 3 Digite o valor: 1 Digite o valor: 2 Digite o valor: 3	O vetor resultante é: 9 8 1
Digite a quantidade de valores: 4 Digite o valor: 4 Digite o valor: 8 Digite o valor: 5 Digite o valor: 6	O vetor resultante é: 216 25 512 16

Nome: _____

Exercício inicial

Instruções para realização do exercício inicial:

- todas as questões devem ser escritas na linguagem de programação C;
 - as questões podem ser implementadas utilizando-se o Code Blocks / Dev-C++;
 - nenhum material além desse caderno de exercícios ou o Code Blocks / Dev-C++ poderá ser utilizado;
 - ao finalizar o exercício:
 - salve as questões com o seguinte formato de nome ei_<seuNome>.c (por exemplo, o arquivo de um aluno cujo nome é Miguel Oliveira deve ser ei_MiguelOliveira.c);
 - envie todas as questões feitas para o e-mail: vcoa@cin.ufpe.br;
 - o assunto do e-mail deverá ser Exercício Inicial – Grupo <N>, onde N vai ser fornecido pelo professor que estiver com você em sala de aula.
1. Construa um programa que receba as dimensões NXM de uma matriz e recebe os números inteiros que devem ser armazenados nessa matriz (máximo de 50X50 números inteiros). Depois, o programa percorre essa matriz para exibir apenas aqueles valores que são pares na mesma posição em que eles se encontram. Além disso, o programa deve exibir a soma desses valores pares armazenados na matriz. O programa deverá funcionar da seguinte maneira:

Entrada	Saída
Digite as dimensões da matriz: 3 3 Digite o valor: 1 Digite o valor: 2 Digite o valor: 3 Digite o valor: 3 Digite o valor: 2 Digite o valor: 1 Digite o valor: 1 Digite o valor: 2 Digite o valor: 3	A matriz original é: 1 2 3 3 2 1 1 2 3 Os números pares da matriz são: 2 2 2 A soma deles é: 6
Digite as dimensões da matriz: 2 3 Digite o valor: 1 Digite o valor: 3 Digite o valor: 3 Digite o valor: 1 Digite o valor: 1 Digite o valor: 3	A matriz original é: 1 3 3 1 1 3 Os números pares da matriz são: A soma deles é: 0

Nome: _____

Exercício final

Instruções para realização do exercício final:

- todas as questões devem ser escritas na linguagem de programação C;
 - as questões podem ser implementadas utilizando-se o Code Blocks / Dev-C++;
 - nenhum material além desse caderno de exercícios ou o Code Blocks / Dev-C++ poderá ser utilizado;
 - ao finalizar o exercício:
 - salve as questões com o seguinte formato de nome ef_<seuNome>.c (por exemplo, o arquivo de um aluno cujo nome é Miguel Oliveira deve ser ef_MiguelOliveira.c);
 - envie todas as questões feitas para o e-mail: vcoa@cin.ufpe.br;
 - o assunto do e-mail deverá ser Exercício Final – Grupo <N>, onde N vai ser fornecido pelo professor que estiver com você em sala de aula.
1. Construa um programa que receba as dimensões NXM de uma matriz e recebe os números inteiros que devem ser armazenados nessa matriz (máximo de 150X150 números inteiros). Depois, o programa percorre essa matriz para exibir apenas aqueles valores que são positivos na mesma posição em que eles se encontram. Além disso, o programa deve exibir a soma desses valores positivos armazenados na matriz. O programa deverá funcionar da seguinte maneira:

Entrada	Saída
Digite as dimensões da matriz: 4 2 Digite o valor: 1 Digite o valor: -2 Digite o valor: 3 Digite o valor: -3 Digite o valor: 2 Digite o valor: -1 Digite o valor: 1 Digite o valor: -2	A matriz original é: 1 -2 3 -3 2 -1 1 -2 Os números positivos da matriz são: 1 3 2 1 A soma deles é: 7
Digite a dimensão da matriz: 2 3 Digite o valor: -1 Digite o valor: -3 Digite o valor: -3 Digite o valor: -1 Digite o valor: -1 Digite o valor: -3	A matriz original é: -1 -3 -3 -1 -1 -3 Os números positivos da matriz são: A soma deles é: 0

Terceiro quase-experimento (minicurso)

Nome: _____

Exercício inicial

Instruções para realização do exercício inicial:

- todas as questões devem feitas utilizando-se o Scratch;
- nenhum material além desse caderno de exercícios ou o Scratch poderá ser utilizado;
- ao finalizar o exercício:
 - salve as questões com o programa resultante com o seguinte formato ei_qn_seuNome (por exemplo, o arquivo para a questão 1 de um aluno cujo nome é Miguel Oliveira deve ser ei_q1_MiguelOliveira);
 - envie todas as questões feitas para o e-mail: vcoa@cin.ufpe.br;
 - o assunto do e-mail deverá ser Exercício Inicial – Grupo <N>, onde N vai ser fornecido pelo professor que estiver com você em sala de aula.

1. Faça o que se pede abaixo. Um vídeo mostrando o comportamento completo do programa que você tem que implementar nesta questão está disponível em <http://bit.ly/1KQmae3>
 - a) Construa um programa que contenha um dinossauro^(a) que se movimenta através das setas direcionais (setas para cima, para baixo, para a direita, para a esquerda).
 - b) Continue o programa anterior criando três borboletas do tipo 1^(b) que devem ser comidas pelo dinossauro quando elas forem tocadas por ele enquanto ele se movimenta.
 - c) Continue o programa anterior para que a cada borboleta do tipo 1 que é comida pelo dinossauro, o jogador ganhe 10 pontos.
 - d) Continue o programa anterior para que no início do programa:
 - i) o dinossauro vá para a posição inicial (x: -160, y: -110);
 - ii) a pontuação seja inicializada com zero.
 - e) Continue o programa anterior acrescentando três borboletas do tipo 2^(c) que também devem ser comidas pelo dinossauro quando elas forem tocadas por ele enquanto ele se movimenta.
 - f) No entanto, estas borboletas fazem mal ao dinossauro quando são comidas por ele. Desta maneira, continue o programa anterior, para que a cada borboleta do tipo 2 que é comida pelo dinossauro, o jogador perca 10 pontos.
 - g) Continue o programa anterior para que a cada borboleta comida, o dinossauro emita um som.
 - h) Continue o programa anterior para que quando a tecla espaço for pressionada, o dinossauro também vá para a sua posição inicial (x: -160, y: -110).
 - i) Continue o programa anterior e crie uma outra variável que indica a quantidade de borboletas dos tipos 1 e 2 que foram comidas pelo dinossauro.
 - j) Continue o programa anterior para que quando todas as borboletas forem comidas, o jogo informe que o jogador finalizou e ganhou o jogo.



Dinosaur2

(a)



Butterfly1

(b)



Butterfly2

(c)

Nome: _____

Exercício final

Instruções para realização do exercício inicial:

- todas as questões devem feitas utilizando-se o Scratch;
- nenhum material além desse caderno de exercícios ou o Scratch poderá ser utilizado;
- ao finalizar o exercício:
 - salve as questões com o programa resultante com o seguinte formato ef_qn_seuNome (por exemplo, o arquivo para a questão 1 de um aluno cujo nome é Miguel Oliveira deve ser ef_q1_MiguelOliveira);
 - envie todas as questões feitas para o e-mail: vcoa@cin.ufpe.br;
 - o assunto do e-mail deverá ser Exercício Final – Grupo <N>, onde N vai ser fornecido pelo professor que estiver com você em sala de aula.

2. Faça o que se pede abaixo. Um vídeo mostrando o comportamento completo do programa que você tem que implementar nesta questão está disponível em <http://bit.ly/1IwTgSx>
 - k) Construa um programa que contenha um tubarão^(a) que se movimenta através das setas direcionais (setas para cima, para baixo, para a direita, para a esquerda).
 - l) Continue o programa anterior criando três peixes do tipo 2^(b) que devem ser comidos pelo tubarão quando eles forem tocados por ele enquanto ele se movimenta.
 - m) Continue o programa anterior para que a cada peixe do tipo 2 que é comido pelo tubarão, o jogador ganhe 10 pontos.
 - n) Continue o programa anterior para que no início do programa:
 - i) o tubarão vá para a posição inicial (x: -160, y: -110);
 - ii) a pontuação seja inicializada com zero.
 - o) Continue o programa anterior acrescentando três peixes do tipo 3^(c) que também devem ser comidos pelo tubarão quando eles forem tocados por ele enquanto ele se movimenta.
 - p) No entanto, estes peixes fazem mal ao tubarão quando são comidos por ele. Desta maneira, continue o programa anterior, para que a cada peixe do tipo 3 que é comido pelo tubarão, o jogador perca 10 pontos.
 - q) Continue o programa anterior para que a cada peixe comido, o tubarão emita um som.
 - r) Continue o programa anterior para que quando a tecla espaço for pressionada, o tubarão também vá para a sua posição inicial (x: -160, y: -110).
 - s) Continue o programa anterior e crie uma outra variável que indica a quantidade de peixes dos tipos 1 e 2 que foram comidos pelo tubarão.
 - t) Continue o programa anterior para que quando todos os peixes forem comidos, o jogo informe que o jogador finalizou e ganhou o jogo.



Shark

(a)



Fish2

(b)



Fish3

(c)

Apêndice C – Artigos publicados

Os artigos a seguir foram publicados durante o período desta tese:

- AURELIANO, V. C. O.; TEDESCO, P. C. D. A. R. Avaliando o uso do Scratch como abordagem alternativa para o processo de ensino-aprendizagem de programação. CONGRESSO DA SOCIEDADE BRASILEIRA DE COMPUTAÇÃO, 2012a.
- AURELIANO, V. C. O.; TEDESCO, P. C. D. A. R. Ensino-aprendizagem de Programação para Iniciantes: uma Revisão Sistemática da Literatura focada no SBIE e WIE. Anais do Simpósio Brasileiro de Informática na Educação, 2012b.
- AURELIANO, V. C. O.; TEDESCO, P. C. D. A. R. Utilizando o Scratch para Apoiar o Processo de Ensino-aprendizagem de Programação para Iniciantes na EaD. XXXIII CONGRESSO DA SOCIEDADE BRASILEIRA DE COMPUTAÇÃO 2013.
- AURELIANO, V. C. O. Self-explaining from video recordings as a methodology for learning programming. Proceedings of the tenth annual conference on International computing education research. Glasgow, Scotland, United Kingdom: ACM: 139-140 p. 2014.
- AURELIANO, V. C. O. Self-explaining from videos as a methodology for learning programming. Psychology of Programming Interest Group. Brighton 2014.
- OLIVEIRA, E. A. et al. Digital and collaborative technologies for smarter distance education. Revista de Sistemas de Informação da FSMA, v. 14, p. 39-47, 2014.
- AURELIANO, V. C. O.; TEDESCO, P. C. D. A. R. Aprendendo linguagem de programação através da auto-explicação de exemplos em vídeo. Proceedings of the IV Congresso Brasileiro de Informática na Educação. Maceió - Alagoas - Brasil 2015.

Avaliando o uso do Scratch como abordagem alternativa para o processo de ensino-aprendizagem de programação

Viviane Cristina Oliveira Aureliano^{1,2}
Patrícia Cabral de Azevedo Restelli Tedesco¹

¹Centro de Informática – Universidade Federal de Pernambuco (UFPE)
50740-560 – Recife – PE – Brasil

²Instituto Federal de Pernambuco – Campus Belo Jardim (IFPE)
55150-000 – Belo Jardim – PE – Brasil
{vcoa,pcart}@cin.ufpe.br

Abstract. *Teaching how to program is one of the big challenges in Computing Education. In this context, this paper presents a preliminary study aimed at evaluating the learning environment Scratch as an alternative approach to the traditional teaching programming process. In order to evaluate the efficacy of this alternative approach, we have compared it to traditional one. The results obtained suggest that it is a good practice to adopt these two approaches together. Such a way of teaching should be investigated in further detail, especially with regards to students' grades and participation in the classroom.*

Resumo. *Ensinar programação é um dos grandes desafios na Educação em Computação. Neste âmbito, este artigo apresenta um estudo preliminar que objetiva avaliar o ambiente introdutório de ensino Scratch como abordagem alternativa para o processo de ensino-aprendizagem de programação, contrastando esta abordagem a uma abordagem tradicional de ensino. Os resultados obtidos sugerem que uma estratégia que adote estas duas abordagens em conjunto deve ser investigada mais detalhadamente no que diz respeito às notas obtidas e também em outros aspectos, tais como a motivação e participação dos alunos em sala de aula.*

1. Introdução

O ensino de programação para iniciantes de cursos da área de tecnologia é uma preocupação que vem crescendo ao longo dos anos em virtude das dificuldades pelas quais grande parte dos alunos passa ao iniciar suas vidas como programadores. Por isso, ensinar programação é um dos sete grandes desafios na educação em Computação [Caspersen e Kölling 2009].

Anos de pesquisas na área mostram que são várias as dificuldades pelas quais passam os alunos. Conceitos que dependem de um entendimento do programa por inteiro, tais como a elaboração de um projeto de um programa, a modularização ou a retirada de erros [Lahtinen, Ala-Mutka e Järvinen 2005], e conceitos abstratos, como recursão e ponteiros [Lahtinen, Ala-Mutka e Järvinen 2005; Milne e Rowe 2002], são algumas das primeiras dificuldades citadas pela literatura. Métodos tradicionais de ensino que não consideram as particularidades de aprendizado de cada um dos alunos e a maneira

inadequada de estudo também são mencionados [Gomes e Mendes 2007]. Contudo, o maior dos problemas está relacionado à combinação e à utilização apropriada dos conceitos básicos de programação para a construção de um programa [Caspersen e Kölling 2009; Lahtinen, Ala-Mutka e Järvinen 2005]. Desta forma, os alunos parecem entender os conceitos e as estruturas que compõem uma linguagem de programação, mas não sabem o momento correto de utilizá-las durante a elaboração dos seus programas.

Pesquisas também apontam vários fatores de sucesso no ensino de programação. Alguns trabalhos afirmam que possuir experiência prévia com programação tem um efeito positivo no sucesso de cursos introdutórios de programação [Byrne e Lyons 2001; Hagan e Markham 2000]. Outros mostram que a habilidade matemática e a exposição a cursos matemáticos são importantes indicativos do bom desempenho obtido em disciplinas introdutórias nos cursos de Ciência da Computação [Bergin e Reilly 2005; Byrne e Lyons 2001; Wilson e Shrock 2001]. Relações entre os diferentes estilos de aprendizagem dos alunos e o aprendizado de programação também foram encontradas. Thomas e co-autores (2002), por exemplo, listam como um dos problemas do ensino em programação que o aprendizado somente pode ser melhorado quando os professores levarem em consideração que cada aluno possui um estilo de aprendizado diferente e, por isso, necessidades e preferências de aprendizado particulares.

Aliada às pesquisas anteriormente citadas, outra linha que tem ganhado destaque dentro da área de Educação em linguagens de programação inclui a construção de ferramentas que apóiam o processo de ensino e aprendizagem dos estudantes. Entre elas, algumas das mais conhecidas são: Alice [Cooper, Dann e Pausch 2000], Greenfoot [Henriksen e Kölling 2004] e Scratch [Resnick *et al* 2009]. Classificadas como “Ambientes Introdutórios de Ensino” - *Initial Learning Environments* [Fincher *et al.* 2010], todas elas apresentam uma linguagem de programação visual e têm por objetivo apoiar os seus usuários no aprendizado de conceitos de programação de uma maneira mais divertida e estimulante. Dentre estas ferramentas, a introdução do Scratch como ferramenta de apoio ao ensino de uma disciplina introdutória de programação na Universidade de Harvard foi encarada como positiva para o aprendizado de alunos iniciantes na disciplina [Malan e Leiner 2007].

Apesar das vantagens relatando o ambiente Scratch como parte integrante de uma abordagem alternativa para o ensino de programação para iniciantes, não foi encontrado nenhum estudo que avalie os benefícios de sua utilização nesse contexto, comparando-o a uma abordagem tradicional de ensino. O estudo preliminar apresentado neste trabalho, então, tem por objetivo avaliar o processo de ensino-aprendizagem de programação para alunos iniciantes através da utilização do Scratch, a qual denominamos abordagem alternativa. Tal abordagem deve ser contrastada com uma abordagem tradicional que envolve a utilização da linguagem de programação C e o ambiente de desenvolvimento Dev-C++ [Dev-C++ 2012].

O restante do artigo está organizado da seguinte maneira. Na seção 2 é feita a apresentação da ferramenta Scratch. O estudo realizado neste trabalho é descrito na seção 3. Na seção 4 são apresentados os resultados. Por último, na seção 5 são apresentados as conclusões e os trabalhos futuros.

2. Scratch

Desenvolvido pelo *Lifelong Kindergarten Group (LLK)*, grupo de pesquisa do *MIT Media Lab*, o ambiente introdutório de ensino Scratch foi criado com o propósito de introduzir a programação de maneira fácil e rápida para aqueles que não possuem nenhum tipo de experiência no assunto [Maloney et al. 2010]. Para tanto, o processo de ensino-aprendizagem, com a utilização do Scratch, ocorre através de uma linguagem de programação visual que permite a manipulação de mídias, tais como imagens e músicas, para a criação de histórias interativas, de jogos ou de animações, por exemplo [Maloney et al. 2010; Maloney et al. 2008].

Um projeto no Scratch é formado por um palco e por diferentes objetos. O palco é o plano de fundo estático onde os objetos podem executar as suas ações. Para dar vida a seus projetos e definir o comportamento de cada um dos objetos presentes, os usuários associam a estes objetos sons, imagens e variáveis que podem ser manipulados pelos diferentes tipos de comandos fornecidos pela linguagem de programação do ambiente. Os comandos do Scratch precisam somente ser arrastados e soltados no espaço de comandos, onde, posteriormente, podem ser encaixados uns nos outros. Os comandos no Scratch assemelham-se a peças de quebra-cabeça e indicam com sua forma como podem ser combinados corretamente. Desta maneira, os comandos quando combinados formam programas que estão sempre sintaticamente corretos. Sem a preocupação com erros sintáticos, os usuários focam apenas na criação da lógica de funcionamento de seus projetos [Malan e Leitner 2007].



Figura 1. Ambiente introdutório de ensino Scratch.

O Scratch vem sendo aplicado em diferentes projetos e vem contribuindo positivamente para a criatividade, a resolução de problemas e a colaboração de seus usuários [Resnick et al. 2009; Maloney et al. 2008; Resnick 2002]. Através da criação de histórias interativas, jogos e animações, os proponentes do Scratch afirmam que é possível tornar as pessoas que fazem uso deste ambiente, digitalmente fluentes, de maneira tal que elas passam a ser construtoras e não somente usuárias de tecnologia [Resnick et al. 2009]. Além disso, acreditando na facilidade de programação fornecida pela ferramenta, Malan e Leitner (2007) obtiveram resultados positivos na utilização do Scratch como ferramenta de apoio ao ensino da disciplina introdutória de programação na Universidade de Harvard.

A linguagem de programação C, por sua vez, foi desenvolvida em 1973 por Dennis M. Ritchie para a construção do sistema operacional Unix e continua sendo muito popular

até os dias atuais. Também bastante conhecido, o ambiente de programação Dev-C++ é utilizado para a edição e compilação de programas escritos na linguagem C. Ao contrário dos programas feitos no Scratch, para construir os seus programas em C, um aluno precisa conhecer as palavras-chave da linguagem e a maneira como estas palavras formam uma estrutura sintaticamente correta. Consequentemente, além de se preocupar com a lógica de funcionamento dos seus programas, os alunos precisam também estar preocupados com as particularidades de uma linguagem de programação convencional. Preocupações como estas envolvem, por exemplo, usar corretamente um laço de repetição *for* ou finalizar uma instrução com um ponto-e-vírgula (;). Para ilustrar a diferença entre as duas linguagens e seus ambientes, apresentamos nas Figuras 1 e 2, o programa mais simples que comumente é ensinado nas primeiras aulas de disciplinas introdutórias de programação em cada um dos ambientes.

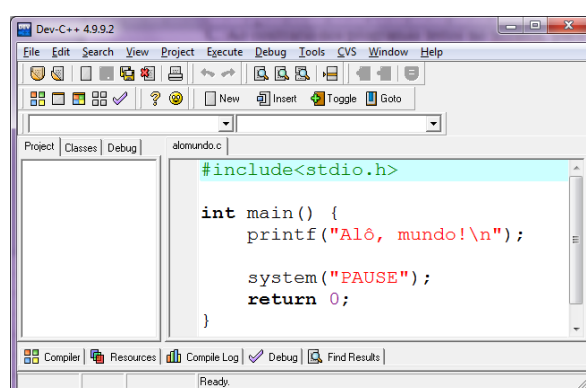


Figura 2. Programa simples escrito em C utilizando o Dev-C++.

Acreditamos que ensinar programação utilizando o Scratch possibilita um melhor desempenho do ponto de vista das notas obtidas por alunos iniciantes em programação.

3. Descrição do estudo

O estudo preliminar apresentado neste trabalho tem por objetivo avaliar o processo de ensino-aprendizagem de programação para alunos iniciantes através de uma abordagem alternativa de ensino que utiliza o Scratch como linguagem de programação e ambiente de desenvolvimento. Tal abordagem deve ser contrastada com uma abordagem tradicional que envolve a utilização da linguagem de programação C e o ambiente de desenvolvimento Dev-C++. Sendo assim, a questão que deve ser respondida neste trabalho é a seguinte: *Ensinar programação utilizando o ambiente de programação visual Scratch possibilita um melhor desempenho para os alunos iniciantes em programação?* O desempenho dos alunos será avaliado em função das notas obtidas por eles em cada uma das abordagens.

3.1. Participantes

O estudo foi realizado em uma instituição de ensino que atua na educação de nível técnico e superior no Estado de Pernambuco. O estudo ocorreu no segundo semestre de 2011 e contou com a participação de 29 alunos matriculados na disciplina de Lógica de programação, disciplina introdutória de programação de um curso técnico profissionalizante em Informática da referida instituição. Neste curso, a disciplina de Lógica de Programação vem seguindo uma abordagem de ensino tradicional. Nesta

abordagem, as aulas são ministradas utilizando-se uma linguagem de programação imperativa, como Pascal ou C, e um ambiente de desenvolvimento tradicional, como o Pascalzim ou o DEV-C++. Neste segundo semestre de 2011 e nesta turma, em especial, a disciplina estava sendo ministrada com a linguagem C e com o ambiente de desenvolvimento, Dev-C++.

Para a execução do estudo, os alunos matriculados na disciplina de Lógica de Programação foram divididos em dois grupos através da realização de um sorteio. Com o primeiro grupo foi adotada a abordagem alternativa e, por este motivo, ele foi nomeado grupo A. Com o segundo grupo foi adotada a abordagem tradicional e, por isso, ele foi nomeado grupo T. O grupo A continha 15 alunos, no entanto, os dados de apenas 13 deles puderam ser aproveitados. Destes 13 alunos, 11 eram homens e 2 eram mulheres. As idades dos alunos desse grupo variavam de 17 a 31 anos, com média de 22,23 anos. Nenhum deles tinha experiência com o Scratch nem tinha conhecimento sobre o conteúdo abordado na aula. O grupo T continha 14 alunos, todos homens. Neste grupo, as idades variavam de 18 a 36 anos, com média de 23,07 anos. Além disso, ainda no grupo T, três alunos haviam participado de disciplinas de programação anteriormente, dois deles porque estavam repetindo a disciplina e o último porque cursava o Curso Superior em Análise e Desenvolvimento de Sistemas em outra instituição de ensino. Destes três, apenas um deles conseguiu mostrar através de suas respostas que estava familiarizado com o assunto. Os outros 11 alunos nunca tinham estudado nada sobre o conteúdo abordado.

3.2. Execução

O estudo foi executado em dois dias distintos. No primeiro dia, como os alunos participantes do grupo A não conheciam o ambiente Scratch, foi realizada uma ambientação neste ambiente. Logo após, o grupo A passou por uma aula da disciplina sobre o conteúdo *arrays* utilizando a abordagem alternativa. O conteúdo *arrays* foi escolhido por ser considerado um dos tópicos mais difíceis de serem ensinados por professores de disciplinas que envolvem programação [Dale, 2006].

Como era necessário que todos os alunos submetidos a este estudo passassem pela experiência com uma linguagem de programação mais tradicional não somente nos conteúdos abordados anteriormente na disciplina, mas também no conteúdo abordado durante o estudo, foi impossível deixar o grupo A de fora do segundo dia do estudo. Assim, neste segundo dia, o grupo T se juntou ao grupo A, e ambos assistiram a uma aula também sobre o conteúdo *arrays* ministrada na abordagem tradicional. Para garantir a consistência do estudo, as aulas ministradas nas duas abordagens seguiram o mesmo roteiro. No início de cada aula foi realizada uma avaliação para medir o nível de conhecimento que os alunos possuíam sobre o conteúdo abordado. Da mesma forma, ao final da execução de cada aula, foi realizada uma avaliação com o objetivo de medir o nível de aprendizado dos alunos com cada uma das abordagens.

Ambientação com o Scratch com o Grupo A

Neste primeiro momento, participaram apenas os alunos do grupo A. Como eles não tinham nenhuma experiência prévia com o Scratch, esta fase do estudo consistiu de uma ambientação com o ambiente Scratch. Nesta ambientação, foram abordados os seguintes tópicos: entrada e saída de dados, variáveis, operadores aritméticos e relacionais,

estruturas de repetição e estruturas condicionais. Estes tópicos já haviam sido abordados durante as aulas anteriores, contudo, utilizando-se apenas a abordagem tradicional. Assim, como os alunos não sabiam como eles deveriam realizar para utilizar tais estruturas na nova ferramenta, mostrou-se necessário apresentar como elas eram executadas utilizando o Scratch. Em seguida à aula propriamente dita, os alunos foram guiados por uma pequena lista de exercícios para que pudessem experimentar a ferramenta. A ambientação com o Scratch durou aproximadamente 1 hora e 15 minutos.

Aula seguindo abordagem alternativa com o Grupo A

Logo após a ambientação com o Scratch, ainda no mesmo dia, ocorreu uma aula do conteúdo *arrays* seguindo a abordagem alternativa que utiliza o Scratch como ambiente e linguagem de programação. Esta etapa do estudo contou também apenas com a participação do grupo A que foi submetido às seguintes etapas: avaliação pré-aula, aula propriamente dita e avaliação pós-aula. A avaliação pré-aula teve como objetivo coletar o conhecimento prévio que os alunos possuíam a respeito do conteúdo *arrays*. A avaliação pós-teste, por sua vez, teve o intuito de avaliar o aprendizado dos alunos sobre o conteúdo ministrado através da abordagem alternativa. Para efeito de comparação das abordagens, os resultados obtidos na avaliação pós-aula da abordagem alternativa serão confrontados com os resultados obtidos na avaliação pós-aula da abordagem tradicional. Durante a aula propriamente dita, os seguintes tópicos foram abordados: definição e declaração de *array*, inserção e remoção de elementos, substituição de elementos, referência a um elemento, tamanho do *array* e verificação se um determinado elemento está contido no *array*. Nesta fase, como nenhum dos alunos tinha conhecimento sobre o assunto *arrays*, a avaliação pré-aula durou apenas 15 minutos. A aula foi ministrada em aproximadamente 1 hora e, posteriormente, foi disponibilizado em torno de 1 hora para que os alunos respondessem as avaliações pós-aula.

Aula seguindo abordagem tradicional com os Grupos A e T

No dia seguinte, os grupos A e T participaram da aula, também sobre o conteúdo *arrays*, seguindo a abordagem tradicional que utilizou a linguagem de programação C e o ambiente de programação Dev-C++. Como toda a turma precisava passar pela linguagem de programação C, não foi possível deixar o grupo A de fora desta fase do estudo. Nesta aula, os grupos A e T foram submetidos a um procedimento semelhante ao da etapa anterior, ou seja, ambos os grupos passaram pelas etapas de avaliação pré-aula, aula propriamente dita e avaliação pós-aula. As avaliações realizadas pelo grupo T neste dia eram exatamente as mesmas que foram realizadas pelo grupo A na etapa anterior, ao passo que as avaliações realizadas pelo grupo A eram equivalentes no conteúdo, mas continham diferentes questões. Nesta aula, um dos participantes do grupo T já possuía algum conhecimento sobre o assunto *arrays*, por isso, a avaliação pré-aula teve duração maior do que a da aula anterior, de aproximadamente 35 minutos. A aula foi ministrada em aproximadamente 1 hora e 45 minutos e, posteriormente, também foi disponibilizado em torno de 1 hora para que os alunos respondessem as avaliações pós-aula.

3.3. Análise do estudo

Para medir o aprendizado dos alunos em ambas as abordagens, ao finalizar as aulas seguindo as diferentes abordagens, foi aplicada uma avaliação pós-aula que mensurou o conhecimento dos alunos em relação ao conteúdo que foi ministrado. Estas avaliações envolveram questões conceituais e de implementação sobre o assunto *arrays*. Logo após o estudo, as avaliações pós-aula foram corrigidas e, por possuir 12 questões, elas foram pontuadas com uma nota de 0 a 12. Estas notas, atribuídas às avaliações pós-aula, serviram então como dados para a análise estatística realizada neste estudo. As avaliações pré-aula, por sua vez, foram utilizadas apenas para verificarmos efetivamente se os alunos não possuíam conhecimento algum sobre o assunto abordado nas aulas. Apenas um aluno do grupo T conseguiu mostrar através das respostas da sua prova que estava familiarizado com o assunto.

Para a análise dos dados coletados, foi escolhida a distribuição t de Student, uma vez que o conceito de aleatoriedade foi adotado para a organização dos alunos nos dois grupos participantes do estudo. A distribuição t de Student é um teste estatístico usado para comparar os valores das médias entre os dois conjuntos de dados e analisar se a diferença entre eles é significativa. O nível de confiança adotado para a análise dos dados foi de 95%.

4. Resultados

Os dados apresentados na Tabela 1 referem-se às notas obtidas nas avaliações pós-aula pelo grupo A na aula que seguiu a abordagem alternativa e pelo grupo T na aula que seguiu a abordagem tradicional. Nota-se que a média das notas obtidas pelos participantes do grupo A foi maior do que a média das notas obtidas pelos participantes do grupo T. No entanto, os participantes do grupo A apresentaram um rendimento mais irregular do que os alunos do grupo T. Quando estes dados são analisados em conjunto, não se pode afirmar que a diferença entre as notas do grupo A e do grupo T é significativa. Consequentemente, nada se pode afirmar entre as abordagens alternativa e tradicional nesta situação.

Tabela 1. Dados entre os grupos A e T nas abordagens alternativa e tradicional.

	Média	Variância	$ t_0 $
Grupo A na abordagem alternativa	4.38	8.60	
Grupo T na abordagem tradicional	2.93	1.29	
			1.72

Os dados apresentados na Tabela 2 referem-se às notas obtidas nas avaliações pós-aula pelo grupo A nas aulas que seguiram as abordagens alternativa e tradicional. Nota-se que a média das notas obtidas pelos participantes do grupo A na aula que seguiu a abordagem alternativa foi ligeiramente maior do que a média das notas obtidas pelos mesmos participantes na aula que seguiu a abordagem tradicional. No entanto, na abordagem alternativa estes participantes apresentaram um rendimento mais irregular do que na abordagem tradicional. Semelhantemente ao primeiro caso, quando estes dados são analisados em conjunto, não se pode afirmar que a diferença entre as notas do grupo A nas duas abordagens é significativa para que se possa inferir uma diferença no grupo nas diferentes situações.

Tabela 2. Dados do grupo A nas abordagens alternativa e tradicional.

	Média	Variância	$ t_0 $
Grupo A na abordagem alternativa	4.38	8.60	
Grupo A na abordagem tradicional	4.03	4.95	
			0.33

Por último, na Tabela 3 são apresentados os dados referentes às notas obtidas pelos grupos A e T na aula que seguiu a abordagem tradicional. Observa-se que a média das notas obtidas pelos participantes do grupo A foi maior do que a média das notas obtidas pelos participantes do grupo T. Além disso, neste caso, as notas dos participantes dos grupos A e T também variaram de maneira pouco regular. Da mesma forma que para os casos anteriores, quando os dados obtidos foram analisados em conjunto, notou-se que não houve uma diferença significativa entre as notas destes grupos na abordagem tradicional.

Tabela 3. Dados comparativos entre os grupos A e T na abordagem tradicional.

	Média	Variância	$ t_0 $
Grupo A na abordagem tradicional	4.03	4.95	
Grupo T na abordagem tradicional	2.93	1.29	
			1.63

Os dados obtidos neste estudo não confirmaram a hipótese inicial apresentada neste trabalho. Contudo, é interessante comentar alguns dos resultados encontrados. Sabemos que os resultados apresentados por Hagam e Markham (2000) indicam que ter alguma experiência prévia em programação favorece o aprendizado da disciplina. Quando o grupo A passa duas vezes pela experiência de aula com o mesmo assunto, ele acaba possuindo uma experiência prévia com o assunto abordado. Analisando-se apenas a variância das notas do grupo A nas abordagens alternativa e tradicional, nota-se que houve um decréscimo da abordagem alternativa para a abordagem tradicional. Consequentemente, pode-se afirmar que houve uma maior regularidade nas notas do grupo A na abordagem tradicional do que na abordagem alternativa. Apesar de não haver uma diferença significativa entre as notas do grupo A nas duas abordagens, tal aumento na regularidade das notas sugere que uma investigação maior da estratégia que foi adotada com o grupo A deve ser realizada. Podemos assim descobrir que outras consequências ocorrem ao se iniciar uma disciplina introdutória de programação com uma abordagem alternativa seguida por uma abordagem tradicional não somente do ponto de vista das notas obtidas. Por exemplo, podemos analisar de que maneira o raciocínio lógico-matemático pode ser reforçado com uma abordagem alternativa antes da abordagem tradicional ou como o aprendizado de alguns assuntos considerados difíceis pelos alunos pode ser facilitado com este tipo de estratégia.

Além das notas obtidas, também podemos fazer algumas considerações sobre a maneira como as aulas foram ministradas. A aula seguindo a abordagem alternativa teve a duração de aproximadamente 1 hora, enquanto a aula seguindo a abordagem tradicional durou aproximadamente 1 hora e 45 minutos. Assim, pode-se perceber que a aula seguindo a abordagem alternativa fluiu mais rapidamente, por conta de características próprias da ferramenta Scratch, como por exemplo, comandos que podem ser facilmente combinados, programas que não geram erros sintáticos e linguagem de programação

escrita em português. Além disso, a interface gráfica do Scratch permite que o programa seja visualizado enquanto é executado. Consequentemente, a sensação que se tem é que a interface gráfica facilita a apresentação do conteúdo por parte do professor e também a assimilação desse conteúdo por parte do aluno, uma vez que se consegue exibir o funcionamento do programa na memória e na interação com o usuário. Por outro lado, a aula mais demorada na abordagem tradicional se deve principalmente à dificuldade de se repassar o conteúdo através de um código escrito e explicar o funcionamento deste código associando-o a uma representação gráfica sem o auxílio de uma ferramenta computacional. Além disso, erros de sintaxe e a linguagem de programação escrita em inglês, acabam tornando o aprendizado dos alunos mais penoso. Desta maneira, acreditamos que há uma influência positiva do uso do Scratch como ambiente de programação para iniciar um curso introdutório em programação, já que o processo de ensino-aprendizado parece ser facilitado em alguns aspectos. Posteriormente, o curso pode seguir seu fluxo normal com uma abordagem mais tradicional que utilize uma linguagem de programação mais comercial, como C ou Java, por exemplo. Como o estudo neste trabalho apresenta dados preliminares, posteriormente, pretendemos realizar novos estudos que analisem outros aspectos, tais como a utilização do Scratch do ponto de vista da motivação e participação dos alunos em sala de aula, por exemplo.

5. Conclusões e Trabalhos futuros

Neste trabalho, apresentamos um estudo preliminar no qual avaliamos o ambiente introdutório de ensino Scratch como abordagem alternativa para o processo de ensino-aprendizagem de programação. Supomos que passando por uma abordagem alternativa como a proposta aqui, os alunos teriam um desempenho mais satisfatório do ponto de vista das notas obtidas do que passando por uma abordagem tradicional usando a linguagem de programação C e o ambiente de desenvolvimento Dev-C++. Entretanto, essa hipótese não pode ser confirmada com os dados obtidos neste estudo.

Ao invés disso, os resultados encontrados sugerem que seja realizada uma investigação mais detalhada das consequências que ocorrem ao se iniciar uma disciplina introdutória de programação com uma abordagem alternativa seguida por uma abordagem tradicional. Essa investigação poderá ser realizada não somente do ponto de vista das notas obtidas. Ela poderá analisar também como o raciocínio lógico-matemático dos alunos pode ser melhorado ou como o aprendizado de assuntos considerados difíceis pode ser facilitado com esta estratégia.

Como trabalhos futuros, realizaremos uma investigação mais detalhada da estratégia de ministrar uma disciplina introdutória de programação utilizando a abordagem alternativa seguida pela abordagem tradicional. A ideia é que o experimento seja executado em uma quantidade maior de aulas. Consequentemente, as abordagens poderão ser avaliadas através de um conteúdo programático mais extenso. Ademais, pretendemos investigar a utilização de outros ambientes de desenvolvimento para programação na linguagem C para compor a abordagem tradicional. Por fim, também pretendemos avaliar a utilização do Scratch do ponto de vista da motivação e da participação dos alunos em sala de aula.

Referências

- Bennedsen, J.; Caspersen, M.E. (2009) Recalling Programming Competence, Proceedings do 9th Koli Calling International Conference on Computing Education Research, Koli National Park, Finland, p. 86-95.
- Byrne, P.; Lyons, G. (2001) The effect of student attributes on success in programming. ACM SIGCSE Bulletin, vol. 33, n. 3, p. 49-52.
- Cooper, S.; Dann, W.; Pausch, R. (2000) Alice: A 3-D tool for introductory programming concepts. Journal of Computing Sciences in Small Colleges, vol.15, n. 5, p.107-116.
- Dale, N. B. (2006) Most difficult topics in CS1: results of an online survey of educators, ACM SIGCSE Bulletin, vol. 38, n. 2, p. 49-53.
- Dev-C++ (2012) Disponível em <<http://bit.ly/14aDr0>>. Acesso em: 05 mai. 2012.
- Fincher, S.; Cooper, S.; Kölling, M.; Maloney, J. (2010) Comparing alice, greenfoot & scratch. Proceedings do 41st SIGCSE'10, Milwaukee, Wiscosin, USA. p. 192-193.
- Gomes, A.; Mendes, A. J. (2007) Learning to Program - Difficulties and Solutions. Proceedings do ICEE 2007, Coimbra, Portugal. Disponível em: <<http://bit.ly/Lj3HLu>>. Acesso em: 05 mai. 2012.
- Henriksen, P.; Kölling, M. (2004) Greenfoot: Combining object visualisation with interaction. Proceedings do 19th OOPSLA, Vancouver, Canadá. p. 73-82.
- Lahtinen, E.; Ala-Mutka, K.; Järvinen, H.-M. (2005) A Study of the Difficulties of Novice Programmers. Proceedings do 10th ITiCSE, Monte de Caparica, Portugal, p. 14-18.
- Malan, D. J.; Leitner, H. H. (2007) Scratch for budding computer scientists. Proceedings do 38th SIGCSE'07, Kentucky, USA, p. 223-227.
- Milne, I.; Rowe, G. (2002) Difficulties in Learning and teaching Programming - Views of Students and Tutors. Education and Information Technologies, vol. 7, n. 1, p. 55-66.
- Maloney, J.; Resnick, M.; Rusk, N.; Silverman, B.; Eastmond, E. (2010) The scratch programming language and environment. ACM Transactions on Computing Education, vol. 10, n. 4, article 16, 15 pages.
- Resnick, M.; Maloney, J.; Monroy-Hernández, A.; Rusk, N.; Eastmond, E.; Brennan, K.; Millner, A.; Rosenbaum, E.; Silver, J.; Silverman, B.; Kafai, Y. (2009) Scratch: programming for all. Communications of the ACM, vol. 52, n. 11, p. 60-67.
- Thomas, L.; Woodbury, J.; Jarman, E. (2002) Learning styles and performance in the introductory programming sequence. Proceedings do 33th SIGCSE'2002, Covington, USA. p. 33-37.
- Wilson, B. C.; Shrock, S. (2001) Contributing to success in an introductory computer science course: a study of twelve factors. ACM SIGCSE Bulletin, vol. 33, n. 1, p. 184-188.

Ensino-aprendizagem de Programação para Iniciantes: uma Revisão Sistemática da Literatura focada no SBIE e WIE

Viviane Cristina Oliveira Aureliano^{1,2}
Patrícia Cabral de Azevedo Restelli Tedesco¹

¹Centro de Informática – Universidade Federal de Pernambuco (UFPE)
50740-560 – Recife – PE – Brasil

²Instituto Federal de Pernambuco – *Campus* Belo Jardim (IFPE)
55150-000 – Belo Jardim – PE – Brasil
{vcoa,pcart}@cin.ufpe.br

Abstract. *This paper presents the results of a systematic literature review which included the analysis of articles on teaching-learning programming process for beginners published in the last 10 years (2002-2011) in two important national events in the area, namely the Brazilian Symposium on Computers in Education (SBIE) and Workshop on Computers in Schools (WIE). The results show that there was an increase in the interest of the Brazilian community in the area, that most research is being conducted by researchers located in the South and Southeast of Brazil. Furthermore, results also show that the main research focus is on higher education and that most research presents new software tools.*

Resumo. *Este trabalho apresenta os resultados de uma revisão sistemática da literatura que contou com a análise dos artigos sobre o processo de ensino-aprendizagem de programação para iniciantes publicados nos últimos 10 anos (2002-2011) em dois importantes eventos nacionais na área, o Simpósio Brasileiro de Informática na Educação (SBIE) e o Workshop de Informática na Escola (WIE). Os resultados obtidos mostraram que houve um aumento no interesse da comunidade brasileira na área, que a maioria das pesquisas está sendo realizada por pesquisadores localizados nas regiões Sul e Sudeste do Brasil, que as pesquisas focam principalmente na educação superior e que grande parte delas apresenta novas ferramentas de software.*

1. Introdução

Várias são as dificuldades pelas quais passam os alunos durante o processo de ensino-aprendizagem de programação. A modularização do programa ou a retirada de erros e conceitos mais abstratos, como recursão e ponteiros são alguns das dificuldades citadas na literatura [Lahtinen, Ala-Mutka e Järvinen 2005; Milne e Rowe 2002]. Contudo, a maior delas está relacionada à combinação e à utilização apropriada dos conceitos básicos de programação para a construção de um programa [Caspersen e Kölling 2009; Lahtinen, Ala-Mutka e Järvinen 2005]. Desta forma, os alunos parecem entender os conceitos e as estruturas que compõem uma linguagem de programação, mas não sabem como utilizá-las corretamente durante a construção de seus próprios programas.

Pesquisas também apontam fatores de sucesso no ensino de programação. Alguns trabalhos afirmam que possuir experiência prévia com programação tem um efeito positivo no sucesso de cursos introdutórios de programação [Byrne e Lyons 2001; Hagan e Markham 2000]. Outros mostram que a habilidade matemática e a exposição a cursos matemáticos são importantes indicativos do bom desempenho dos alunos nestas disciplinas [Bergin e Reilly 2005; Byrne e Lyons 2001; Wilson e Shrock 2001].

No Brasil, o debate sobre o ensino de programação possui um discurso semelhante ao apresentado pela literatura internacional. No entanto, nenhum panorama sobre o assunto foi publicado nos veículos científicos nacionais na área em que educação e informática são tratadas conjuntamente. Neste contexto, há uma nítida necessidade de se analisar a produção científica nacional sobre o processo de ensino-aprendizagem de programação para iniciantes. Para alcançar este objetivo, neste trabalho iremos apresentar os resultados de uma revisão sistemática da literatura (RSL) que contou com a análise dos artigos sobre o processo de ensino-aprendizagem de programação para iniciantes publicados nos últimos 10 anos (2002-2011) em dois importantes eventos nacionais na área, o Simpósio Brasileiro de Informática na Educação (SBIE) e o Workshop de Informática na Escola (WIE).

O restante do artigo está organizado da seguinte maneira. Na seção 2 é feita a apresentação do método adotado nesta RSL. Os resultados são apresentados na seção 3. Por último, na seção 4 são apresentados as conclusões e os trabalhos futuros.

2. Método

Segundo Kitchenham (2007), uma RSL é realizada para identificar, avaliar e interpretar os estudos que estejam disponíveis e que sejam relevantes a uma determinada questão de pesquisa. Ela pode ser realizada com dois objetivos principais: identificar lacunas existentes em uma área de pesquisa ou fornecer um conjunto relevante de trabalhos relacionados para embasar novas pesquisas.

2.1. Questões de pesquisa

Com estes objetivos em mente, a RSL aqui realizada tem como questão central de pesquisa a seguinte pergunta: *Qual o panorama atual das publicações científicas no Brasil sobre o processo de ensino-aprendizagem de programação para iniciantes?* Para responder a essa questão, foram definidas as seguintes questões específicas de pesquisa:

- QP1: Quais são as instituições de pesquisa envolvidas na área e como elas estão distribuídas pelo Brasil?
- QP2: Quais são os níveis de escolaridade para os quais as pesquisas na área estão sendo destinadas?
- QP3: Quais são os tipos de artefatos propostos nas pesquisas realizadas na área?

2.2. Critérios de inclusão e exclusão

A pesquisa foi realizada através de busca manual nos anais dos eventos SBIE e WIE nos últimos 10 anos. Esta busca foi executada em duas etapas. A pré-seleção dos artigos consistiu em verificar os sites ou CDs contendo os anais dos referidos eventos e acessar manualmente todos os artigos completos, lendo os seus títulos, resumos e palavras-chave. Depois da pré-seleção, em uma segunda etapa, todos os artigos pré-selecionados

foram analisados pela primeira autora deste trabalho e a cada um deles foram aplicados os critérios de inclusão e exclusão que são apresentados na Tabela 1. A aplicação de apenas um dos critérios de exclusão ou inclusão determinou se o artigo deveria ser excluído ou incluído, respectivamente. As dúvidas provenientes da aplicação dos critérios de inclusão e exclusão foram discutidas e resolvidas com a segunda autora deste trabalho. Posteriormente, os estudos selecionados de acordo com os critérios de inclusão tiveram seus dados extraídos para fornecer as respostas das perguntas de pesquisa. Os estudos semelhantes que apresentavam a evolução de uma mesma pesquisa e que continham os mesmos autores foram removidos para evitar duplicações. Em um último momento, os dados extraídos dos artigos selecionados para inclusão foram sintetizados para apresentação dos resultados.

Tabela 1. Critérios de exclusão e de inclusão.

Critérios de exclusão	Critérios de inclusão
• Artigos que abordam o processo de ensino-aprendizagem de programação como ferramenta para alcançar objetivos diversos; • Artigos que abordam o processo de ensino-aprendizagem de programação para estudantes que não são iniciantes; • Artigos que abordam o processo de ensino-aprendizagem de programação em disciplinas diversas àquelas disciplinas introdutórias em programação; • Artigos que não sejam claramente sobre o processo de ensino-aprendizagem de programação para iniciantes.	• Artigos completos que abordam o processo de ensino-aprendizagem de programação para iniciantes.

2.3. Extração e síntese dos dados

Nesta fase, foram extraídos os seguintes dados gerais: título, autores, evento, ano, instituições de pesquisa, unidades federativas destas instituições, nível de escolaridade dos sujeitos pesquisados, disciplina e curso onde a pesquisa foi realizada. Além disso, foram extraídos trechos dos artigos que responderam as perguntas de pesquisa desta revisão: perguntas de pesquisa e objetivos, artefatos propostos, principais resultados e limitações e sugestões indicadas para pesquisas futuras.

3. Resultados

A busca resultou em 79 artigos completos, dos quais 31 foram selecionados para a extração dos dados. A Tabela 2 apresenta os resultados gerais dos processos de pré-seleção e inclusão dos artigos para o SBIE e WIE.

Tabela 2. Resultado geral das buscas para o SBIE e WIE.

Evento	Artigos pré-selecionados	Artigos incluídos	Total de artigos completos	Artigos incluídos / Total de artigos completos (%)
SBIE	48	24	723	3,32
WIE	31	7	425	1,65
Total	79	31	1148	4,97

Durante o processo de busca, não conseguimos acesso a 3 dos artigos pré-selecionados no WIE de 2004, em um total de aproximadamente 3,8% de 79 artigos que poderiam ter

sido incluídos na seleção. Estes artigos foram analisados apenas através dos seus resumos contidos em uma pequena parte dos anais impressos aos quais tivemos acesso. No entanto, isso não foi suficiente para confirmar se eles se adequariam aos critérios de inclusão caso as suas versões completas fossem analisadas. Desta forma, eles não foram considerados em nossa análise.

3.1. Distribuição temporal

O gráfico apresentado na Figura 1 mostra a distribuição temporal dos artigos incluídos nesta revisão. Apesar de não haver nenhum padrão ou tendência clara para a distribuição dos artigos ao longo dos anos, podemos notar que aproximadamente 51% deles foram publicados nos anos de 2009 a 2011. Além disso, a quantidade de artigos por ano aumentou consideravelmente de 2008 para 2009 e se estabilizou desde então. Tal observação sugere que, a partir de 2009, houve um aumento no interesse dos pesquisadores brasileiros sobre a questão do processo de ensino-aprendizagem de programação para iniciantes e que esse interesse permanece inalterado até o momento.

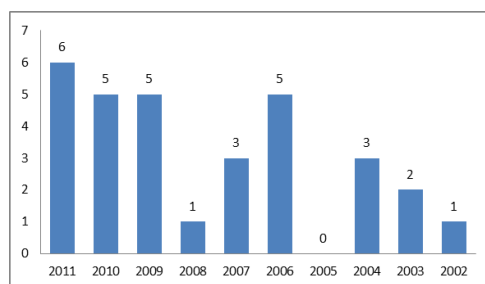


Figura 1. Distribuição temporal dos artigos.

3.2. Distribuição geográfica das instituições de pesquisa

Na Tabela 3 é apresentada a quantidade de artigos por instituições de pesquisa. Apenas 3 destas instituições, a Universidade do Vale do Itajaí (UNIVALI), o Instituto Tecnológico da Aeronáutica (ITA) e a Universidade Federal do Rio Grande do Sul (UFRGS) em parceria com a Faculdade de Tecnologia Senac Pelotas, tiveram mais de um trabalho. As outras instituições tiveram, cada uma, apenas 1 artigo nesta revisão.

Tabela 3. Quantidade de artigos por instituições de pesquisa.

Instituição	Artigos	Instituição	Artigos
UNIVALI	3	UFF	1
ITA	2	UFPA / UFRA / UFES	1
UFRGS / Faculdade de Tecnologia Senac Pelotas	2	UFPB	1
CEFET-ES / UFES	1	UFPR	1
FATEC-ID / PUC-Campinas	1	UFRPE	1
FURB / Universiteit Van Amsterdam	1	UFSC / Católica UNISANTOS	1
ITA / IMES / Mackenzie	1	UnB	1
PUC-Campinas	1	UNICAMP / UFRJ	1
UEM / CTI Renato Archer	1	UNIPLAC	1
UESC / UFES / UFRGS	1	UNIRIO	1
UFAM	1	UNISINOS / UNILASALLE	1
UFAM / UFES	1	UNIVALI / PUCRS	1
UFC	1	USP	1
UFCG / UFAL	1		

Na Figura 2 é apresentada a distribuição geográfica das instituições que realizaram as pesquisas apresentadas nos artigos. Aproximadamente 58% das pesquisas apresentadas nos artigos foram realizadas por instituições localizadas nas regiões Sul e Sudeste do Brasil, enquanto 26% dos artigos foram estudos desenvolvidos de maneira colaborativa, contando com a participação de instituições de pesquisa situadas em dois ou mais locais. Neste último caso, 75% dos trabalhos contam com a participação de instituições de pesquisa que estão localizadas nas regiões Sul e Sudeste do Brasil e um único artigo foi realizado em colaboração com uma instituição de pesquisa internacional.

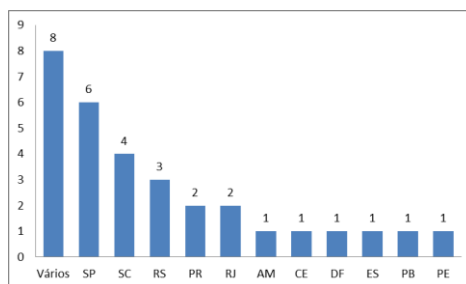


Figura 2. Distribuição geográfica das instituições de pesquisa.

3.3. Nível de escolaridade

Os artigos foram classificados de acordo com o nível de escolaridade para os quais as pesquisas se destinam. Os níveis de escolaridade indicados são apresentados no gráfico da Figura 3. Verificamos que aproximadamente 61% dos artigos abordaram estudos no contexto da educação superior. Apenas 19% dos artigos apresentaram estudos no contexto dos níveis fundamental, médio e técnico. Desta forma, estes dados mostram claramente a carência de pesquisas que abordem o processo de ensino-aprendizagem de programação para alunos iniciantes que estejam cursando os níveis fundamental, médio e técnico.

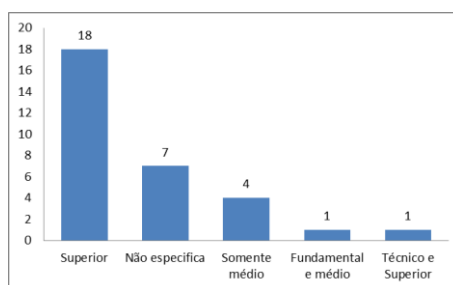


Figura 3. Níveis de escolaridade para os quais as pesquisas são destinadas.

3.4. Tipos de artefatos propostos

Para fornecer um quadro geral dos tipos de artefatos que foram propostos nos artigos, foram definidas categorias que pudessem classificar tais artefatos. Como não conhecemos uma classificação padrão que se encaixasse à necessária nesta revisão, para definir as categorias aqui apresentadas, os artefatos propostos nos artigos foram descritos brevemente. Em seguida, buscou-se por padrões nestas descrições que geraram os tipos apresentados na Tabela 4. Nos dados apresentados, percebemos que quase 65% dos artigos propõem ferramentas de software para o processo de ensino-aprendizagem

de programação para iniciantes, ou seja, há uma preferência clara pela proposição de novas ferramentas de software. As metodologias respondem por quase 23% dos artigos e a definição de novas linguagens de programação é o objetivo de aproximadamente 10% dos artigos. Os outros tipos de artefato respondem pelos 6% restantes.

Tabela 4. Tipos de artefatos propostos nos artigos.

Tipo de artefato	Descrição	Artigos
Avaliação de ferramenta	Avaliação de ferramenta de apoio à escrita cooperativa	1
Ferramentas de software	Ambiente de programação de microcontroladores	2
	Ambiente para aprendizagem cooperativa	2
	Ferramenta com técnicas de Inteligência Artificial	2
	Ambiente contendo um sistema de raciocínio baseado em casos	1
	Ambiente de programação para robôs	1
	Ambiente inteligente para melhoria de assistência ao aluno	1
	Ferramenta com gerenciamento de recursos didáticos e acompanhamento das práticas laboratoriais, correção automática	1
	Ferramenta com simulação e visualização de código, correção automática	1
	Ferramenta de acompanhamento do conteúdo programático da disciplina, correção automática de provas e programas	1
	Ferramenta de autoria e interpretação tutorial	1
	Ferramenta de datawarehouse para acompanhamento do desempenho do aluno	1
	Ferramenta de personalização e acompanhamento da resolução de exercícios	1
	Ferramenta para acompanhamento e análise de programas	1
	Ferramenta para suporte à programação em pares	1
	Ferramenta que usa padrões pedagógicos	1
	Ferramenta que usa técnicas de computação afetiva para detectar nível de frustração dos alunos	1
Linguagem de programação	Linguagem de programação icônica	1
	Linguagem de programação para estudantes surdos	1
Linguagem de programação e ferramenta	Linguagem de programação de microcontroladores, kit de robótica e agente pedagógico	1
Metodologia	Uso de jogos	2
	Estratégia para melhoria na abstração de resolução de problemas	1
	Modelo para avaliação e acompanhamento da aprendizagem	1
	Ontologia para auxiliar na revisão de programas	1
	Uso da robótica	1
	Uso de blog	1
Técnica de avaliação	Estudo da taxonomia para elaboração de um instrumento de avaliação	1

3.5. Limitações da revisão

A principal limitação que existe neste trabalho é a mais comum das limitações em revisões sistemáticas. Esta limitação diz respeito à execução do processo, com a inserção de erros e vieses em qualquer uma das etapas, primeiro, na pré-seleção dos artigos, depois, na inclusão e exclusão e, por último, na extração dos dados. Como o número de artigos analisados durante a pré-seleção é elevado, há uma grande chance de que alguns artigos importantes passem despercebidos na análise dos títulos, resumos e palavras-chave. Essa situação é ainda pior quando os títulos, resumos e palavras-chave são confusos e não refletem adequadamente os objetivos, escopo do trabalho e

principais resultados ou contribuições da pesquisa. Outra limitação está na aplicação dos critérios de inclusão e exclusão. Neste caso, para reduzir as chances de erro, as dúvidas da primeira autora nesta etapa foram discutidas e resolvidas com a segunda autora deste trabalho. Finalmente, a última limitação deste trabalho diz respeito aos 3 artigos potencialmente relevantes aos quais não tivemos acesso. Apesar de eles representarem apenas 3,8% dos artigos pré-selecionados, quando verificamos essa quantidade no conjunto de artigos incluídos, eles representam quase 10% desse total. Isso pode significar que a revisão apresentada neste trabalho não tenha contemplado todos os estudos importantes na área de ensino-aprendizagem de programação para iniciantes no contexto do SBIE e WIE.

4. Conclusões e Trabalhos Futuros

Neste trabalho, apresentamos os resultados de uma revisão sistemática da literatura sobre o processo de ensino-aprendizagem de programação para iniciantes publicados nos últimos 10 anos no SBIE e no WIE. A busca realizada nesta revisão resultou na pré-seleção de 79 artigos dos quais 31 foram incluídos para a extração de dados.

A distribuição temporal mostrou que mais da metade dos artigos foi publicada de 2009 a 2011. Além disso, de 2008 para 2009 houve um aumento considerável na quantidade de artigos publicados na área de processo de ensino-aprendizagem para iniciantes. Tais observações sugerem que, a partir de 2009, houve um aumento no interesse dos pesquisadores brasileiros com a questão do processo de ensino-aprendizagem de programação para iniciantes e que esse interesse permanece inalterado até o momento.

A distribuição geográfica das instituições de pesquisa mostrou que a maior parte delas está localizada nas regiões Sul e Sudeste do Brasil. Além disso, 3/4 das pesquisas que foram realizadas por vários grupos de maneira colaborativa contaram com a participação de instituições de pesquisas que também estão localizadas no Sul e Sudeste do Brasil e um único artigo contou com a colaboração de uma instituição de pesquisa internacional.

A discrepância entre a quantidade de artigos publicados nos contextos da educação superior e da educação nos níveis fundamental, médio e técnico mostrou claramente a carência de pesquisas que abordem o processo de ensino-aprendizagem de programação para iniciantes nos níveis médio e técnico, principalmente.

A organização apresentada para os tipos de artefatos propostos sugere que há uma preferência nítida dos pesquisadores brasileiros por construir ferramentas que apoiem o processo de ensino-aprendizagem de programação para iniciantes. Em seguida, vemos a proposição de novas metodologias e novas linguagens de programação.

Como trabalhos futuros, pretendemos continuar a análise dos dados dos artigos incluídos, passando para o último passo que uma RSL possui: a análise da qualidade. Além disso, pretendemos expandir esta revisão englobando outros importantes veículos de publicação na área de processo de ensino-aprendizagem de programação para iniciantes, tais como o Workshop de Educação em Computação (WEI) e a Revista Brasileira de Informática na Educação (RBIE).

Referências

- Bergin, S. and Reilly, R. (2005) Programming: factors that influence success. ACM SIGCSE Bulletin, New York, v. 37, n. 1, p. 411-415.
- Byrne, P.; Lyons, G. (2001) The effect of student attributes on success in programming. ACM SIGCSE Bulletin, vol. 33, n. 3, p. 49-52.
- Caspersen, M. E.; Kölling, M. (2009) STREAM: A First Programming. Journal ACM Transactions on Computing Education (TOCE), v.9, n.1, p.4:1-4:29.
- Hagan, D.; Markham, S. (2000) Does it help to have some programming experience before beginning a computing degree program? Proceedings do 5th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education, 2000, Helsinki. New York: ACM, p. 25-28.
- Kitchenham, B. A. (2007) Guidelines for performing Systematic Literature Reviews in Software Engineering.
- Lahtinen, E.; Ala-Mutka, K.; Järvinen, H.-M. (2005) A Study of the Difficulties of Novice Programmers. Proceedings do 10th ITiCSE, Monte de Caparica, Portugal, p. 14-18.
- Milne, I.; Rowe, G. (2002) Difficulties in Learning and teaching Programming - Views of Students and Tutors. Education and Information Technologies, vol. 7, n. 1, p. 55-66.
- Wilson, B. C.; Shrock, S. (2001) Contributing to success in an introductory computer science course: a study of twelve factors. ACM SIGCSE Bulletin, vol. 33, n. 1, p. 184-188.

Artigos incluídos na revisão (em ordem cronológica)

- Ribeiro, P. C.; Martins, C. B.; Bernardini, F. C. (2011) A Robótica como Ferramenta de Apoio ao Ensino de Disciplinas de Programação em Cursos de Computação e Engenharia. In: 17º Workshop de Informática na Escola, Aracaju, Brasil.
- Marques, D. L.; Costa, L. F. S.; Silva, M. A. de A.; Rebouças, A. D. D. S. (2011) Atraindo Alunos do Ensino Médio para a Computação: Uma Experiência Prática de Introdução a Programação utilizando Jogos e Python. In: 17º Workshop de Informática na Escola, Aracaju, Brasil.
- Santos, R. E. S.; Magalhães, C. V. C.; Neto, J. S. C.; Júnior, S. S. L. P. (2011) PROGLIB - Uma Linguagem de Programação Baseada na Escrita de LIBRAS. In: 17º Workshop de Informática na Escola, Aracaju, Brasil.
- Sirotheau, S.; Brito, S. R. de; Silva, A. do S. da; Eliasquevici, M. K.; Favero, E. L.; Tavares, O. de L. (2011) Aprendizagem de iniciantes em algoritmos e programação: foco nas competências de autoavaliação. In: 22º Simpósio Brasileiro de Informática na Educação, Aracaju, Brasil.
- Iepsen, E. F.; Bercht, M.; Reategui, E. (2011) Detecção e Tratamento do Estado Afetivo Frustração do Aluno na Disciplina de Algoritmos. In: 22º Simpósio Brasileiro de Informática na Educação, Aracaju, Brasil.

- França, A. B.; Soares, J. M. (2011) Sistema de apoio a atividades de laboratório de programação via Moodle com suporte ao balanceamento de carga. In: 22º Simpósio Brasileiro de Informática na Educação, Aracaju, Brasil.
- Marques, A. de M.; Pimentel, M.; Siqueira, S. (2010) Dinâmicas Educacionais com o Uso de Blogs: Requisitos a partir de Experiências. In: 16º Workshop de Informática na Escola, Belo Horizonte, Brasil.
- Piccolo, H. L.; Sena, V. de F.; Nogueira, K. B.; Silva, M. O. da; Maia, Y. A. N. (2010) Ambiente Interativo e Adaptável para ensino de Programação. In: 21º Simpósio Brasileiro de Informática na Educação, João Pessoa, Brasil.
- Jesus, E. A. de; Raabe, A. L. A. (2010) Avaliação Empírica da Utilização de um Jogo para Auxiliar a Aprendizagem de Programação. In: 21º Simpósio Brasileiro de Informática na Educação, João Pessoa, Brasil.
- Piva Jr., D.; Freitas, R. L. (2010) Estratégias para melhorar os processos de abstração na disciplina de Algoritmos. In: 21º Simpósio Brasileiro de Informática na Educação, João Pessoa, Brasil.
- Iepson, E. F.; Bercht, M.; Reategui, E. (2010) Persona-Algo - Personalização dos Exercícios de Algoritmos auxiliados por um Agente Afetivo. In: 21º Simpósio Brasileiro de Informática na Educação, João Pessoa, Brasil.
- Benitti, F. B. V.; Vahldick, A.; Urban, D. L.; Krueger, M. L.; Halma, A. (2009) Experimentação com Robótica Educativa no Ensino Médio: ambiente, atividades e resultados. In: 15º Workshop de Informática na Escola, Porto Alegre, Brasil.
- Vieira, P. V.; Raabe, A. L. A.; Zeferino, C. A. (2009) Persona-Algo - Personalização dos Exercícios de Algoritmos auxiliados por um Agente Afetivo. In: 20º Simpósio Brasileiro de Informática na Educação, Florianópolis, Brasil.
- Barbosa, M. R. G.; Silva, F. A.; Oliveira, V. M. de A.; Feltrim, V. D.; Mirisola, L. G. B.; Gonçalves, P. C.; Ramos, J. J. G.; Alves, L. T. (2009) Implementação de Compilador e Ambiente de Programação Icônica para a Linguagem Logo em um Ambiente de Robótica Pedagógica de Baixo Custo. In: 20º Simpósio Brasileiro de Informática na Educação, Florianópolis, Brasil.
- Jesus, E. A. de; Raabe, A. L. A. (2009) Interpretações da Taxonomia de Bloom no Contexto da Programação Introdutória. In: 20º Simpósio Brasileiro de Informática na Educação, Florianópolis, Brasil.
- Sievers Jr., F.; Germano, J. S. E.; Almeida, F. de (2009) Kit Programe Fácil – Um Kit Educacional Para Subsidiar A Programação De Microcontroladores Em Sala De Aula. In: 20º Simpósio Brasileiro de Informática na Educação, Florianópolis, Brasil.
- Júnior, G. P. dos S.; Costa, E. de B. C.; Fachine, J. M. (2008) Raciocínio Baseado em Casos para auxílio a Alunos na Resolução de Problemas por Analogia – Uma abordagem para Representação e Recuperação de Casos. In: 19º Simpósio Brasileiro de Informática na Educação, Fortaleza, Brasil.
- Neto, W. C. B.; Schuvartz, A. A. (2007) Ferramenta Computacional de Apoio ao Processo de Ensino-Aprendizagem dos Fundamentos de Programação de Computadores. In: 18º Simpósio Brasileiro de Informática na Educação, São Paulo, Brasil.
- Miranda, L. C. de; Sampaio, F. F.; Borges, J. A. dos S. (2007) ProgrameFácil: Ambiente de Programação Visual para o Kit de Robótica Educacional RoboFácil. In: 18º Simpósio Brasileiro de Informática na Educação, São Paulo, Brasil.

- Borges, R. M.; Pinto, S. C. C. S.; Barbosa, J. L. V.; Barbosa, D. N. F. (2007) Usando o modelo 3C de colaboração e Vygotsky no ensino de programação distribuída em pares. In: 18º Simpósio Brasileiro de Informática na Educação, São Paulo, Brasil.
- Lopes, A. P. B.; Menezes, C. S. de (2006) Avaliando o EpA – um estudo de caso na aprendizagem de computação. In: 12º Workshop de Informática na Escola, Campo Grande, Brasil.
- Petry, P. G.; Rosatelli, M. C. (2006) Ensino e Aprendizagem de Algoritmos com o AlgoLC. In: 17º Simpósio Brasileiro de Informática na Educação, Brasília, Brasil.
- Neves, M. de F.; Coello, J. M. A.. (2006) OntoRevPro - Uma Ontologia sobre Revisão de Programas para o Aprendizado Colaborativo de Programação em Java. In: 17º Simpósio Brasileiro de Informática na Educação, Brasília, Brasil.
- Raabe, A. L. A.; Giraffa, L. M. M. (2006) Uma Arquitetura de Tutor para Promover Experiências de Aprendizagem Mediadas. In: 17º Simpósio Brasileiro de Informática na Educação, Brasília, Brasil.
- Neto, F. A. de A.; Castro, T. H. C. de; Júnior, A. N. de C. (2006) Utilizando o Método Clínico Piagetiano para Acompanhar a Aprendizagem de Programação. In: 17º Simpósio Brasileiro de Informática na Educação, Brasília, Brasil.
- Barros, L. N. de; Delgado, K. V.; Machion, A. C. G. (2004) An ITS for Programming to Explore Practical Reasoning. In: 15º Simpósio Brasileiro de Informática na Educação, Manaus, Brasil.
- Castro, T. H. C. de; Júnior, A. N. de C.; Menezes, C. S. de (2004) Aprende – um Ambiente Cooperativo de Apoio à Aprendizagem de Programação. In: 15º Simpósio Brasileiro de Informática na Educação, Manaus, Brasil.
- Pêgas, D. dos S.; Yano, E. T. (2004) O uso de data warehousing no processo de aprendizagem de programação. In: 15º Simpósio Brasileiro de Informática na Educação, Manaus, Brasil.
- Pimentel, E. P.; França, V. F. de; Noronha, R. V.; Omar, N. (2003) Avaliação Contínua da Aprendizagem, das Competências e Habilidades em Programação de Computadores. In: 9º Workshop de Informática na Escola, Campinas, Brasil.
- Santos, G. dos; Direne, A. I.; Guedes, A. L. P. (2003) Autoria e Interpretação Tutorial de Soluções Alternativas para Promover o Ensino de Programação de Computadores. In: 14º Simpósio Brasileiro de Informática na Educação, Rio de Janeiro, Brasil.
- Nobre, I. A. M.; Menezes, C. S. de (2002) Suporte à Cooperação em um Ambiente de Aprendizagem para Programação (SAmbA). In: 13º Simpósio Brasileiro de Informática na Educação, Rio de Janeiro, Brasil.

Utilizando o Scratch para Apoiar o Processo de Ensino-aprendizagem de Programação para Iniciantes na EaD

Viviane Cristina Oliveira Aureliano^{1,2}
Patrícia Cabral de Azevedo Restelli Tedesco¹

¹Centro de Informática – Universidade Federal de Pernambuco (UFPE)
50740-560 – Recife – PE – Brasil

²Instituto Federal de Pernambuco – *Campus* Belo Jardim (IFPE)
55150-000 – Belo Jardim – PE – Brasil
{vcoa,pcart}@cin.ufpe.br

Abstract. *Nowadays, one of the biggest challenges of Computer Education area is how to teach programming. In this context, this paper presents a study aimed at evaluating the potential of Scratch in the process of teaching and learning programming to beginners in distance learning from the student point of view. The results indicate that the students' opinions about Scratch were divided. However, the reasons given by these students regarding the use of Scratch constitute the most important contribution of this work.*

Resumo. *Nos dias atuais, um dos grandes desafios da área de Educação em Computação é o ensino de programação. Neste contexto, este artigo apresenta um estudo que objetivou avaliar o potencial do Scratch no processo de ensino-aprendizagem de programação para iniciantes na EaD a partir da coleta das opiniões dos alunos. Os resultados obtidos indicam que os alunos ficaram divididos em relação à contribuição do ambiente para o seu aprendizado. Entretanto, as justificativas fornecidas por estes alunos para as opiniões a respeito da utilização do Scratch constituem a contribuição mais importante deste trabalho.*

1. Introdução

Ensinar programação é um dos grandes desafios da área de Educação em Computação [Caspersen e Kölling 2009]. Dentro desta área, pesquisas apontam fatores de sucesso no ensino de programação. Alguns trabalhos afirmam que possuir experiência prévia com programação tem um efeito positivo no sucesso de cursos introdutórios de programação [Byrne e Lyons 2001; Hagan e Markham 2000]. Outros mostram que a habilidade matemática e a exposição a cursos matemáticos são importantes indicativos do bom desempenho dos alunos nestas disciplinas [Bergin e Reilly 2005; Byrne e Lyons 2001; Wilson e Shrock 2001].

Por outro lado, várias são as dificuldades pelas quais passam os alunos durante o processo de ensino-aprendizagem de programação. A modularização do programa ou a retirada de erros e conceitos mais abstratos, como recursão e ponteiros são alguns das dificuldades citadas na literatura [Lahtinen, Ala-Mutka e Järvinen 2005; Milne e Rowe 2002]. Contudo, o maior dos obstáculos está relacionado à combinação e à utilização

apropriada dos conceitos básicos de programação para a construção de um programa [Caspersen e Kölling 2009; Lahtinen, Ala-Mutka e Järvinen 2005]. Desta forma, os alunos parecem entender os conceitos e as estruturas que compõem uma linguagem de programação, mas não sabem como utilizá-las corretamente durante a construção de seus próprios programas.

Aliadas às pesquisas anteriormente citadas, algumas ferramentas que apoiam o processo de ensino-aprendizagem de programação vem sendo propostas. Construída com o objetivo de apoiar os seus usuários no aprendizado de conceitos de programação de uma maneira mais divertida e estimulante através de uma linguagem de programação visual, o ambiente introdutório de ensino Scratch vem ganhando destaque [Fincher et al. 2010; Resnick et al 2009]. A introdução do Scratch como apoio ao ensino de uma disciplina introdutória de programação na Universidade de Harvard foi encarada como positiva para o aprendizado de alunos iniciantes nesta disciplina [Malan e Leiner 2007].

Apesar dos bons resultados obtidos no cenário internacional, poucos são os estudos que avaliam o potencial do ambiente no ensino-aprendizagem de programação com estudantes brasileiros. É ainda mais difícil encontrar algum estudo que utilize o Scratch em um curso oferecido na Educação a Distância (EaD). Dentro desse contexto, o trabalho aqui apresentado tem por objetivo avaliar o potencial deste ambiente no processo de ensino-aprendizagem em programação para alunos iniciantes na EaD. A disciplina introdutória de programação pertence à matriz curricular de um curso técnico profissionalizante em informática oferecido na EaD e o potencial do Scratch é analisado através da coleta das opiniões dos alunos matriculados nesta disciplina. Para tanto, a coleta das opiniões dos estudantes foi realizada utilizando-se duas técnicas de pesquisa, a análise documental dos fóruns de discussão e um questionário que respondido pelos alunos ao final da execução da disciplina.

O restante do artigo está organizado da seguinte maneira. Na seção 2 é feita a apresentação do ambiente Scratch. O estudo realizado neste trabalho é descrito na seção 3. Na seção 4 são apresentados os resultados. Por último, na seção 5 são apresentados as conclusões e os trabalhos futuros.

2. Scratch

O Scratch foi criado pelo grupo de pesquisa *Lifelong Kindergarten (LLK)* do *MIT Media Lab* com o propósito de introduzir a programação de maneira fácil e rápida para aqueles que não possuem nenhum tipo de experiência no assunto [Maloney et al. 2010]. Para tanto, o processo de ensino-aprendizagem, com a utilização do Scratch, ocorre através de uma linguagem de programação visual que permite a manipulação de mídias, tais como imagens e sons, para a criação de jogos ou de animações interativas [Maloney et al. 2010; Maloney et al. 2008].

Para produzir os seus projetos, os usuários associam sons, imagens e variáveis aos objetos que podem ser manipulados pelos diferentes comandos fornecidos pelo Scratch. Os comandos do ambiente assemelham-se a peças de quebra-cabeça e indicam com sua forma como podem ser combinados corretamente e, para utilizá-los, os usuários precisam somente arrastá-los e soltá-los no espaço de comandos, onde são encaixados uns nos outros. Por indicar com suas formas como devem ser encaixados, os comandos do Scratch quando combinados formam programas que estão sempre sintaticamente

corretos. Então, sem a preocupação com erros sintáticos, os usuários focam apenas na criação da lógica de funcionamento de seus projetos [Malan e Leitner 2007]. Um dos primeiros programas que podem ser construídos na versão 1.4 do Scratch é apresentado na Figura 1.



Figura 1. Programa construído no ambiente de programação Scratch.

O Scratch vem sendo aplicado em diferentes projetos e vem contribuindo positivamente para a criatividade, a resolução de problemas e a colaboração de seus usuários [Resnick *et al.* 2009; Maloney *et al.* 2008]. Através da utilização do Scratch, os seus proponentes afirmam que é possível tornar as pessoas que o utilizam, digitalmente fluentes, de maneira que elas passem não somente usuárias, mas também a ser construtoras de tecnologia [Resnick *et al.* 2009]. Além disso, acreditando na facilidade de programação fornecida pela ferramenta, Malan e Leitner (2007) obtiveram resultados positivos na utilização do Scratch como ferramenta de apoio ao ensino da disciplina introdutória de programação na Universidade de Harvard.

3. Descrição do estudo

Na última edição do Workshop sobre Educação em Computação, Aureliano e Tedesco (2012) apresentaram um estudo no qual afirmam acreditar que há uma influência positiva do uso do Scratch como ambiente de apoio ao processo de ensino-aprendizagem de disciplinas introdutórias de programação. Segundo as autoras, essa influência é decorrente da visualização gráfica do programa enquanto ele é executado, o que facilita a apresentação do conteúdo ministrado em sala de aula pelo professor e a consequente assimilação desse conteúdo por parte do aluno. Para tanto, as autoras afirmam que investigar a utilização do Scratch no início de um curso introdutório de programação e, posteriormente, seguir com o fluxo normal da disciplina com a utilização de uma linguagem de programação escrita pode trazer novas conclusões a respeito da experiência de uso desse ambiente de programação em sala de aula.

Neste contexto, o estudo apresentado neste trabalho tem por objetivo analisar a experiência de utilização do Scratch no início de uma disciplina introdutória em programação. Assim, a disciplina foi dividida em dois momentos, o primeiro com a utilização do Scratch nos conteúdos iniciais da disciplina. Em seguida, a disciplina prosseguiu com a utilização de uma linguagem de programação escrita em Português, o Portugol e do ambiente de programação Visualg para tratar dos assuntos mais complexos. Tendo em vista esta dinâmica, procuramos responder a seguinte questão com este trabalho: *Qual a opinião dos alunos sobre as suas experiências ao utilizar o Scratch como ambiente de apoio ao aprendizado de programação?* O estudo foi realizado em uma instituição de ensino que atua na educação de nível técnico e superior no Estado de Pernambuco. A experiência ocorreu no segundo semestre de 2012 em uma disciplina introdutória de programação pertencente à matriz curricular de um curso técnico profissionalizante em Informática oferecido na EaD.

3.1. Participantes

Inicialmente, a disciplina contou com 78 alunos matriculados. Deste montante, 34 alunos não participaram das atividades avaliativas da disciplina e, por este motivo, foram considerados desistentes do curso. Dos 44 alunos restantes, ou seja, aqueles que realizaram pelo menos uma atividade avaliativa da disciplina, 27 foram aprovados e 17 foram reprovados. Para coletar o ponto de vista dos alunos, fizemos uso de duas técnicas de pesquisa, a análise documental dos fóruns de discussão e os questionários. Apesar de bastante incentivada, a utilização dos fóruns nesta disciplina teve uma participação pequena e apenas 22 alunos fizeram alguma postagem relacionada ao Scratch. Deste total, 10 dos alunos foram aprovados, 10 foram reprovados e 1 desistiu da disciplina. Os questionários, por sua vez, foram realizados ao final da disciplina e contou com 24 respondentes. Deste total, 15 alunos foram aprovados e 9 reprovados.

Ademais, como os questionários são ferramentas que permitem uma coleta mais detalhada de informações dos alunos, pudemos coletar outros tipos de informações dos alunos, tais como idade e se eles possuíam algum curso anterior de programação. À disciplina onde o estudo foi executado. Assim, no grupo dos 24 respondentes dos questionários, as idades variavam de 19 a 39 anos, com média de 26,29 anos. Neste mesmo grupo, 11 alunos tinham algum conhecimento de programação, pois haviam cursado minicursos ou outros cursos técnicos ou de graduação cuja matriz curricular possuía uma disciplina de programação. Os outros 13 alunos nunca haviam estudado programação anteriormente à disciplina.

3.2. Execução

O curso técnico profissionalizante no qual o estudo foi realizado é oferecido através do Ambiente Virtual de Aprendizagem (AVA) Moodle. Este curso é ofertado em um total de 4 períodos e a disciplina introdutória de programação, cujo nome é Lógica de Programação, é componente curricular obrigatório do segundo período. O curso técnico em questão foi criado em 2010 e esta disciplina foi ofertada anteriormente apenas duas vezes, obtendo um alto índice de reprovação em suas duas edições. Neste segundo semestre de 2012, a disciplina foi ministrada pela primeira autora deste trabalho que foi responsável, entre outras atividades, por planejar, desenvolver e avaliar as metodologias de ensino adotadas durante a disciplina. Esta professora teve o apoio de dois tutores à distância responsáveis por todas as atividades referentes à manutenção da comunicação e ao acompanhamento das atividades dos alunos no AVA.

Esta disciplina foi ministrada em 12 semanas letivas que foram assim distribuídas: 9 semanas onde o conteúdo programático foi apresentado, 1 semana para avaliação presencial, 1 semana para a avaliação de segunda chamada e 1 semana para avaliação final. A cada semana de conteúdo programático, um novo tópico foi abordado através da disponibilização de um conjunto de diferentes recursos, composto por livro-texto, slides, vídeo-aulas, chats e fóruns de discussão online. Além disso, com o objetivo de avaliar continuamente os estudantes ao longo da disciplina, a cada semana uma atividade avaliativa correspondente ao conteúdo vivenciado na semana corrente foi proposta.

Estas 9 semanas de conteúdo foram organizadas em dois momentos conforme apresentado na Quadro 1. Como podemos observar, nas semanas 1 a 3 foram abordados

os conteúdos iniciais da disciplina e o Scratch foi utilizado como linguagem e ambiente de programação. Estas primeiras semanas foram seguidas por uma semana de transição do Scratch para o Portugol. Nesta semana de transição, todo conteúdo abordado anteriormente foi revisto utilizando-se o Portugol como linguagem de programação e o Visualg como ambiente de programação. Logo depois, as semanas 5 a 8, que englobaram o conteúdo mais complexo da disciplina, seguiram utilizando-se o Portugol e o Visualg. Por fim, na semana 9 foi feita uma revisão de todos os tópicos abordados.

Quadro 1. Resumo do conteúdo programático abordado na disciplina.

	Semana	Conteúdo programático
Scratch	1	Introdução à programação / Tipos de dados, constantes e variáveis, operadores aritméticos, relacionais e lógicos
	2	Comandos de atribuição, entrada e saída / Estruturas condicionais
	3	Estruturas de repetição
	4	Semana de transição do Scratch para o Portugol
Portugol	5	Vetores e matrizes
	6	Procedimentos e funções / Variáveis globais e locais
	7	Passagem de parâmetros
	8	Registros

3.3. Análise do estudo

Como dito anteriormente, o estudo apresentado neste trabalho foi realizado em uma disciplina oferecida a distância. Portanto, para coletar o ponto de vista dos alunos, fizemos uso de duas técnicas de pesquisa, uma análise documental e um questionário. A análise documental realizada no estudo em questão utilizou os fóruns de discussão online da disciplina como fonte de pesquisa. Tais fóruns são comumente utilizados para que os alunos possam expressar suas opiniões, questionamentos e observações sobre a disciplina. Portanto, eles se constituem em um importante recurso para que os alunos possam exprimir os seus pontos de vista a respeito da disciplina. Sendo assim, em um primeiro momento, verificamos todas as postagens que foram realizadas pelos alunos nos fóruns de discussão online no decorrer da disciplina e selecionamos aquelas que estavam relacionadas diretamente à utilização do Scratch. Em seguida, durante a análise dos dados, classificamos e categorizamos todas as postagens selecionadas para a apresentação dos resultados alcançados.

Ao final da execução da disciplina, os alunos foram convidados a responder a um questionário através da ferramenta Google Forms. Nesta ferramenta, as respostas ao questionário são armazenadas em uma planilha no Google. Posteriormente, os dados foram organizados para a exibição dos resultados que foram alcançados. Além disso, como o questionário possuía questões em aberto para coleta de justificativas às respostas fornecidas pelos alunos, classificamos e categorizamos essas justificativas para embasar os resultados obtidos.

4. Resultados

A análise dos fóruns da disciplina revelou que as opiniões dos alunos sobre as suas experiências de uso com o Scratch ficaram divididas. Inicialmente, os alunos elogiaram e aprovaram a utilização do ambiente, conforme as opiniões dos alunos 11, 20, 12 e 5 retratadas na Figura 2.



Figura 2. Elogios iniciais sobre o uso do Scratch.

Além disso, o Scratch despertou o interesse dos alunos, fazendo com que fossem em busca de novas informações e dicas sobre o ambiente para guiar o seu aprendizado e retirar suas dúvidas. Esse interesse é retratado pelos depoimentos dos alunos 3 e 1 apresentados na Figura 3.

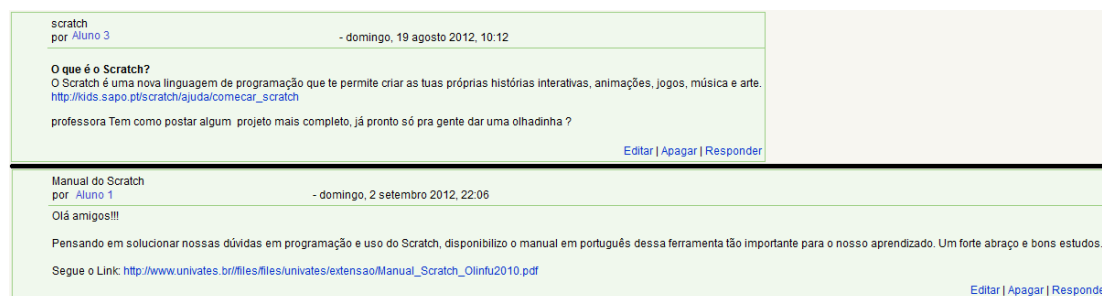


Figura 3. Interesse dos alunos por buscar novos materiais sobre o Scratch.

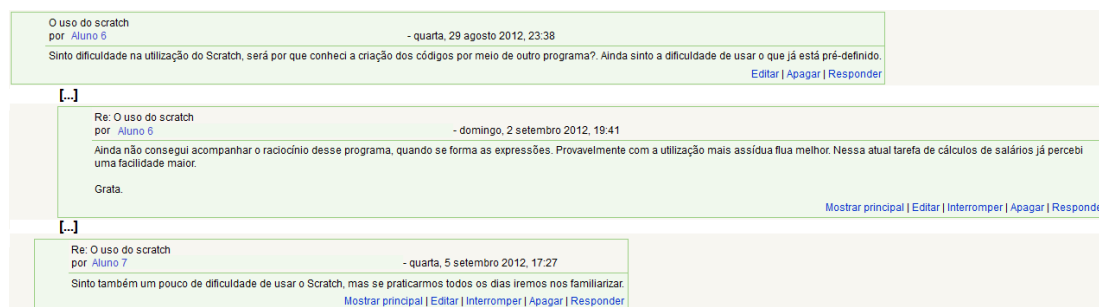


Figura 4. As dificuldades dos alunos ao utilizar o Scratch.

Por outro lado, muitas foram as críticas dos alunos sobre a utilização do Scratch. Estes relataram diferentes dificuldades no uso do ambiente como as opiniões dos alunos 6 e 7 exibidas na Figura 4. No entanto, é interessante notar que ambos os alunos concordam que, com o passar do tempo, sentiram menos dificuldades ao utilizar a ferramenta.

Ademais, os alunos também responsabilizaram o Scratch por não estarem conseguindo realizar os exercícios sugeridos durante a disciplina, como as opiniões dos alunos 22 e 10 exibidas na Figura 5.

Dificuldade para fazer a lista de menu no scratch por Aluno 22 - segunda, 27 agosto 2012, 21:32 Professores, boa noite. Estou com dificuldade para fazer a lista de menu no scratch, como devo fazer? Grato, Aluno 22 Editar Apagar Responder
duvida por Aluno 16 - segunda, 3 setembro 2012, 22:00 estou com dificuldade em usar a opeacao correta para porcentagem ja coloquei de varias formas mas o resultado nao da certo, poderia ter o sinal de porcentagem la ai ficaria bem mais facil, gostaria de uma dica para responder o exercicio da semana, obrigado. Editar Apagar Responder
Re: duvida por Aluno 10 - quarta, 5 setembro 2012, 14:44 Este scratch tá me tirando do sério... Mostrar principal Editar Interromper Apagar Responder

Figura 5. Alunos responsabilizam o Scratch por dificuldades.

Scratch ? por Aluno 12 - quinta, 23 agosto 2012, 13:56 Boa tarde! Venho através dessa tira uma duvida e fazer uma proposta. 1 - Porque não usamos o eclipse ou visualg ou outra ferramenta para o aprendizado em Logica de programação ? 2 - O Scratch é bem intuitivo mas tem sua parcela de não adequação, pois o fato de saber aonde armazenamos a variável e como isso volta pra o usuário achei ele meio programa para criança. Não sei se meus colegas de turma tem a mesma ideia. Eu acharia melhor o VisualG acho o aprendizado mais efetivo. Desde de já agradeço a atenção ☺ Editar Apagar Responder
Re: Scratch ? por Aluno 11 - sábado, 25 agosto 2012, 19:19 Achei o visualg bastante interessante, tenho alguns amigos que usam o eclipse e falaram que era muito bom, estou ansioso para aprender a usa-lo. Abraços!!! Mostrar principal Editar Interromper Apagar Responder
Re: Scratch ? por Aluno 5 - sexta, 24 agosto 2012, 20:40 Até agora estou achando bem fácil de utilizar o scratch, mas temos que pensar se os outros alunos estão se adaptando para podermos caminhar juntos, e como o professor mesmo disse mais adiante iremos usar outros programas. Mostrar principal Editar Interromper Apagar Responder
Re: Scratch ? por Aluno 9 - segunda, 27 agosto 2012, 10:17 Realmente o scratch é bem interativo. Mostrar principal Editar Interromper Apagar Responder
Re: Scratch ? por Aluno 2 - segunda, 27 agosto 2012, 14:24 Eu concordo, porque ta um pouco complicado de interagir com essa ferramenta.. Abraços Mostrar principal Editar Interromper Apagar Responder

Figura 6. Aluno sugere a utilização de outra ferramenta diferente do Scratch.

Visualg !! por Aluno 6 - quarta, 5 setembro 2012, 13:31 O visualg dá maior liberdade de criação de códigos. Acredito que com esse novo programa a assimilação de algoritmos ficará mais clara. Editar Apagar Responder
VisuALG por Aluno 21 - sábado, 8 setembro 2012, 10:26 O VisuALG é bem melhor de se trabalhar, eu particularmente Achei melhor do que o Scratch. Nele é mais pratica as funções, pois é mais facil organizar as linhas de comando que por sua vez me lembra o MS-DOS, permite erecuitar uma serie de teste durante a iscrição das linhas, mas claro exige muita pratica. Grande abraço! Editar Apagar Responder
Re: VisuALG por Aluno 9 - sábado, 8 setembro 2012, 11:18 É verdade Aluno 21. Com o Visualg podemos testar melhor os algoritmos e praticar todas as funções conforme é estudado na apostila. Mostrar principal Editar Interromper Apagar Responder
Re: VisuALG por Aluno 11 - sábado, 8 setembro 2012, 20:39 Realmente a ferramenta do visualg é bem interessante pra ver como funciona a programação em código de linha, temos que treinar bastante para poder dominar o programa. Abraços!!! Mostrar principal Editar Interromper Apagar Responder
Re: VisuALG por Aluno 5 - sexta, 14 setembro 2012, 00:48 O Visualg tem mais cara de programação, diferente do scratch que é mais parecido com animação, mais para quem está aprendendo realmente. Mostrar principal Editar Interromper Apagar Responder
Re: VisuALG por Aluno 12 - quinta, 20 setembro 2012, 23:36 eu prefiro ele pois você cria uma rotina de sempre distribuir sua logica dentro da programação e se acostuma a escrever um código Mostrar principal Editar Interromper Apagar Responder

Figura 7. Alunos aprovam a mudança para o Portugol e o Visualg.

É importante destacar a discussão iniciada pelo aluno 12 que postou o seu desagrado com o Scratch afirmando que o ambiente é “meio programa para criança” e

sugerindo a utilização de outras ferramentas. Conforme podemos perceber na Figura 6, a discussão foi continuada pelos alunos 11 e 2 que concordaram com a sugestão dada pelo aluno 12 enquanto os alunos 5 e 9 discordaram e se mostraram satisfeitos com o Scratch. Seguindo a mesma linha de opinião, na semana em que ocorreu a transição do Scratch para o Portugol e o Visualg, os alunos aprovaram a mudança de linguagem e ambiente de programação. Os alunos 6 e 21, conforme exibido na Figura 7, iniciaram discussões em que deixam claro as suas preferências por uma linguagem de programação escrita como é o caso da fornecida pelo Visualg. É interessante ressaltar a fala do aluno 5 que afirma que o Visualg “tem mais cara de programação” e do aluno 12 que reforça essa ideia, dizendo que com este ambiente, o aluno “se acostuma a escrever (sic) um código”.

A análise dos questionários, por sua vez, confirmou o que havia sido percebido anteriormente nos fóruns da disciplina, que os alunos tiveram suas opiniões divididas em relação ao uso do Scratch. Assim, quando questionados se haviam gostado ou não de utilizar o Scratch nas semanas iniciais da disciplina de Lógica de Programação, metade dos alunos afirmou que havia gostado de utilizar o ambiente e a outra metade afirmou que não havia gostado. Em seguida, quando questionados sobre a contribuição do Scratch para o aprendizado desses alunos na disciplina, a maioria concordou que a ferramenta contribuiu para o aprendizado do conteúdo abordado durante as semanas 1 a 3 na disciplina. Entretanto, a maioria teve uma opinião contrária quando se tratou dos conteúdos abordados durante as semanas 5 a 8. Da mesma maneira, a maioria dos alunos disse que o Scratch não contribuiu para o aprendizado de Portugol nas semanas 5 a 8.

Tabela 1. Respostas às questões do questionário.

	Sim	Não
Você gostou de utilizar o Scratch nas semanas iniciais da disciplina?	50%	50%
Você acha que utilizar o Scratch contribuiu para o seu aprendizado nos assuntos iniciais da disciplina?	58%	42%
Você acha que utilizar o Scratch na disciplina foi útil para o seu aprendizado nos outros assuntos da disciplina?	33%	67%
Você acha que a utilização do Scratch contribuiu para o seu aprendizado do Portugol?	42%	58%
Você acha que o Scratch deve continuar sendo utilizado nas próximas edições da disciplina?	50%	50%

Por fim, os alunos ficaram mais uma vez divididos quando perguntados se achavam que o Scratch deveria continuar sendo usado nas próximas edições da disciplina de Lógica de programação. A esta pergunta, metade dos alunos respondeu que o Scratch deve ser utilizado nas próximas edições da disciplina e outra metade respondeu que as próximas edições da disciplina não devem contar com a utilização do ambiente. Um resumo das respostas fornecidas às questões é apresentado na Tabela 1.

Entre as justificativas mais citadas pelos alunos que gostaram de utilizar o Scratch e acharam que ele contribuiu de alguma maneira para o seu aprendizado estavam a facilidade de entendimento e utilização do ambiente para a construção de programas pelos alunos iniciantes na disciplina. Ademais, a exibição do passo a passo de execução dos programas criados também auxiliava os iniciantes na disciplina na visualização da lógica do programa. Por outro lado, aqueles alunos que afirmaram não

ter gostado de utilizar o Scratch justificaram suas escolhas dizendo que o tempo empregado na sua utilização tinha sido desnecessário, uma vez que o ato de programar é feito através de uma linguagem de programação escrita. Assim, eles preferiam ter iniciado a disciplina com o Portugol e o Visualg que possui uma linguagem de programação mais próxima da que eles irão utilizar no futuro, ainda dentro do curso ou em suas vidas profissionais. O início da disciplina diretamente com o Portugol também teria evitado a transição do Scratch para o Portugol citado por eles como um ponto complicado da disciplina. Por fim, eles também afirmaram que o Scratch é um ambiente voltado principalmente para crianças ou alunos muito iniciantes em programação.

5. Conclusões e Trabalhos futuros

Neste trabalho, apresentamos um estudo que teve por objetivo avaliar o potencial do Scratch no processo de ensino-aprendizagem em programação para alunos iniciantes na EaD. Tal estudo foi realizado em uma disciplina introdutória de programação pertence à matriz curricular de um curso técnico profissionalizante em informática oferecido na EaD e o potencial da ferramenta foi avaliado através da coleta das opiniões dos alunos matriculados nesta disciplina. Para a coleta das opiniões dos estudantes utilizamos duas técnicas de pesquisa, a análise documental dos fóruns de discussão e um questionário que foi respondido pelos alunos ao final da execução da disciplina.

Os resultados encontrados da análise documental sugerem uma divisão na opinião dos alunos. No início da disciplina, alguns alunos elogiaram e aprovaram a utilização do Scratch. Além disso, o ambiente também despertou o interesse dos alunos fazendo com que eles buscassem por novas informações e dicas para guiar seu aprendizado e retirar suas dúvidas. Contrariamente, outros alunos pareceram não ter gostado de utilizar o Scratch. Estes relataram diferentes dificuldades no uso do ambiente, responsabilizaram-no por não conseguirem realizar os exercícios sugeridos na disciplina, sugeriram a utilização de outras ferramentas onde a linguagem de programação fosse escrita e, por fim, aprovaram a mudança do Scratch para o Portugol.

A análise dos questionários, por sua vez, confirmou a divisão de opiniões vivenciada pelos alunos em relação à utilização do Scratch. Assim, metade dos alunos afirmou haver gostado de utilizar o ambiente e a outra metade afirmou não haver gostado. Além disso, a maioria concordou que a ferramenta contribuiu para o aprendizado dos conteúdos abordados nas semanas 1 a 3. No entanto, a maioria concordou que o Scratch não contribuiu para o aprendizado do conteúdo abordado nas semanas 4 a 8 nem para o aprendizado do Portugol. Por fim, metade dos alunos concorda que o Scratch deveria ser utilizado nas próximas edições da disciplina e a outra metade discorda.

As justificativas para esses resultados foram várias. Aqueles que avaliaram o Scratch positivamente citaram a facilidade de entendimento e utilização do ambiente por iniciantes na disciplina e a exibição do passo a passo de execução dos programas como vantagens de sua utilização. Por outro lado, aqueles que avaliaram o Scratch negativamente afirmaram que a sua utilização havia sido desnecessária e que preferiam que a disciplina tivesse iniciado com o Portugol por ser mais semelhante a uma linguagem de programação escrita. Esta preferência também teria evitado a transição do Scratch para o Portugol, considerada de difícil compreensão por eles. Finalmente, estes

alunos afirmaram que o Scratch é voltado principalmente para crianças ou alunos iniciantes em programação.

Como trabalhos futuros, pretendemos realizar novas investigações sobre a utilização do Scratch nas semanas iniciais de uma disciplina introdutória de programação e, em seguida à sua utilização, prosseguir com uma linguagem de programação escrita. Ademais, também pretendemos avaliar a utilização do Scratch do ponto de vista da motivação e da participação dos alunos em sala de aula.

Referências

- Aureliano, V. C. O.; Tedesco, P. C. A. R. (2012) Avaliando o uso do Scratch como abordagem alternativa para o processo de ensino-aprendizagem de programação. In: 20º Workshop sobre Educação em Computação, Curitiba, Brasil.
- Bergin, S. and Reilly, R. (2005) Programming: factors that influence success. ACM SIGCSE Bulletin, New York, v. 37, n. 1, p. 411-415.
- Byrne, P.; Lyons, G. (2001) The effect of student attributes on success in programming. ACM SIGCSE Bulletin, vol. 33, n. 3, p. 49-52.
- Caspersen, M. E.; Kölling, M. (2009) STREAM: A First Programming Process. ACM Transactions on Computing Education, vol. 9, n. 1, article 4, p. 4:1-4:29.
- Fincher, S.; Cooper, S.; Kölling, M.; Maloney, J. (2010) Comparing alice, greenfoot & scratch. Proceedings do 41st SIGCSE'10, Milwaukee, Wisconsin, USA. p. 192-193.
- Hagan, D.; Markham, S. (2000) Does it help to have some programming experience before beginning a computing degree program? Proceedings do 5th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education, 2000, Helsinki. New York: ACM, p. 25-28.
- Lahtinen, E.; Ala-Mutka, K.; Järvinen, H.-M. (2005) A Study of the Difficulties of Novice Programmers. Proceedings do 10th ITiCSE, Monte de Caparica, Portugal, p. 14-18.
- Malan, D. J.; Leitner, H. H. (2007) Scratch for budding computer scientists. Proceedings do 38th SIGCSE'07, Kentucky, USA, p. 223-227.
- Maloney, J.; Peppler, K.; Kafai, B. Y.; Resnick, M.; Rusk, N. (2008) Programming by choice: Urban Youth Learning Programming with Scratch. Proceedings do 39th SIGCSE'08, Portland, USA, p. 367-371.
- Maloney, J.; Resnick, M.; Rusk, N.; Silverman, B.; Eastmond, E. (2010) The scratch programming language and environment. ACM Transactions on Computing Education, vol. 10, n. 4, article 16, 15 pages.
- Milne, I.; Rowe, G. (2002) Difficulties in Learning and teaching Programming - Views of Students and Tutors. Education and Information Technologies, vol. 7, n. 1, p. 55-66.
- Resnick, M.; Maloney, J.; Monroy-Hernández, A.; Rusk, N.; Eastmond, E.; Brennan, K.; Millner, A.; Rosenbaum, E.; Silver, J.; Silverman, B.; Kafai, Y. (2009) Scratch: programming for all. Communications of the ACM, vol. 52, n. 11, p. 60-67.
- Wilson, B. C.; Shrock, S. (2001) Contributing to success in an introductory computer science course: a study of twelve factors. ACM SIGCSE Bulletin, vol. 33, n. 1, p. 184-188.

Self-explaining from Video Recordings as a Methodology for Learning Programming

Viviane C. O. Aureliano
 IFPE - Campus Belo Jardim
 UFPE - Center of Informatics
 Av. Jornalista Aníbal Fernandes,
 Recife, PE, 50740-560, Brazil
 vcoa@cin.ufpe.br

ABSTRACT

The literature has shown that novice learners have several difficulties in acquiring programming skills. As a way to overcome these difficulties, novice learners should be guided in the process of studying the programming process. One evidence-based practice for improving student learning by guiding them while they are studying from instructional material is through the use of self-explanations. Videos have been proposed as an ideal instructional material for presenting the dynamics of programming to novices. In this context, my research proposes that novices should be guided in the process of studying videos by means of self-explanation. In order to evaluate the benefits of using this methodology, two studies will be realised as part of the research.

Categories and Subject Descriptors

K.3.2 [Computers and Education]: Computer and Information Science Education – *computer science education*.

Keywords

Learning programming; novices; video recordings; self-explanations.

1. RESEARCH SITUATION

I am enrolled in an academic PhD programme in Computer Science at the Center of Informatics of the Federal University of Pernambuco, Brazil. I am also a teacher at the Federal Institute of Pernambuco, Brazil, where I mainly teach programming language subjects to technical high school students. My main supervisor is Dr. Patrícia Tedesco.

The PhD programme in which I participate must be completed in four years and I am at the beginning of the fourth year. I defended my research proposal in November 2013. Since December 2013, I am working as a visiting PhD student with Prof. Michael Caspersen at Centre for Science Education at Aarhus University, Denmark.

2. CONTEXT & MOTIVATION

Years of research have shown that novice learners experience

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the Owner/Author.

Copyright is held by the owner/author(s).

ICER '14, Aug 11-13 2014, Glasgow, United Kingdom
 ACM 978-1-4503-2755-8/14/08.
<http://dx.doi.org/10.1145/2632320.2632321>

several difficulties in acquiring programming skills. Novices struggle when they have to apply the knowledge learned in new situations of use [1]. However, the major difficulty experienced by novices is how to combine and properly use basic structures to build a program [1, 5].

As a way to overcome these difficulties, novice learners should be guided while they are learning the programming process. One evidence-based practice for improving student learning by guiding them while they are studying from some instructional material is through the use of self-explanations [2, 3]. In particular, this practice is especially important for novice learners in programming and contributes to their learning while they are studying alone [3].

Video recordings have been proposed as an ideal instructional material for presenting the dynamic process of programming for novice learners [4]. In the presented context, my PhD research proposes a methodology to promote and guide novices' self-explanations while they are learning programming languages from video recordings. Through the use of this methodology I would like to teach technical high school students how to study properly in a programming course and encourage them to become deep learners.

3. BACKGROUND & RELATED WORK

Currently, numerous video recordings for teaching programming are available over the internet. Instead of textbooks that are static, video recordings are an important instructional material to expose the programming process since they are essentially a dynamic medium [4].

Stepwise Improvement framework describes programming as an incremental process [5]. It provides guidance regarding the structure of instructional material and, consequently, the manner novices learn programming by extending, refining and restructuring small pieces of code systematically and incrementally during the course.

In this sense, Stepwise Improvement was employed in the structure of the videos of the Joy of code channel on Youtube [6] and also in a set of worked examples in videos [5]. However, the framework does not provide any guidance regarding the way that students may study and comprehend this instructional material. One way of providing this kind of guidance is through self-explanations.

Self-explanation is a type of dialog that learners have with themselves while they are learning from different instructional materials [2, 3]. The process of self-explaining benefits the building of new knowledge through the refinement of the given information [8].

4. STATEMENT OF THESIS

I agree with Bennedsen and Caspersen (2008) that video recordings are an important medium for exposing the programming process. In particular, I believe that videos are an important resource when students are studying programming alone and outside the classroom. Also, I believe that the process of studying programming can be guided by prompting self-explanations from the students while they are studying these video recordings.

In this context, the main objective of this PhD dissertation is to propose a methodology to promote and guide novices' self-explanations while they are learning programming languages from video recordings which are structured according the ideas of Stepwise Improvement framework. In order to do that, the main research question that guides this PhD is *how could novice learners in programming benefit from using self-explanations while studying videos that expose the programming process?* In this sense, novice learners would benefit if they achieve better results, obtain higher grades, code faster or make fewer errors.

5. RESEARCH GOALS & METHODS

Aiming to answer the question which conducts this research, I am realizing two different studies. Both studies are based on the previous works by Bielaczyc and colleagues (1995) and Chi and colleagues (1994), but instead of studying plain texts, learners are using video recordings as instructional material.

The first study has the following objectives: (i) check whether the proposed self-explanations prompts are appropriate for the video recordings; (ii) analyse when to use the self-explanations prompts and the videos together; (iii) verify if there is any benefit of using self-explanations while studying video recordings (if so, which are these benefits); and (iv) refine the self-explanations prompts and the manner that they should be used. The participants in the study will be divided in three groups: (i) the instructional group in which the self-explanations are prompted while watching the video; (ii) the instructional group in which the self-explanations are prompted only when the video finishes; and (iii) the control group.

This first study consists of three phases. The first phase aims to provide the participants with a common prior knowledge of the programming language chosen and familiarize them with the think aloud process. The second phase aims to train the two instructional groups in how to use the self-explanation prompts. The third phase aims to collect data from all participants after the second phase. The students will be assessed by means of the explanations provided by them while studying the video recordings. Together with these explanations, students are asked to solve similar problems in order to assess their process of transfer of knowledge.

Aiming to answer the question which conducts this research, I believe that is an effective approach to vary the kind of video recordings adopted in the studies. For this reason, the second study will be similar in structure to the first one regarding the objectives, group participants, sequence of phases, programming language and development tool. However, it will vary regarding the following characteristics: (i) a refined set of self-explanations prompts and practice problems; and (ii) worked examples in videos as instructional materials, enriched with the self-explanations prompts.

6. DISSERTATION STATUS

As I mentioned before, I am a PhD student in Brazil, but I am working as visiting PhD student in Denmark. I expect that the first study occurs next August in Denmark. Currently, I am preparing the instructional materials that will be used in the first study. Also, I am planning each phase of the study in detail.

After the preparation for the first study, I will start to organize the instructional materials for the second study which I expect to occur next October in Brazil. Then, from the ICER Doctoral Consortium, I expect to discuss and obtain feedback mainly about the two studies that I am planning.

7. EXPECTED CONTRIBUTIONS

The expected contributions of this PhD research are: (i) a methodology for studying programming from video recordings in a deeper manner by self-explaining them; (ii) a study to analyse how novice learners in programming would benefit from using self-explanations while studying video recordings; and (iii) a suite of worked examples structured according the Stepwise Improvement framework, enriched with self-explanations prompts.

8. ACKNOWLEDGMENTS

I would like to thank CAPES Foundation for the scholarship provided under report n° 1127-91-3.

9. REFERENCES

- [1] Robins, A.; Rountree, J.; Rountree, N. (2003) Learning and Teaching Programming: A Review and Discussion, Computer Science Education, v. 13, n. 2, p. 137-172.
- [2] Chi, M. T. H.; Bassok, M.; Lewis, M.; Reimann, M., & Glaser, R. (1989). Self-explanations: How students study and use examples in learning to solve problems. Cognitive Science, 13, 145-182.
- [3] Bielaczyc, K.; Pirolli, P.; Brown, A. L. (1995). Training in self-explanation and self-regulation strategies: Investigating the effects of knowledge acquisition activities on problem solving. Cognition and Instruction, 13, 221-253.
- [4] Bennedsen, J.; Carpersen, M. E. (2008). Exposing the Programming Process. In Reflection on the Theory of Programming: Methods and Implementation, J. Bennedsen, M. E. Carpersen, M. Kölling (Eds.) Springer-Verlag, 6-16.
- [5] Caspersen, M. E. (2007) Educating novices in the skills of programming. PhD dissertation PD-07-4, Department of Computer Science, University of Aarhus.
- [6] Kölling, M. (2012). *The Joy of code* [Video recordings]. Retrieved from <http://bit.ly/1jbTbHq>
- [7] Chi, M. T. H.; deLeeuw, N.; Chiu, M.; LaVancher, C. (1994). Eliciting self-explanations improves understanding. Cognitive Science, 18, 439-477.
- [8] Chiu, J. L.; Chi, M. T. H. (2014). Supporting Self-Explanation in the Classroom. In V.A. Benassi, C.E. Overson, C.M. Hakala (Eds.). Applying science of learning in education: Infusing psychological science into the curriculum. Retrieved from: <http://bit.ly/KbYLtG>

Self-explaining from videos as a methodology for learning programming

Viviane Cristina Oliveira Aureliano

*Center of Informatics - Federal University of Pernambuco
Federal Institute of Pernambuco - Campus Belo Jardim
vcoa@cin.ufpe.br*

Keywords: POP-I.A. Learning to program; POP-I.B. Choice of methodology; POP-II.A. Novices.

Abstract

The literature has shown that novice learners have several difficulties in acquiring programming skills. As a way to overcome these difficulties, novice learners should be guided in the process of studying the programming process. One evidence-based practice for improving student learning by guiding them while they are studying from some instructional material is through the use of self-explanations. Video recordings have been proposed as an ideal instructional material for presenting the dynamics of programming to novices. In this context, my research proposes that novices should be guided in the process of studying video recordings by means of self-explanation. In order to evaluate the benefits of using the proposed methodology, two studies will be realized as part of this research.

1. Introduction

Years of research have shown that novice learners experience several difficulties in acquiring programming skills. They have fragile knowledge in learning programming (Perkins and Martin 1986). One type of fragile knowledge is the inert knowledge, which is the knowledge novice learners have, but fail to retrieve when it is necessary (CTGV 1992). Novices use inappropriate study methodologies and do not take constant effort to develop their programming skills (Gomes and Mendes 2007). However, the major difficulty experienced by novices is how to combine and properly use the basic structures to build a program (Caspersen & Kölling 2009).

Stepwise Improvement is a conceptual framework that describes programming as an incremental process that encompasses three different activities, extension, refinement and restructuring (Caspersen 2007; Caspersen & Kölling 2009). In the area of programming education, the authors advocate the application of the framework in order to guide the process of teaching and learning programming. In this case, it provides guidance regarding the structure of the instructional materials in a way that novices learn programming by extending, refining and restructuring small pieces of code systematically and incrementally during the course.

Bennedsen and Caspersen (2008) have proposed the use of videos as an ideal medium for revealing the programming process for learners. Instead of textbooks that are static, the authors advocate that videos are an important instrument to expose the programming process since they are essentially a dynamic medium. Caspersen (2007) extended this idea by proposing in his doctoral dissertation that worked examples in videos could be used as an instructional material in an introductory programming course. The application of worked examples and practice exercises that are similar in structure but diverse in situations of use has been demonstrated as a good strategy to promote the transfer of learning (Clark, Nguyen and Sweller 2006).

Self-explanation is a type of dialog that learners have with themselves while they are learning from different instructional materials (Bielaczyc, Pirolli & Brown 1995; Chi, Bassok, Lewis, Reimann & Glaser 1989; Clark et al. 2006). It is an effective learning activity that leads learners to move further in their knowledge about a subject (Fonseca & Chi 2011) by building new knowledge through the refinement of the given information (Chiu & Chi 2014).

As a way to overcome the aforementioned difficulties, novice learners should be guided in the process of studying the programming process. This guidance will be provided by means of the use of self-

explanation practice. Since video recordings have been proposed as an ideal instructional material for exposing the programming process to novices, the self-explanation practice will be applied with this kind of instructional material. In the presented context, my PhD research proposes that novice learners should be guided in the process of studying video recordings by means of self-explanation. This methodology of using videos in conjunction with self-explanations will adopt the programming pathway defined by the Stepwise Improvement framework.

2. Theoretical framework

2.1 Cognitive Load Theory

Cognitive Load Theory (CLT) provides a set of learning principles and related instructional guidelines that are proven to result in instructional environments that are more effective regarding students learning (Clark et al. 2006). In order to provide such a set of principles and guidelines, CLT is closely aligned with the human cognitive structure and the way that people learn. In their book, the authors of the theory explain that the human cognitive structure has schemas or patterns as units of information. In order to manipulate and store this information, the human cognitive structure has also two memory systems, the working memory and the long-term memory. The working memory is the active partner of our cognitive structure. It is responsible for processing new information while we are learning and also for organizing new schemas from the ones that we already have. The working memory works closely together with the long-term memory. While the working memory has a very limited capacity, long-term memory has a huge capacity for storing information and, once we learn, the resultant schemas are stored on it. Because of this, the long-term memory is our main knowledge repository.

According to Clark et al (2006), while students are learning they have to deal with three different types of cognitive load: intrinsic, germane and extraneous. Intrinsic cognitive load results from the complexity of the instructional content. Germane cognitive load is the productive mental effort required for building new schemas and relevant to the learning process. Extraneous cognitive load is defined as the unproductive (or irrelevant) mental effort that does not lead to learning. In order to have efficient learning environments, teachers have to manage intrinsic cognitive load, increase germane cognitive load and reduce extraneous cognitive load. Doing that, they can free the working memory from irrelevant mental effort. Consequently, it is possible to apply that free space for the work demanded to integrate new information into schemas in the long-term memory.

2.2 Stepwise Improvement

Stepwise Improvement is a conceptual framework that describes programming as a systematic and incremental process that encompasses three different activities, extension, refinement and restructuring (Caspersen 2007; Kölling & Caspersen 2009). Extension activity occurs when the specification is expanded in order to cover more use cases. Refinement activity occurs when abstract code is modified towards an executable code in order to implement the current specification. Restructure activity occurs when an improvement of nonfunctional aspects of a solution is made, but this modification does not involve a change in its apparent behavior. These activities are organized in a three-dimensional space that is explored by programmers while they are building programs.

Caspersen and Kölling (2009) advocate the application of Stepwise Improvement by using guided tours rather than random walks. In order to conduct learners' steps in the programming space offered by the framework, the authors state that teachers should be concerned with the right amount of guidance and scaffolding (Collins, Brown & Newman 1989) given during the instruction. It guarantees to balance important aspects of programming education: train learners' programming skills in a proper manner and, at the same time, maintain the cognitive load while students are learning under control.

In that sense, Stepwise Improvement brings a significant contribution to the programming education field by providing guidance in the three activities that it encompasses. Regarding the extension and restructure activities, it provides guidance in the way that the instructional material is organized. Therefore, programming textbooks, assignments, lectures and examples, among other instructional

materials, can be structured using the set of steps defined in the framework. For instance, Stepwise Improvement has been employed in the organization of the Greenfoot's textbook (Kölling 2010) and the problem solving videos of the Joy of code channel on Youtube (Kölling 2012). In addition, regarding the refinement activity, Caspersen (2007) defined in his doctoral dissertation an object-oriented programming process for teaching novices.

2.3 Worked examples

One effective manner of maximizing the germane cognitive load is by using worked examples. Worked examples are sequences of steps defined to demonstrate one particular task performance or to solve a specific problem (Clark et al. 2006). They are especially important for novice learners, once they represent a step-by-step guide necessary to execute a procedure or reach a goal. For this reason, worked examples help novice learners to build new schemas while they are acquiring a new knowledge or skill.

In that sense, worked examples are an effectively evidence-based instructional material. In their book, Clark et al. (2006) demonstrate some advantages of using worked examples. For instance, learners using pairs of worked examples-problems could spend less time studying and still have equivalent learning results rather than learners using all problems as practice exercises. In addition, the authors showed that the application of worked examples and practice exercises that are similar in structure but diverse in situations of use is a good strategy to promote the transfer of learning.

Worked examples have been used in different subjects, such as Physics (Chi et al. 1989), Chemistry (Crippen & Earl 2007) and Programming (Abdul-Rahman & du Boulay 2014; Caspersen 2007), among others. In addition, they can be shown in various formats and modalities, including text (Chi et al. 1989), web (Crippen & Earl 2007), learning environment (Abdul-Rahman & du Boulay 2014) or video recordings (Caspersen 2007).

2.4 Self-explanations

Self-explanation is a kind of dialog that learners have with themselves while they are learning from different instructional materials (Bielaczyc et al. 1995; Chi et al. 1989; Chi et al. 1994; Clark et al. 2006; Crippen & Earl 2007). Fonseca and Chi (2011) presented a literature review about the self-explanation effect (Chi et al. 1989) and its effectiveness as a learning activity that leads people to produce overt output that goes further the presented information. In this sense, the process of self-explaining benefits the building of new knowledge through the refinement of the given information, associating this new information with prior knowledge, reasoning about it and connecting it with other different pieces of information (Chi & Chiu 2014). Self-explanations also promote the exploration of worked examples by learners in a deeper way and maximize the germane cognitive load (Clark et al. 2006). Because of their evidence-based benefits, self-explanations are considered one of the seven practices recommended for improving student learning (Pashler et al. 2007).

The self-explanation effect was demonstrated in different subjects, such as Biology (Chi et al. 1994), Physics (Chi et al. 1989) and Programming (Bielaczyc et al. 1995), among others. In addition, self-explanations can be elicited from various formats and modalities of instructional material, including text (Chi et al. 1994; Chi et al. 1989; Bielaczyc et al. 1995) and worked examples (Clark et al. 2006). The activity of self-explanation may be spontaneously generated (Chi et al. 1989). Also, it can be prompted and trained (Bielaczyc et al. 1995; Crippen & Earl 2007).

3. Statement of thesis

As mentioned before, Stepwise Improvement is a conceptual framework that describes programming as a systematic and incremental process that takes place in a three dimensional space of extension, restructure and refinement activities. In order to move in this space, Caspersen and Kölling (2009) advocate guided tours instead of random walks. Guidance has been provided regarding the manner that the instructional material for programming courses may be presented to students and also the programming activity itself. However, the framework does not provide any guidance regarding the way that students may study and comprehend this instructional material.

In addition, the concept of self-explanation was cited previously as well as its benefits as a learning activity. Self-explanations promotes the building of new knowledge through the refinement of the given information, associating this new information with prior knowledge, reasoning about it and connecting it with other different pieces of information (Chi & Chiu 2014). Explaining to oneself is an effective practice recommended for improving students learning (Pashler et al 2007). In particular, this practice is especially important for novice learners in programming and contributes to their learning while they are studying alone (Bielaczyc et al. 1995).

Nowadays, the use of videos for teaching programming is becoming more and more popular. For instance, numerous videos are available in Youtube channels or in organizations as Khan Academy (2014) and Code.org (2014). However, the use of videos for teaching programming is not a novelty. Some years ago, Bennedsen and Caspersen (2008) showed the benefits of using videos as a tool for revealing the programming process for learners. Instead of textbooks that are static, the authors advocate that videos are an important instrument to expose the programming process since they are essentially a dynamic medium. In addition, using videos students can manipulate these videos - play, forward or rewind - in their own pace and how many times they need.

Stepwise improvement has been employed in the incremental structure of different video collections for learning programming. For example, the structure of problem solving videos of the Joy of code channel on Youtube (Kölling 2012) followed this framework. Also, Caspersen (2007) adopted this idea by proposing in his doctoral dissertation that worked examples in videos could be used as an instructional material in an introductory object-oriented programming course.

I agree with Bennedsen and Caspersen (2008) that videos are an important medium for exposing the programming process. In particular, because of their aforementioned advantages, I believe that videos are an important resource when students are studying programming outside the classroom. Also, I believe that the process of studying programming can be guided by prompting self-explanations from the students while they are studying from these videos. In this context, the main objective of this PhD dissertation is to provide guidance regarding the way students are studying and comprehending videos for learning programming. This guidance will be provided by means of prompting self-explanation while novice learners are studying the programming pathway defined in the Stepwise Improvement framework.

In order to do that, the main research question that guides this PhD is *how novice learners in programming could benefit from using self-explanations while studying video recordings that expose the programming process?* In this sense, novice learners would benefit if they achieve better results, obtain higher grades, code faster or make fewer errors.

4. Research goals

Aiming to answer the question which conducts this research, I am realizing two different studies. Both studies are based on the previous researches developed in the works of Bielaczyc et al. (1995) and Chi et al. (1994), but instead of studying plain texts, learners are using videos as an instructional material.

In the first study, I am using Java as programming language, Greenfoot (2011) as development tool and the problem solving videos of the Joy of code channel on Youtube (Kölling 2012). It has the following objectives: (i) check whether the proposed self-explanations prompts are appropriate for the videos; (ii) analyse when to use the self-explanations prompts and the videos together; (iii) verify if there is any benefit of using self-explanations while studying videos that expose the programming process (if so, which are these benefits); and (iv) refine the self-explanations prompts and the manner that they should be used. In order to do that, the participants in the study should be divided in three different groups: (i) the instructional group in which the self-explanations are asked while showing the video; (ii) the instructional group in which the self-explanations are asked only when the video ends; and (iii) the control group.

This first study consists of three phases: (i) pre-intervention phase; (ii) instructional intervention phase; and (iii) post-intervention phase. The objective of the pre-intervention phase is to provide the participants with a common prior knowledge of the programming language Java and familiarize them

with the self-explanation prompts and the think aloud process. The objective of the instructional intervention phase is training the two instructional groups in how to use the self-explanation prompts. In that sense, for the case of the first instructional group, the video is divided in goals and for each goal the video is paused and the student is prompted with the self-explanation questions. On the other hand, for the second instructional group, the self-explanations questions will be prompted only when the video finishes. At the end of this study, the post-intervention phase aims at collecting data from all participants after the instructional intervention phase.

The students are assessed in two different ways. First, after studying each programming video, students are asked to solve similar problems in order to assess their process of transfer of knowledge. Second, they are assessed by means of the explanations provided by them while studying the programming videos in the instructional and post-intervention phases. In addition, I am getting feedback from the students about their usage of the videos and self-explanation prompts for learning programming.

As mentioned previously, worked examples are an effectively evidence-based instructional material that accelerates the novice's process of learning when adopted together with practice exercises. The concept of worked examples is closely connected with the self-explanations' concept, since self-explanations consist in a manner of promoting deeper learning of worked examples. Aiming to answer the question which conducts this research, I believe that an effective approach is to vary the kind of video recordings exposing the programming process adopted on it. For this reason, the second study will be extremely similar in structure to the first one regarding the objectives, types of group participants, sequence of phases, programming language and development tool. However, it will vary regarding the following characteristics: (i) a larger number of participants; (ii) a refined set of self-explanations prompts and practice problems; and (iii) worked examples in videos as instructional materials instead of using the problem solving videos of the Joy of code channel on Youtube. To do that, I am producing these worked examples in videos in a similar structure proposed in the Joy of code's videos.

5. Dissertation status and expected contributions

I am enrolled in an academic PhD programme in Computer Science at the Informatics Centre of the Federal University of Pernambuco, Brazil. I am also a teacher at the Federal Institute of Pernambuco, Brazil, where I mainly teach programming language subjects. My supervisor is Professor Patrícia Tedesco. The PhD programme in which I participate must be completed in four years and I am at the beginning of the fourth year. I have already completed all six courses that I needed. Also, I defended my research proposal in November 2013. Currently, I am working as a visiting PhD student with Professor Michael Caspersen at Centre for Science Education at Aarhus University, Denmark.

The expected contributions of this PhD research are: (i) a methodology for studying programming from videos that expose the programming process in a deeper manner by self-explaining them; (ii) a study to analyse how novice learners in programming would benefit from using self-explanations while studying videos that expose the programming process; and (iii) a suite of worked examples using Java as programming language and Greenfoot as development tool, enriched with self-explanations prompts. All these contributions use the Stepwise Improvement as the principal conceptual framework.

6. Acknowledgments

I would like to thank CAPES Foundation for the scholarship provided under report nº 1127-91-3.

7. References

- Abdul-Rahman, S. S., & du Boulay, B. (2014). Learning programming via worked-examples: Relation of learning styles to cognitive load. *Computers in Human Behavior*, 30, 286-298. doi:10.1016/j.chb.2013.09.007

- Bennedsen, J., & Carpersen, M. E. (2008). Exposing the Programming Process. In J. Bennedsen, M. E. Carpersen, & M. Kölling (Eds.), *Reflection on the Theory of Programming: Methods and Implementation* (pp. 6-16). Berlin, Germany: Springer-Verlag.
- Bielaczyc, K., Pirolli, P., & Brown, A. L. (1995). Training in self-explanation and self-regulation strategies: Investigating the effects of knowledge acquisition activities on problem solving. *Cognition and Instruction*, 13, 221-253. doi:10.1207/s1532690xc1302_3
- Caspersen, M. E. (2007). Educating novices in the skills of programming. PhD dissertation PD-07-4, Department of Computer Science, University of Aarhus.
- Caspersen, M. E., & Kölling, M. (2009). STREAM: A First Programming Process, *ACM Transactions on Computing Education*, 9 (1), 4:1-29. doi:10.1145/1513593.1513597
- Chiu, J. L., & Chi, M. T. H. (2014). Supporting Self-Explanation in the Classroom. In V.A. Benassi, C.E. Overson, & C.M. Hakala (Eds.), *Applying science of learning in education: Infusing psychological science into the curriculum*. Retrieved from: <http://bit.ly/KbYLtG>
- Chi, M. T. H., Bassok, M., Lewis, M., Reimann, M., & Glaser, R. (1989). Self-explanations: How students study and use examples in learning to solve problems. *Cognitive Science*, 13, 145-182. doi:10.1207/s15516709cog1302_1
- Chi, M. T. H., deLeeuw, N., Chiu, M., & LaVancher, C. (1994). Eliciting self-explanations improves understanding. *Cognitive Science*, 18, 439-477. doi:10.1016/0364-0213(94)90016-7
- Clark, R., Nguyen, F., & Sweller, J. (2006). Efficiency in learning: evidence-based guidelines to manage cognitive load. San Francisco, CA: John Wiley & Sons, Inc.
- Cognition and Technology Group at Vanderbilt - CTGV (1992). The Jasper series as an example of anchored instruction: Theory, program description, and assessment data. *Educational Psychologist*, 27, 291-315. doi:10.1207/s15326985ep2703_3
- Collins, A., Brown, J. S., & Newman, S. E. (1987). Cognitive apprenticeship: Teaching the crafts of reading, writing, and mathematics. Technical Report No. 403, University of Illinois.
- Code.org (2014) Retrieved from: <http://code.org/learn>
- Crippen, K. J., & Earl, B. L. (2007). The impact of Web-based worked examples and self-explanation on performance, problem solving, and self-efficacy. *Computers & Education*, 49 (3), 809-821. doi:10.1016/j.compedu.2005.11.018
- Gomes, A., & Mendes, A. J. (2007) Learning to Program - Difficulties and Solutions. In *Proceedings of the International Conference on Engineering Education, Coimbra, Portugal*. Retrieved from: <http://bit.ly/1ldTqkE>.
- Greenfoot (Version 2.3.0) [Computer software] (2011) Retrieved from <http://www.greenfoot.org/door>
- Fonseca, B. & Chi, M.T.H. (2011). Instruction based on self-explanation. In R. E. Mayer, & P. A. Alexander (Eds.), *The Handbook of Research on Learning and Instruction* (pp. 296-321). NY: Routledge Taylor and Frances Group.
- Khan Academy (2014) Retrieved from <https://www.khanacademy.org/>
- Kölling, M. (2010) Introduction to Programming with Greenfoot: Object-Oriented Programming in Java with Games and Simulations. New Jersey, USA: Pearson Higher Education.
- Kölling, M. (2012). *The Joy of code* [Video recordings]. Retrieved from <http://bit.ly/1jbTbHq>
- Pashler, H., Bain, P., Bottge, B., Graesser, A., Koedinger, K., McDaniel, M., and Metcalfe, J. (2007). Organizing Instruction and Study to Improve Student Learning (NCER 2007-2004). Retrieved from: <http://ncer.ed.gov>
- Perkins, D. N.; Martin, F. (1986) Fragile knowledge and neglected strategies in novice programmers. In E. Soloway & S. Iyengar (Eds.), *Empirical studies of programmers, First Workshop* (pp. 213-229). Norwood, NJ: Ablex.



Digital and Collaborative Technologies for Smarter Distance Education

Eduardo A. Oliveira^{1,3}, Viviane C. O. Aureliano^{1,2},
Rozelma S. de França¹ and Patricia C. A. R. Tedesco¹

¹ Centro de Informática – Universidade Federal de Pernambuco (UFPE)
50740-560 – Recife – PE – Brasil

² Instituto Federal de Pernambuco – Campus Belo Jardim (IFPE)
55150-000 – Belo Jardim – PE – Brasil

³ Department of Computing and Information Systems, The University of Melbourne.
Parkville 3010 VIC, Australia

Abstract— With the integration of web 2.0 and social technologies into educational practices, researchers have been faced with the challenge of providing better online virtual learning environments, that engage students in the learning processes. This challenge entails using the massive amounts of knowledge we generate in our digital footprints to provide personalized learning contents. In our research group, we have tried to tackle this challenge by answering two questions: (i) How to share, at the same time, the knowledge of a virtual course in a decentralized way in Social Networks, MOOC, VLE and others? (ii) How to provide pedagogical practices in virtual learning environments that allow students to build their knowledge in an autonomous and reflective way? In this article we present the work we have carried out to answer both questions.

Index Terms — Collaborative Technologies, Distance Education, Personalization, Computational Context, Student Protagonism, Pedagogical Practices

I. INTRODUCTION

Since the web 2.0 and social technologies were irrevocably integrated into Educational practices, many research efforts have been carried out in order to understand and optimize online virtual learning environments [1].

Implementing Distance education courses typically involves using software to support the educational activity. The use of such software leads to the creation of large repositories that record massive amounts of data about the students' learning process [2][3]. However, it is important to emphasize that simply creating large databases is useless to support courses management and students' monitoring. Therefore, supplying appropriate tools for extraction, analysis and interpretation of the information stored in these databases is essential [4].

Different from what happened a few years ago, virtual learners are now much more familiar with mobile devices and

Internet resources (in their various platforms and social networks – thus characterizing what is now known as digital nomadism [5]) . Thus, they are used to work in groups.

Collaboration has been widely accepted as a way to activate relevant learning mechanisms that cannot be triggered by working alone [6]. This is partly because working in groups creates the need to articulate internal thoughts to explain a point of view, or to defend a position, and it also helps individuals to learn about others' perspectives [7].

Given this paradigm change and the growth of collaborative practices, it becomes even harder to get students of virtual courses to become interested in traditional Virtual Learning Environments (VLE), where the interfaces are not appropriate to their particular needs and collaborative tools often do not allow these students to establish relations with other colleagues, unlike what happens naturally in Social Networks and in Massive Open Online Courses (MOOC) [8].

Social Networks and MOOCs have evolved and reinvented themselves much faster than traditional VLE. The evolution of these VLE to more social ones, such as those introduced by the Ivy League MOOC generators [9] that have rapidly extended throughout universities around the world [10], is the most irrefutable fact of the integration and disruption that social and web 2.0 technologies have caused in learning and educational processes, notwithstanding the potential benefits of social collaboration among peers [11], for example.

Students do not perceive the advances in VLE at the same speed with which they sense changes in other environments, such as Social Networks [12]. MOOC are said to be a new form of online training and tend to have a decentralized, network-based, non-linear structure focused on exploration and conversation rather than emphasizing instructor-provided content.

All these issues bring another implication that needs to be taken into consideration when we use VLE: students have become much more autonomous, connected and collaborative builders of their own knowledge.

Despite the growth of Distance Education and new digital learning environments around the world, several challenges are still found in the current tools:

- From a computational perspective, we can cite the lack of interoperability/integration between systems/devices, difficulties to reuse information, unintelligent/not customized search of contents;
- From the user experience perspective, we can cite, lack of personalized support, lack for content adaptation to different devices, lack of integration with tools they already use, among others that also contribute for the demotivation of students [8].

To proactively meet the needs of students, there is a need to consider the elements that are related to the context of the learner. Context can be understood, in this scenario, as what underlies the ability to define what is relevant at any given time.

The use of context allows the system to filter and disseminate useful information and adapt its services to the particular needs of the student, providing recommendations and changes in interfaces (to become more flexible and easier to use)[13].

Elements surrounding the learner that are of interest are: the location, the devices they use, their activities, the time of day, among others.

Thus, new approaches like context, learning analytics or educational data mining [14] should be used more often, empowering both teachers and learners in their roles. On one hand, teachers can be able to understand and optimize learning processes within their courses and on the other, students can have a better assistance and a more efficient and personalized learning [15].

Unfortunately many of the commonly used current educational technologies are not integrated; they have fixed hardware settings and typically work in isolation. As a consequence, context awareness and prediction capabilities are sparse, with limited adaptation based on the students' spatial and temporal dynamics.

To complicate matters even further, most e-learning environments centralize information, making themselves the only gateway to courses, when perhaps a more natural way would be to make information accessible in a distributed way – thus instructors could explore whichever tools the students are most comfortable with.

Many e-learning environments have been built in a similar manner over the years and continue dealing with all students in the same way, impersonally, without regard to their particular needs [16].

Considering that mobile devices and the Internet have a myriad of resources for communication and interaction, massive amounts of stored information and knowledge produced by users, together with endless possibilities for various combinations, either of resources, information and/or persons, we now face the challenge of allowing students to learn in distributed and context sensitive manners [8]. This is precisely what motivated the following research question in this study:

How to provide distributed and personalized learning contents with existing digital resources (mobile devices/Internet)?

From the question that motivated this research, two other questions arose:

- (i) How to share, at the same time, the knowledge of a virtual course in a decentralized way in Social Networks, MOOC, VLE and others?
- (ii) How to provide pedagogical practices in virtual learning environments that allow students to build their knowledge in an autonomous and reflective way?

The concept of a “smart education” encapsulating the use of Information and Communication Technologies (ICT) for monitoring, controlling and supporting decision making in education is now widely accepted. However, most of the data found over the ICT (e.g. social networks, websites, and mobile devices) has an “unstructured” format, residing not in structured databases but in a variety of files (documents/presentations/calendars). Some research [3] [17] [18] have been developed to exploit this large amount of data and transform them into useful information for those involved in distance education courses, such as teachers, tutors, managers and students.

In this context, the overarching aim of this article is to discuss the fundamental concepts related to the challenges presented above, contemplating both the technological and pedagogical challenges. In particular, we want to support students both when they are collaborating and when they are working on their own. This article also discusses the potential generated by using social networks, intelligent agents, context and other techniques in order to promote the interaction between students and distributed virtual learning contents. In doing so, we present examples of works developed within our group that fit the two main challenges discussed above.

This article is organized as follows: Section II presents a brief literature review, discussing works related to the challenges presented; Section III presents the i-collaboration 3.0, a framework, developed as part of our work, that tries to promote the adaptation and personalization of learning contents in distributed web environments; Section IV discusses aspects of pedagogical practices that consider students' autonomy, presenting two examples of our work in the domain of teaching programming. Finally, section V presents our conclusions and open research avenues.

II. LITERATURE REVIEW

The concept of Web 2.0 highlights the growing popularity of ‘social software’ or digital social media, where users are connected to and collaborate with each other in a variety of group interactions [37]. According to [38], the Internet has become an essential mean to many new aspects of our daily lives today. Even though it may have become a commodity,

the evolution of the Internet is taking new and unexpected turns [35].

Internet in education has also reached to a new context known as e-learning or web-based education process, in which large amounts of information about teaching-learning interaction are endlessly generated and ubiquitously available [34]. Development of both educational software as well as on-line web applications and databases or repositories capable of storing knowledge are indicators of the exponentially increased need of Internet and web learning systems [36]. Current implementations of distance education presents some impersonal characteristics that replace the personal interaction between teacher and student in a classroom, thus distinguishing it from traditional classroom education [19]. Although different from traditional education, the interaction, dialogue and collaboration between students and professors in VLE are factors that determine the nature of learning [20], just as it happens in traditional education. These factors are closely related to the quality of participation of those involved in the process of knowledge production [4].

In the context of Technology-Enhanced Learning (TEL), system designers have tried to systematically exploit the modeling potential of computers and develop systems that support learners through adaptive or intelligent operation, to support and improve the quality of interaction and collaboration in virtual learning environments [21].

Adaptive and Intelligent systems are model-based systems that, although they have different purposes in supporting learning, aim to address the new needs of the new digital users, who are much more dynamic and nomadic. Hence, new systems should support adaptive content selection and adaptive contents presentation (interface). Thus, the learning content can be directed to better meet the needs of each student.

To meet this challenge, researchers in the field of adaptive systems try to overcome the shortcomings of traditional approaches, which deal with all users in the same way (one-size-fits-all), exploring ways in which they can adapt their behavior to the goals, tasks, interests and other characteristics of interested students [22].

In educational contexts, while the definitions of “adaptive systems” differ in the literature, many of the interpretations converge along the lines of the system's ability to adjust itself to suit individual learners' characteristics and needs. Hence, an Adaptive Educational System (AES) is a system that aims at adapting some of its key functionalities (for example, content presentation and/or navigation support) to the learner needs and preferences [22]. Thus, AES operate differently for different learners, taking into account information accumulated in the individual or group learner models.

On the other had, Intelligent Tutoring System (ITS) aim to provide learner-tailored support, similarly to what a human tutor would do. To achieve this, ITS designers apply techniques from the broader field of Artificial Intelligence (AI) and implement extensive modeling of problem-solving processes in the specific application domain.

Both AES and ITS seek primarily to meet the individual needs of each student in an intelligent (autonomous) way. The main difference between AES and ITS relates to their overall goals. While AES focus on adapting content and interfaces,

ITS directly focus on supporting the learning of each student, simulating the behavior of a virtual tutor (communication). In this article, we assume that both are intelligent and autonomous systems.

During the last decade, new tools have emerged in the online learning and thus, the traditional teaching models and methods have started to change [39]. Several learning models (i.e., informal or non-formal) associated to new training methods are now present in the academic and educational sector. One of the main drivers of this transformation is the MOOC, which is an online course aimed at unlimited participation and open access via the web [39].

Despite the fact that these initiatives are very laudable, according to [43], the main problem with them is they do not guarantee effective collaboration. Just putting a group of people around a task does not imply that people will collaborate; it is necessary to encourage people to participate and collaborate. Besides that, the initiatives are still restricted to specific environments and do not provide personalized contents, which does not satisfy the aspirations of most nomadic students nowadays. To promote the adaptation and personalization of learning contents in a distributed manner in the Internet (creation of virtual learning spaces – contents available into Twitter, Facebook, Skype and GTalk, for instance), based on each student profile and needs, we present the i-collaboration 3.0 framework, detailed in section 3.

Other challenge that distance learning education brings to us is the definition of new pedagogical practices or the adaptation of old ones to match its needs. In this sense, we should be concerned about how to align the definition or adaptation of these pedagogical practices with the literature about how people learn [50]. Bransford and colleagues (2005) state that two of the key findings regarding how people learn are: (1) students should learn subjects in-depth; and (2) teaching of some subject should be integrated with the teaching of metacognitive skills.

One evidence-based practice for helping students learn subjects in-depth is through the use of self-explanations [21]. Prompting students to self-explain while they are learning was evaluated as an effective practice to foster student learning in different areas and education contexts [21, 22, 24, 51, 52].

Another way of teaching contents integrated with the teaching of metacognitive skills is through the use of self-regulation [49]. Due to challenges in VLEs, the discovery of ways to foster self-explanation and self-regulation practices in these learning environments is a new and promising path for researching.

Thus we present in section 4 two works that are being developed by our research group. The first work encourages in-depth learning of programming languages through self-explanation of video recordings [30]. The second promotes self-regulated learning of computational thinking through programing [45].

III. I-COLLABORATION

To contribute to the minimization of the challenges found in VLEs (communication difficulties, centralized access, interoperability and data integration), and thus also contribute to minimizing the various problems currently found in

distance education (motivation and isolation feeling), this work presents the *i-collaboration 3.0* framework. *i-collaboration 3.0* is a work developed within our group that fits the first main challenge discussed in the Introduction of this article: (i) How to share, at the same time, the knowledge of a virtual course in a decentralized way in Social Networks, MOOC, VLE and others?

Despite the fact that this framework has been designed and experimented for learning domains, this project proposes innovative infrastructures fulfilling the requirements related to the nature of Context Aware and Predictive Analysis and, the Internet of Things, which are among others: heterogeneity (e.g., different objects, sensors, protocols and applications), dynamicity (e.g., arrival and departure of systems and objects), analysis (e.g., contents personalization, recommendations and prediction) and evolution (e.g., support for new protocols, systems and sensors).

The framework was designed to use intelligent agents in order to be able to collect and process data from multiple sources and to infer events or patterns that suggest more complicated circumstances (Complex Event Processing - CEP).

The proposed intelligent agent based framework deals with Distributed Problem Solving, Adaptive Personal Assistants, and Social Recommender Systems research areas, and will be better detailed in the next sub-section. The proposed framework is an extension of *i-collaboration (1.0)* model [31]. Table 1 presents the evolution from *i-collaboration (v1.0)* to *i-collaboration 3.0*.

Table 1. Proposed evolution from *i-collaboration 1.0* model to *i-collaboration 3.0* system

<i>i-collaboration (1.0)</i>	<i>i-collaboration 3.0</i>
Available only in Cleverpal VLE	Can be integrated with any VLE. It has no dependencies
An independent instance of the intelligent agent for communicating with the students in each Web 2.0 tool (Twitter, MSN and Websites).	A single instance of the intelligent agent shared for communicating with the students in all Web 2.0 tools (Twitter, MSN and Websites) and social networks
Contextual information for students (knowledge that each student has: MBTI profile, content already viewed by each student, which content each student needs to study more ...) are not integrated with Web 2.0 tools that support the model.	The contextual information of the students are integrated between the various Web 2.0 tools, VLEs, social networks ...
The domain content is customized and distributed but is not integrated	The content domain of the system is customized, distributed and integrated
Model is not available to the scientific community	System will be available to the scientific community under a license

i-collaboration 3.0 tries to ensure distributed access to learning contents available in different Web 2.0 tools (e.g., Twitter, Skype and blogs, among others) and social networks (e.g., Facebook and LinkedIn, among others). The framework also integrates students' data to personalize the learning contents (students are distributed in the Internet – the same student can learn using Skype, Facebook and Twitter, at the

same time, for example), based on the particular tastes and needs of each student (identified through the student behavior in the various Web platforms).

Given the support for virtual learning spaces, students will be able to study through the Web, using platforms and environments that they are already familiar with.

As shown in Figure 1, we assume that Twitter, MSN, a blog (Blogger site), Facebook and Moodle (VLE) are integrated with *i-collaboration 3.0* framework, meaning that these environments are using *i-collaboration 3.0* (there is an user account in each of these environments connected with the framework).

In the presented scenario, a single instance of an intelligent agent, which is provided by the *i-collaboration 3.0* framework, is available in each of these environments (such as a contact on MSN, as a user in Twitter, and as a chatterbot in Sites and VLEs, for instance). Despite the fact that the intelligent agent appears in many different environments, the framework provides a single agent to them all. In other words, the student talks across different environments with the same intelligent agent. If a Science Computer student starts communicating with the intelligent agent in Gtalk, asking him about the 'main function' of a program: "what is a main function?" he will get an answer about the main function, as requested. A few minutes later, the student goes to the MSN and asks the same thing to the intelligent agent: "main" (because he still has doubts). At this time, the intelligent agent recognizes that this student has communicated with him through Gtalk, asking him about the same thing (and within a short time interval). The intelligent agent then reasons about the students' interactions with him, such as student question, student environment, studied contents, student exams scores, student profile and answers him with new questions: "Did we not talk about it?" "You need more help with this issue?". If the student needs more help, the intelligent agent must suggest to this student related contents based on his doubts in introduction to programming.

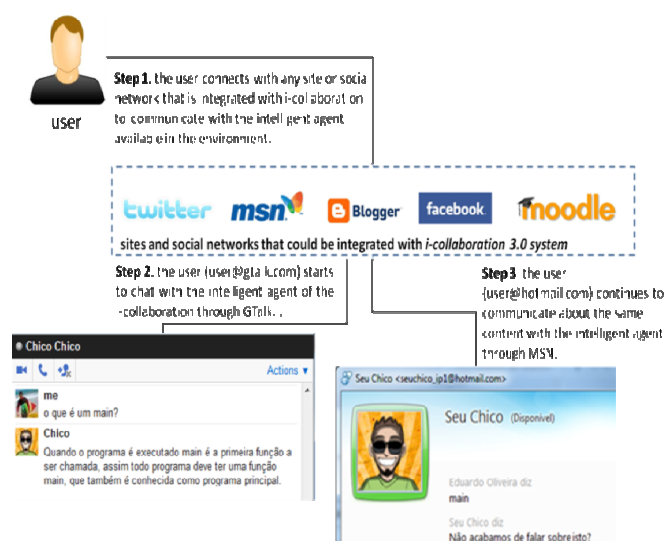


Figure 1. *i-collaboration 3.0* example of use.

To try exams and to suggest other logins on other Web tools 2.0/social networks, students use special commands such as “#exam”, to do an exam, and “#addEnvironment Gtalk mylogin@gmail.com” to set a new login to the student (student is in MSN adding a Gtalk login, for example – teaching the bot his others logins distributed in the Web).

These metadata are monitored through Drools inference engine (rule-based reasoning) [32]. Drools is responsible for integrating students distributed data and for considering context while students are making questions (repeated questions, in a small period of time, about the same subject, means that the student is finding difficulties and needs help, for example).

The advantage of providing a single intelligent agent in the system lies in the fact that with only one agent, we can also have a single integrated database in the framework (based on students’ interaction with the agent in distributed environments). If a student interacts with the intelligent agent through Facebook, the agent will know, referring to the historical database of the student that he has already communicated with him through Twitter and MSN, and that s/he has demonstrated interest in studying programming concepts before. i-collaboration 3.0 extends the model of i-collaboration 1.0 [31] in order to make it accessible to all students of VLEs with decentralized, integrated and adaptive features. If needed, for performance reasons, more intelligent agents can be used in the framework.

A big challenge in developing virtual learning spaces that offer interoperability of distributed data on the Web, is the personalization of these distributed contents, so that each student has their needs met in the environments that they use to acquire new knowledge.

According to Vieira and colleagues [13], the quality of context-aware services is directly related to the quality of the information collected by the systems. Context can help i-collaboration 3.0 to improve how contents are provided to each student, adapting them based on the students’ individual profiles, which are based their own needs (and on their favorite environments).

Student data are dynamically collected and analyzed based on their interaction with the intelligent agent of the i-collaboration 3.0. The architecture designed and developed to achieve the objectives of i-collaboration 3.0 was defined based on the following requirements:

Requirement 1 - Decentralization in access to contents of virtual courses

Web environments and social networks that make use of the framework are identified as clients of the framework. The client (Web environment or social network) to be integrated with i-collaboration 3.0 must implement an interface provided by the proposed framework. After the communication interface implementation by one or more Web environments, an instance of the framework, which relies on the intelligent agent of the i-collaboration 3.0, the context and personality tests modules (Myers-Briggs Type Indicator - MBTI [33]) will be available and integrated into the Web environment or social network, so students can use them.

Requirement 2 - Interoperability

Interoperability is the ability of a system (computerized or not) to communicate seamlessly with another system (similar or not). After having identified a way to allow interoperability between different systems and Web environments with the proposed framework, through the use of an intelligent agent common to all environments, our third challenge in the design of the framework was to ensure the data consolidation between the various student environments.

Requirement 3 - Consolidation of distributed data of students

i-collaboration 3.0 framework has been designed to work with a single intelligent agent and a single database. All Web environments and social networks should use the framework share this unique database. The database stores information such as the id of the student, Web environments and social networks they use to learn, areas of interest, exams scores, among other information. The intelligent agent manages and controls the student’s data stored in the database.

Requirement 4 - Personalization and Content Adaptation

A context module was designed for the contextual information that should be considered in the framework and how such information is connected. Based on data analysis (Requirement 3), combined with MBTI [33] profiles and behaviors of students, the contextual module is responsible for personalization in the learning contents.

i-collaboration 3.0 supports the creation of virtual learning spaces and proposes innovative infrastructures related to the nature of Context Aware and the Internet of Things. Each student is treated in a unique way and uses the environment they are more comfortable with, thus motivating these students to interact and learn.

The results obtained from an experiment with 65 students in a period of 60 days suggested that the framework can contribute to greater student interest in the pursuit of knowledge. We intend to carry out more detailed experiments in the near future, as a way to ensure the quality of the results obtained so far.

The framework is also being expanded and integrated with new platforms (Gmail and Facebook). In the future, the framework will be available under a software license for the scientific community use.

IV. PEDAGOGICAL PRACTICES

Although providing appropriate infrastructure for virtual learning environments is a first step in the direction of reducing the challenges found in distance education, it is not enough. Besides that, we need also to provide pedagogical practices that allow students to build their own knowledge in an autonomous and reflective way.

Hence, this work presents in the following sub-sections two different pedagogical practices that fit the second main challenge discussed in the Introduction of this article: (ii) How to provide pedagogical practices in virtual learning environments that allow students to build their knowledge in an autonomous and reflective way?

The first pedagogical practice we present is the self-explanation of video recordings [30] which has been developed to the programming domain. The other practice proposes a collaborative model that combines self-regulated learning with the computational thinking domain [45].

A. Self-explanations

Usually, courses in distance education learning environments have a set of different kinds of instructional material in which teachers present pieces of the subject they are teaching every lecture. In the case of programming courses, video recordings have been proposed as an ideal instructional material for presenting the dynamic process of programming to novice learners [25].

However, when teachers are planning their courses, they should be concerned not only about the way they structure these video recordings, but also about the way students are guided while studying them. For this purpose, one example of pedagogical practice that is being developed in the programming domain is the self-explanations of video recordings [30], a practice we adopted in i-collaboration 3.0.

This practice builds on two different lines of research. First, in order to organize the set of video recordings produced by teachers, we are using the Stepwise Improvement [26] [27], a conceptual framework that describes programming as a systematic and incremental process that comprises the activities of extending, refining and restructuring code. These activities are organized in a three-dimensional space that is explored by programmers while they are building programs. This framework provides guidance regarding the structure of instructional material and, thus, novice learners can learn to program in small steps by extending, refining and restructuring pieces of code systematically and incrementally during their course.

The second line of research aims to provide guidance in the way students study and understand these video recordings. For that, we propose the use of self-explanation [22][23][24], a type of dialog that learners have with themselves while they are learning from different instructional materials. According to the literature, the process of self-explaining increases students' knowledge through the refinement of the given information, associating this new information with their prior knowledge and connecting it with other different pieces of information.

In this context, for each activity in the Stepwise Improvement framework, we specified intended learning outcomes (ILOs) as shown in Table 2. These ILOs guided the definition of teaching and learning activities and assessment tasks [29] for a programming course structured according to the framework.

Table 2: ILOs for activities in Stepwise Improvement.

Activity	Intended learning outcomes
Extension	Describe goals that should be achieved in the use case and define the action plan to achieve these goals.
Refinement	Apply the basic concepts of programming language to build a coherent program that follows the action plan previously defined.
Restructure	Evaluate the current code, define and apply an action plan to alter the current solution.

For each activity performed in the programming space, students have a teaching activity, that is, a video recorded by an expert to watch. After watching each video, students have a corresponding learning activity, that is, a set of questions that should be given to them. These questions, called self-explanation prompts, guide students to explain by themselves what they have just seen in each video.

In order to illustrate this approach, we have chosen to use The Joy of Code video recordings [28] as teaching activities. These videos teach Java programming language using the Greenfoot tool. They were produced according to the Stepwise Improvement framework and edited to match its activities. We present a printscreen of one of The Joy of Code video recordings in Figure 2.

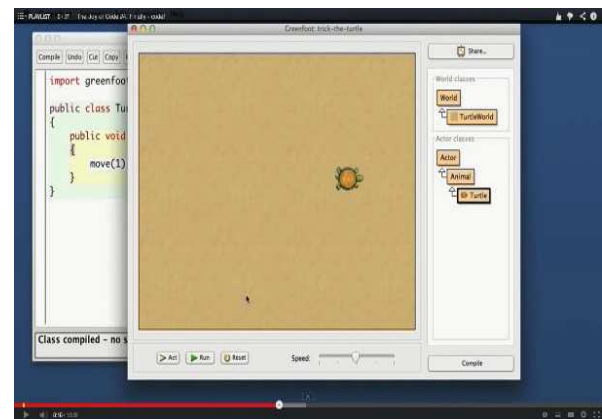


Figure 2: Printscreen of The Joy of Code video recording.

The expert that recorded the video recording presented in Figure 2 performed an extension followed by a refinement activity. Because of that, first, he described the goal he wanted to achieve with the new use case he was extending, and defined an action plan to achieve this goal.

In this case, the goal was to make the turtle move and the action plan was to write the line `move(1);` inside the `void act()` method. After that, the expert coded that plan in the program, applying the programming concepts that students had learned up to that time.

For each piece of video, we defined a corresponding set of self-explanation prompts that were aligned with the ILOs previously defined. Hence, when the students finish watching this video recording, they will be able to answer the questions presented in Table 3, which is related to these activities of extending and refining code.

Table 3: Questions related to The Joy of Code video.

Activity	Questions
Extension	What is the main goal of the piece of code that he has just written?
Refinement	How did he write code to make the turtle move? What is the purpose of the parameter (number '1') in the move method? Where did he write code to make the turtle move? When did he use the move method?

Through stepwise self-explanations, we would like to guide students in making sense out of the different pieces of content that are presented to them in different video recordings, and thus acquire knowledge. Besides, we would like to encourage them to become deeper and more reflective learners.

B. Self-regulated learning

Recent literature on the teaching of computational thinking in high school emphasizes the need to engage students in activities of reflection while they are learning computational thinking skills [47].

From the perspective of self-regulated learning [49], reflection, or more specifically self-reflection, plays an essential role in the learning process by enabling learners to evaluate their own knowledge in an attempt to identify the causes of their academic mistakes and successes. Students can perform these activities of reflection and evaluation of their own learning both individually and in collaboration with their peers.

To this end, we present penC [45], a collaborative model that introduces self-regulation practices into the domain of teaching and learning of computational thinking skills through programming.

penC is built on the notion that encouragement to students' self-reflection during problem solving triggers the development of self-regulation practices and has the potential to improve the learning of computational thinking through programming.

This model intends to create the appropriate conditions for high school students to develop skills and competencies required today, think about themselves as problem solvers and reflect on their ongoing learning experiences.

To do that, penC consists of four phases which are performed while students are solving each new programming problem.

The first phase, pre-reflection consists of two activities. In order to stimulate students' reflection on their learning process as whole, the first activity in the pre-reflection phase presents student current state with respect to the ILOs that should be achieved in an introductory programming course. The second activity aims to help the students' reflection about the problem to be solved, helping them recognize the goals and identify the data, and building on their confidence to solve it. To do that, this activity presents a set of self-assessment questions that should be answered by the student before s/he starts to solve the problem. The answers collected are used in the last phase of penC model in order to evaluate the student's level of confidence.

The second phase of the penC model is called resolution. In this phase students solve a programming problem following coding standards (e.g. naming variables). During problem solving, there are scaffolds that help learners to reflect on the current problem based on their previous experience on solving programming problems. At the end of this phase, students submit the final solution before the deadline set by the teacher, who can monitor the students with difficulties in solving the

problem and provide feedback according to the needs of learners.

Peer assessment is the third phase of the penC model. In this phase, the solutions submitted by students are evaluated by their peers. At the beginning of this phase, these solutions are assigned to at least three other students who should understand the assessment criteria. This activity is important in two aspects. The first aspect is because an inaccurate understanding of the criteria may interfere with the form of assessment and, consequently impact the learning of the students who receive the comments. The other aspect is that the teacher will be notified of those student evaluators who do not understand the criteria and can follow their assessment.

The list of criteria for assessing students' solution provides a quantitative and qualitative analysis of the solution. For each assessment criterion, the assessor should provide a score and also a written feedback to assist the student to reflect on what they have done. In this sense, scaffolds were designed to assist evaluators to write reviews that demonstrate the strengths of the solution, point errors and suggest improvements. At the end of this phase, each evaluator sends back its analysis to the creator of the solution.

The fourth phase of penC is post-reflection, which aims to engage students in reflecting about solving programming problems.

At the beginning of this phase, students can see the feedback given to their solution in the previous phase. After seeing their feedback, students also assess the quality of the received feedback. Then, students may choose to share with other students their solutions and receive new reviews, a process that stimulates new discussion about the solution shared.

At the end of the post-reflection phase, students can also monitor their learning process through different activities. First, they can analyze charts which present two aspects of monitoring capabilities: the Knowledge Monitoring Accuracy (KMA) [48] and the Knowledge Monitoring Bias (KMB) [46].

The KMA refers to the student's ability to infer how s/he will perform in a learning task and, depending on the result obtained from the KMA, the KMB indicates if the learner is pessimistic, optimistic or random. Analyzing the charts presented, students can reflect and monitor their own learning process of computational thinking. In addition, s/he can interact with the charts presented and interpreting the results in a self-assessment process. Second, they may compare their results with the results of their peers. Finally, they may discuss different problem solutions shared with their peers.

V. CONCLUSION

The Web 2.0 has changed significantly the way users relate and communicate through different digital environments. With respect to technology-enhanced learning, the consequences of this state of affairs are that students are much more comfortable using their own social tools, and thus are not happy to spend time and effort using particular VLE.

Since the Internet has a myriad of resources for communication and interaction, we now have massive amounts of stored information and knowledge produced by users. Moreover, lots of new wearable and mobile devices are available and together these technologies can bring more

possibilities for new educational strategies.

A new educational challenge for i-collaboration 3.0 framework is to provide better experiences for students, researchers, teaching staff and those in the surrounding communities in various aspects of their university campuses daily lives. In this light, the new i-collaboration 3.0 project aims to develop a context-aware middleware for a smart campus. As a middleware, we expect that i-collaboration 3.0 will be able to collect, integrate and analyze distributed and heterogeneous data from a large number of independent, autonomous, distinct and interacting sub-systems. These are the necessary support for extremely complex smart services and applications needed to create a Smarter Campus. To do this, it will be necessary to research alternative methodologies/frameworks/algorithms, specify and develop a context-aware middleware that integrates solutions and employs careful monitoring, accurate prediction of future behavior, and automatic maintenance of the involved network systems.

Besides the development of this new infrastructure, we should also be concerned about developing and evaluating other pedagogical practices that match VLEs' needs.

For instance, we promote a more active learning through the definition of a set of learning activities that engage students and, consequently, promote deeper learning in different types of activities, such as standalone or collaborative study.

In the programming domain, these activities can include production of different instructional materials by the students (e.g., video recordings of problem solutions and program codes, among others) and spaces where they can share those materials and also interact and receive feedback from their teachers and peers. Also, teachers can reuse instructional materials produced by other teachers and include learning activities, such as questions, which make students study more thoroughly while learning a specific content.

Given the importance of teaching metacognitive skills integrated with some subjects, we should provide new mechanisms that can promote other skills besides self-reflection and self-monitoring.

We also intend to carry out more elaborate experiments, in order to better evaluate the impact of such frameworks on learning.

ACKNOWLEDGMENTS

Eduardo A. Oliveira would like to thank the National Council for Scientific and Technological Development (CNPq) for supporting his postdoc position (scholarship provided under report nº BEX 9213/13-9). Viviane C. O. Aureliano would like to thank CAPES Foundation for supporting her PhD Sandwich (scholarship provided under report nº 1127-91-3) and Federal Institute of Pernambuco for supporting her PhD. Rozelma Soares de França would like to thank the CAPES foundation for granting her a MSc Scholarship.

REFERENCES

- [1] Daniel Amo Filvà, Francisco J. García-Peñalvo, and Marc Alièr Forment. 2014. Social network analysis approaches for social learning support. In Proceedings of the Second International Conference on Technological Ecosystems for Enhancing Multiculturality (TEEM '14). ACM, New York, NY, USA, 269-274. DOI=10.1145/2669711.2669910 <http://doi.acm.org/10.1145/2669711.2669910>
- [2] Merceron, A., Yacef, K. Tada-ed for educational data mining. In Interactive Multimedia Electronic Journal of Computer-Enhanced Learning, 7, 2005.
- [3] Romero, C. Ventura, S. Educational Data Mining: A Review of the State of Art. IEEE Transactions on Systems, Man, and Cybernetics – Part C: Application and Reviews. VOL. 40, NO.6., p. 601-618, 2010.
- [4] Gottardo, Ermani, and Robinson Vida Noronha. "Social networks applied to distance education courses: analysis of interaction in discussion forums." Proceedings of the 18th Brazilian symposium on Multimedia and the web. ACM, 2012.
- [5] MEYROWITZ, J., Global Nomads in the digital veldt., in Revista Famescos, julho, 2004, PUC-RS, Porto Alegre, pp. 23-30.
- [6] Dillenbourg, P. What do you mean by 'collaborative learning'? Collaborative Learning: Cognitive and Computational Approaches. Advances in Learning and Instruction Series. Elsevier Science (1998), 1-19
- [7] Martínez, Roberto, et al. "Who did what? Who said that?: Collaid: an environment for capturing traces of collaborative learning at the tabletop." Proceedings of the ACM International Conference on Interactive Tabletops and Surfaces. ACM, 2011.
- [8] Oliveira, Eduardo A., and Patrícia Tedesco. "i-collaboration 3.0: um framework de apoio ao desenvolvimento de Ambientes Distribuídos de Aprendizagem Sensíveis ao Contexto." Anais dos Workshops do Congresso Brasileiro de Informática na Educação. Vol. 1. No. 1. 2013.
- [9] Media Legislation Reports. The maturing of the MOOC: Literature review of Massive Open Online Courses and other forms of Online Distance Learning (BIS Research Paper Number 130). <http://goo.gl/twgmcn> (26-11-2014).
- [10] Baggaley, J., 2013. REFLECTION: MOOC rampant. Distance Education, 368-378.
- [11] Arcos Gracia, F., Ortega Gil, P., and Amilburu Osinaga, A., 2012. Social learning in formal education through peer collaborative networks and activities: research results and hazards. <http://goo.gl/U4Kz9a> (07-07-2014).
- [12] Oliveira, Eduardo Araujo, and Patrícia Tedesco. "i-collaboration: Um modelo de colaboração inteligente personalizada para ambientes de EAD." Revista Brasileira de Informática na Educação 18.01 (2010): 17.
- [13] VIEIRA, V., SOUZA D., SALGADO, A. C., TEDESCO, P. (2006) "Uso e Representação de Contexto em Sistemas Computacionais". In: Cesar A. C. Teixeira; Clever Ricardo G. de Farias; Jair C. Leite; Raquel O. Prates. (Org.). Tópicos em Sistemas Interativos e Colaborativos. São Carlos: UFSCAR, 2006, v. , p. 127-166.
- [14] Van Harmelen, M. and Workman, D., 2012. Analytics for Learning and Teaching. CETI Analytics Series, 1, 3. <http://goo.gl/q5Egdc>.
- [15] U.S. Department of Education, Office of Educational Technology, 2012. Enhancing Teaching and Learning Through Educational Data Mining and Learning Analytics: An Issue Brief. Washington, D.C. <http://goo.gl/IOHggU>.
- [16] Oliveira, Eduardo A., Patrícia Tedesco, and Thun Pin TF Chiu. "Creating personalized and distributed virtual learning spaces through the use of i-collaboration 3.0." Intelligent Virtual Agents. Springer Berlin Heidelberg, 2012.
- [17] Dringus, L.P., Ellis, T. Using Data Mining as a Strategy for Assessing Asynchronous Discussion Forums. Computers & Education, n. 45, p.141-160, 2005.
- [18] Li, Y. Huang, R. Analyzing Peer Interaction in Computer-Supported Collaborative Learning: Model, Method and Tool. Lecture Notes in Computer Science (LNCS). n. 5169, p.125- 136, 2008.
- [19] Pereira, J.M. Educação Superior a Distância, Tecnologias de Informação e Comunicação e Inclusão Social no Brasil. Revista de Economia Política de las Tecnologías de la Información y Comunicación, vol. XII, n.2, Mayo-Agosto, 2010.
- [20] Borba, M.C.D., Malheiros, A.P.S.D., Zulatto, R.B.A. Educação a Distância Online. 2.ed. Belo Horizonte:Autêntica, 2008.
- [21] Magnisalis, I., Demetriadis, S., Karakostas, A.: Adaptive and Intelligent Systems for Collaborative Learning Support: A Review of the Field. IEEE Trans. Learn. Technol. 4, 5-20 (2011).
- [22] Chi, M. T. H.; Bassok, M.; Lewis, M.; Reimann, M., & Glaser, R. (1989). Self-explanations: How students study and use examples in learning to solve problems. Cognitive Science, 13, 145-182.

- [23] Chi, M. T. H.; deLeeuw, N.; Chiu, M.; LaVancher, C. (1994). Eliciting self-explanations improves understanding. *Cognitive Science*, 18, 439-477.
- [24] Bielaczyc, K.; Pirolli, P.; Brown, A. L. (1995). Training in self-explanation and self-regulation strategies: Investigating the effects of knowledge acquisition activities on problem solving. *Cognition and Instruction*, 13, 221-253.
- [25] Bennedsen, J.; Carpersen, M. E. (2008). Exposing the Programming Process. In *Reflection on the Theory of Programming: Methods and Implementation*, J. Bennedsen, M. E. Carpersen, M. Kölling (Eds.) Springer-Verlag, 6-16.
- [26] Caspersen, M. E. (2007) Educating novices in the skills of programming. PhD dissertation PD-07-4, Department of Computer Science, University of Aarhus.
- [27] Caspersen, M.E. and Kölling, M. (2009). STREAM: A First Programming Process. *ACM Transactions on Computing Education (TOCE)*, Vol. 9 (1), Article No. 4.
- [28] Kölling, M. (2012). The Joy of code [Video recordings]. Retrieved from <http://bit.ly/1jbTbHq>
- [29] Biggs, J.; Tang, C. Teaching for Quality Learning at University. New York, N.Y.: McGraw-Hill/Society for Research into Higher Education/Open University Press, 2011.
- [30] Aureliano, V. C. O. Self-explaining from video recordings as a methodology for learning programming. In: the tenth annual conference, 2014, Glasgow. Proceedings of the tenth annual conference on International computing education research - ICER '14. New York: ACM Press. p. 139-140.
- [31] Oliveira, Eduardo Araujo, and Patricia Tedesco. "i-collaboration in practice: Results from our investigation within the cleveralp environment." *Intelligent Computing and Intelligent Systems*, 2009. ICIS 2009. IEEE International Conference on. Vol. 1. IEEE, 2009.
- [32] JBoss Drools - <http://www.drools.org/>
- [33] Myers & Briggs Foundation: About the MBTI Instrument. Disponível em: <http://www.myersbriggs.org>. Accessed: 04 jun, 2014.
- [34] Castro, F. Vellido, A. Nebot, A. and Mugica, F. 2007. Applying Data Mining Techniques to E-Learning Problems. In *Evolution of Teaching and Learning Paradigms in Intelligent Environment*. Berlin, Heidelberg: Springer Berlin Heidelberg, 62, pp. 183-221.
- [35] Freeman, I. and Hasnaoui, A. 2011. Society and business in the era of WEB 2.0: State of the art and prospects. [Online]. Available at: <http://aisel.laisnet.org/mcis2011/127>
- [36] Koedinger, K. R., Cunningham, A. Skogsholm, and B. Leber, "An open repository and analysis tools for finegrained, longitudinal learner data," in EDM, 2008, pp. 157-166.
- [37] Shirky, C. 2013. Social Software and the Politics of Groups. [Online]. Available at: http://www.shirky.com/writings/group_politics.html
- [38] Kontogiannis, Sotirios, et al. "Course opinion mining methodology for knowledge discovery, based on web social media." *Proceedings of the 18th Panhellenic Conference on Informatics*. ACM, 2014.
- [39] Núñez, Margarita Martínez, Oriol Borrás Gené, and Ángel Fidalgo Blanco. "Social community in MOOCs: practical implications and outcomes." *Proceedings of the Second International Conference on Technological Ecosystems for Enhancing Multiculturality*. ACM, 2014.
- [40] MARTIN, F.G. 2012. Will Massive Open Online Courses Change How We Teach? *Communications of the ACM* 55(8), 26-28.
- [41] Collazos, César A., Carina S. González, and Roberto García. "Computer Supported Collaborative MOOCs: CSCM." *Proceedings of the 2014 Workshop on Interaction Design in Educational Environments*. ACM, 2014.
- [42] Downes, S. (2013). Connective Knowledge and Open Resources: Retrieved from: <http://halfanhour.blogspot.co.uk/2013/10/connectiveknowledge-and-open-resources.html>
- [43] Collazos C., Guerrero L.A, Pino J., Renzi S., Klobas J., Ortega M., Redondo M.A., Bravo C. (2007). Evaluating Collaborative Learning Processes using System-based Measurement. *Journal of Educational Technology & Society*, Vol.10, No.3, April 2007, pp. 257-274.
- [44] ACM & IEEE Computer Society (2013). Computer Science Curricula 2013 - Curriculum Guidelines for Undergraduate Degree Programs in Computer Science.
- [45] França, R. S., & Tedesco, P. C. A. R. (2014). Um modelo colaborativo para a aprendizagem do pensamento computacional aliado à autorregulação. In *Anais do Simpósio Brasileiro de Informática na Educação* (Vol. 25, No. 1, pp. 1133-1142).
- [46] Gama, C. A. (2005). Integrating metacognition instruction in interactive learning environments (Doctoral dissertation, University of Sussex).
- [47] Lye, S. Y., & Koh, J. H. L. (2014). Review on teaching and learning of computational thinking through programming: What is next for K-12?. *Computers in Human Behavior*, 41, 51-61.
- [48] Tobias, S., & Everson, H. T. (2002). Knowing what you know and what you don't: Further research on metacognitive knowledge monitoring.
- [49] Zimmerman, B. J. (2002). Becoming a self-regulated learner: An overview. *Theory into practice*, 41(2), 64-70.
- [50] COUNCIL, N. R. *How People Learn: Brain, Mind, Experience, and School: Expanded Edition*. Washington, DC: The National Academies Press, 2000. ISBN 978-0-309-07036-2.
- [51] Crippen, K. J., & Earl, B. L. (2007). The impact of Web-based worked examples and self-explanation on performance, problem solving, and self-efficacy. *Computers & Education*, 49 (3), 809-821. doi:10.1016/j.compedu.2005.11.018
- [52] Abdul-Rahman, S. S., & du Boulay, B. (2014). Learning programming via worked-examples: Relation of learning styles to cognitive load. *Computers in Human Behavior*, 30, 286-298. doi:10.1016/j.chb.2013.09.007.



Eduardo Oliveira has a Bachelor's degree in Computer Science from the Catholic University of Pernambuco and a Master's degree in Artificial Intelligence in Education from the Federal University of Pernambuco. During his Masters and later on his PhD, Eduardo researched about distance education, complex adaptive systems, intelligent agents, context awareness and social networks. He is now a Postdoctoral Research Fellow at Department of Computing and Information Systems, The University of Melbourne.



Viviane C. O. Aureliano has a Bachelor's degree in Computer Science from Federal University of Pernambuco (UFPE, 2004) and a Master's degree in Human-Computer Interaction from Pontifical University of Rio de Janeiro (PUC-Rio, 2007). Currently, she is a teacher at Federal Institute of Pernambuco, Campus Belo Jardim and a PhD student at UFPE working mainly with pedagogical practices for programming education.



Rozelma Soares de França has a licentiate's degree in Computer from University of Pernambuco (2012). Currently, she is a master student at Federal University of Pernambuco researching about teaching of computational thinking in high school and self-regulated learning.



Patricia Tedesco has a Bachelor's degree in Computer Science from the Federal University of Pernambuco, a MSc in Computer Science also from UFPE, and a PhD from the University of Leeds. She has been a researcher in Artificial Intelligence in Education for over fifteen years, and has since supervised over 20 postgraduate students.

Aprendendo linguagem de programação através da auto-explicação de exemplos em vídeo

Viviane Cristina Oliveira Aureliano^{1,2}
Patricia Cabral Restelli de Azevedo Tedesco²

¹Instituto Federal de Pernambuco – Campus Belo Jardim
Belo Jardim, PE – Brasil

² Universidade Federal de Pernambuco – Centro de Informática
Recife, PE – Brasil

{vcoa, pcart}@cin.ufpe.br

Abstract. *This paper presents an approach to learn programming through self-explanation of video recordings examples. In order to assess the effects of the proposed approach, we carried out an experiment with beginners in programming. Although significant differences were not found in the results, they are encouraging and they suggest further investigations are needed.*

Resumo. *O presente trabalho apresenta uma abordagem para o aprendizado de linguagem de programação a partir da auto-explicação de exemplos gravados em vídeo. Para avaliar os efeitos desta abordagem, realizamos um experimento com iniciantes nesta disciplina. Apesar de não terem sido encontradas diferenças significativas, os resultados obtidos são encorajadores e sugerem que novas investigações sejam realizadas.*

1. Introdução

Disciplinas introdutórias em linguagem de programação (LP) são intrinsecamente difíceis [ROBINS; ROUNTREE; ROUNTREE, 2003], comumente relacionadas aos seus altos índices de reprovação [BENNEDSEN; CASPERSEN, 2007] e, por estarem nos semestres iniciais de muitos cursos na área de tecnologia da informação, são possivelmente culpadas pelos altos índices de evasão nestes cursos no Brasil [SILVA FILHO et al., 2007]. De fato, os obstáculos para os estudantes nestas disciplinas são vários, sendo o maior deles não conseguir combinar e utilizar adequadamente os conceitos de programação aprendidos para a construção de um determinado programa [CASPERSEN; KÖLLING, 2009].

Para minimizar as dificuldades pelas quais os iniciantes em programação passam, lançamos mão de duas linhas principais de pesquisa que foram combinadas neste trabalho: (i) o framework *Stepwise Improvement* (FSI) [CASPERSEN, 2007; CASPERSEN; KÖLLING, 2009] e (ii) as auto-explicações (AEs) [CHI et al., 1989]. Utilizamos o FSI para estruturar e organizar os exemplos em vídeo utilizados como parte do material instrucional de um minicurso introdutório de programação. Aliado a isso, utilizamos questões para promover e orientar o processo de reflexão decorrente da auto-explicação dos participantes enquanto eles estudavam estes exemplos durante a execução do minicurso.

Dentro deste contexto, o presente trabalho tem por objetivo apresentar uma abordagem de auto-explicação de exemplos em vídeo que unifica essas duas linhas de pesquisa. Ademais, descrevemos um experimento no qual analisamos os efeitos de aprender linguagem de programação através da abordagem proposta. Para alcançar estes objetivos, o restante deste trabalho encontra-se organizado da maneira descrita a seguir. Na Seção 2 apresentamos o seu referencial teórico. Na Seção 3 descrevemos a abordagem que unifica as linhas de pesquisa apresentadas na seção anterior. Na Seção 4 apresentamos o experimento realizado. Por fim, na Seção 5 apresentamos as conclusões e os trabalhos futuros decorrentes desta pesquisa.

2. Referencial teórico

Esta seção apresenta as linhas de pesquisa deste trabalho, FSI e AE, que são detalhadas nas próximas sub-seções.

2.1. Framework do *Stepwise Improvement* (FSI)

O FSI é um framework conceitual que descreve a atividade de programação como um processo sistemático e incremental que engloba três atividades diferentes, extensão, refinamento e reestruturação [CASPERSEN, 2007; CASPERSEN; KÖLLING, 2009]. A atividade de extensão ocorre quando a especificação é estendida de maneira a cobrir mais casos de uso. A atividade de refinamento ocorre quando o código abstrato é modificado de maneira a se construir código executável que implemente a especificação correspondente. A atividade de reestruturação ocorre quando é realizada uma melhoria dos aspectos não funcionais do programa, no entanto essa modificação não envolve uma mudança no comportamento aparente do programa. Estas atividades são organizadas em um espaço tridimensional que é percorrido pelos programadores enquanto eles estão construindo programas.

O FSI oferece uma contribuição significativa à área de educação em programação fornecendo orientação nas três atividades que engloba, oferecendo orientação na maneira que materiais instrucionais (e.g., livro-texto, exercícios, aulas e exemplos) podem ser estruturados e na construção de programas propriamente dita. Neste sentido, o FSI foi empregado na estruturação do livro do *Greenfoot* [KÖLLING, 2010] e na definição de um processo para programação orientada a objetos [CASPERSEN, 2007].

2.2. Auto-explicação (AE)

Apesar do FSI fornecer orientação na estruturação e organização de materiais instrucionais, ele não oferece nenhum tipo de orientação na maneira que os alunos podem estudar e aprender programação a partir destes materiais instrucionais. Em virtude disso, propomos que este tipo de orientação aconteça através da AE [CHI et al., 1989]. A AE consiste em “*um diálogo mental que aprendizes possuem enquanto estudam um exemplo trabalhado e que os ajuda a entender o exemplo e a construir um esquema a partir deste exemplo*” [CLARK; NGUYEN; SWELLER, 2005]. De acordo com Chiu e Chi (2014), a atividade de auto-explicar, ou explicar para si mesmo, promove o aprendizado através da elaboração da informação que está sendo estudada, da associação dessa nova informação com a conhecimento prévio que o aprendiz possui, da construção de inferências e da conexão dos diferentes pedaços de informação.

A AE mostrou ser positiva para estudantes de variadas disciplinas, tais como biologia [CHI et al., 1994], física [CHI et al., 1989] e programação [BIELACZYC; PIROLI; BROWN, 1995; VIHAVAINEN; MILLER; SETTLE, 2015], entre outras. Além disso, AEs podem ser obtidas a partir de diferentes formatos e modalidades de materiais instrucionais, incluindo exemplos em texto [BIELACZYC et al., 1995; CHI et al., 1989] e exercícios [VIHAVAINEN et al., 2015].

3. Auto-explicação de exemplos em vídeo

Os exemplos trabalhados¹ (*worked examples*) são uma sequência de passos que demonstram a realização de uma tarefa em particular ou a solução de um problema específico [CLARK et al., 2005]. De acordo com estes autores, os exemplos trabalhados são um tipo de material instrucional comprovadamente eficiente para alunos iniciantes em diferentes disciplinas, auxiliando-os na construção de novos esquemas enquanto eles estão adquirindo um novo conhecimento ou habilidade. Os exemplos trabalhados foram utilizados em diferentes disciplinas, tais como física [CHI et al., 1989], química [CRIPPEN; EARL, 2007] e programação [BIELACZYC et al., 1995; CASPERSEN, 2007], entre outros.

Os exemplos trabalhados são mais comumente apresentados de maneira estática, em formato textual [BIELACZYC et al., 1995; CHI et al., 1989; CRIPPEN; EARL, 2007]. Porém, eles também foram apresentados através de vídeos [CASPERSEN, 2007]. Os vídeos possuem o formato ideal para exibição de programas para iniciantes em uma disciplina de programação [BENNEDSEN; CASPERSEN, 2008]. Diferentemente dos livros que são estáticos e apresentam somente os programas resultantes, os vídeos têm a vantagem de poder combinar a apresentação dos programas com todo o seu processo de construção. Por esta razão, da mesma forma que Caspersen (2007), utilizaremos exemplos em vídeos estruturados de acordo com o FSI como parte do material instrucional disponível para os iniciantes em programação.

Como dissemos anteriormente, o FSI fornece orientação na organização dos exemplos em vídeos, no entanto, não fornece orientação na maneira como os alunos irão estudar estes exemplos. Por este motivo, durante a execução dos exemplos em vídeos, vamos estimular o processo de reflexão dos estudantes através da inclusão de questões para que eles auto-expliquem os exemplos que estão estudando. Espera-se com a inclusão destas questões que os estudantes sejam estimulados a refletir sobre os exemplos estudados e, conseqüentemente, aprendam a construir programas mais corretamente, aplicando as estruturas de programação aprendidas de maneira adequada do ponto de vista sintático e semântico.

Para que pudessemos incluir as questões nos exemplos em vídeo, estruturamo-os de forma que eles tivessem duas partes distintas. A primeira delas é a apresentação do enunciado do problema seguido por exemplos de execução do programa com entradas específicas e as saídas correspondentes. Em seguida, a segunda parte consiste na apresentação de uma possível sequência de passos que soluciona o problema apresentado. Para facilitar o processo de aprendizado dos iniciantes em programação, decidimos construir vídeos que apresentam somente sequências de atividades de

¹ Utilizaremos os termos *exemplos trabalhados* e *exemplos* indistintamente no decorrer deste trabalho.

extensões e refinamentos. Assim, a cada novo passo da solução executado no vídeo, teremos uma extensão e um refinamento.

As questões, por sua vez, foram adicionadas aos vídeos em quatro momentos distintos de sua execução: (i) um após a apresentação do enunciado do problema e de seus exemplos de entrada e saída; dois a cada novo passo da solução apresentada: (ii) um após se informar a funcionalidade que será implementada no passo e antes de se apresentar a implementação para o referido passo (quando uma extensão é executada) e (iii) outro após a implementação para o passo ser apresentada (quando um refinamento é executado); e (iv) o último após apresentar a solução completa do problema. As questões apresentadas no momento (i) tem por objetivo fazer com que o estudante auto-explique o propósito geral do programa apresentado no exemplo. As questões apresentadas no momento (ii) tem por objetivo fazer com os estudantes reflitam a respeito de uma possível solução para o passo de resolução do problema antes da solução do professor ser apresentada e, depois comparem a solução pensada por eles com a solução apresentada pelo professor. As questões apresentadas no momento (iii) tem por objetivo fazer com que os alunos façam inferências sobre a implementação apresentada pelo professor através da proposição de modificações no código apresentado. As questões apresentadas no momento (iv) tem por objetivo fazer com os alunos calculem mentalmente a saída do código a partir de uma determinada entrada, de forma que eles reflitam sobre a execução do programa apresentado.

1 of 1

1. Se eu fosse implementar esse primeiro passo (fazer com que o gato movimente-se para cima quando a seta para cima for pressionada), eu usaria a seguinte opção abaixo. Em seguida, eu vou comparar mentalmente a opção que eu escolhi com o código que será apresentado pela professora.

- ☐ Colocaria somente a condição [se <=> então] no código].
- ☐ Colocaria somente o sensor que verifica [tecla seta para cima foi pressionada?]
- ☐ Colocaria a condição [se <=> então] combinada o sensor que verifica [tecla seta para cima foi pressionada?]
- ☐ Colocaria somente [aponte para a direção 90 graus].

Submeter respostas

(a)

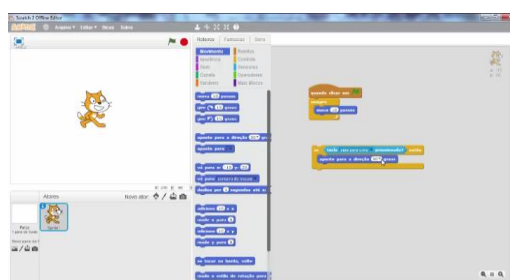
1 of 1

1. Em relação à implementação do trecho de código que acabou de ser apresentado é correto afirmar que:

- ☐ [aponte para a direção 0 graus] poderia ser retirado sem nenhum prejuízo para o funcionamento do programa.
- ☐ [tecla seta para cima foi pressionada?] poderia ser substituído por [tecla seta para baixo foi pressionada?] sem modificação no comportamento
- ☐ [mova 10 passos] dentro do [se<=>então] poderia ser substituído por [mova 20 passos] com resultado semelhante para a saída do programa.
- ☐ [aponte para a direção 0 graus] poderia ser substituído por [aponte para a direção 90 graus] sem alteração no programa.

Submeter respostas

(b)



(c)

Figura 1. Trecho de exemplo em vídeo, apresentando passo da implementação do problema (c) juntamente com as questões anterior (a) e posterior (b) à implementação do passo do problema.

Para ilustrar esse procedimento, apresentamos na Figura 1 o trecho de um dos vídeos produzidos neste trabalho, onde utilizamos a ferramenta *Scratch* para a construção de um jogo. Este trecho apresenta o passo “fazer com que o gato movimente-se para cima quando a seta para cima for pressionada” que é parte do problema “fazer com que o gato se movimente através das quatro setas direcionais”. Juntamente com o trecho de vídeo (Figura 1 (c)), apresentamos também as questões para os momentos

anterior (Figura 1 (a)) e posterior (Figura 1 (b)) à implementação do passo de resolução do problema. A cada questão apresentada, o vídeo pausa; em seguida, a questão é carregada e o estudante continua o vídeo após a submissão de sua resposta. Portanto, para que os períodos entre a parada do vídeo para responder as questões e a continuação da visualização do vídeo não fossem muito longos, todas as questões produzidas para os vídeos neste trabalho foram de múltipla escolha.

4. Descrição do experimento

O experimento realizado tem por objetivos: (i) analisar os efeitos da AE de exemplos em vídeo para o ensino introdutório de programação; e (ii) avaliar a qualidade e utilidade dos vídeos e das questões para AE do ponto de vista dos alunos. Sendo assim, as perguntas que norteiam a nossa pesquisa são as seguintes:

[QP1] O aprendizado de programação através da AE de exemplos em vídeo apresenta melhores resultados quando comparado ao aprendizado de programação somente através de exemplos em vídeo?

[QP2] Qual a opinião dos alunos em relação a qualidade e utilidade dos exemplos trabalhados em vídeo e das questões para a AE que foram utilizadas por eles?

4.1. Participantes

O experimento contou com 14 participantes voluntários. Estes participantes foram organizados em dois diferentes grupos, o grupo controle (GC) e o grupo instrucional (GI), de acordo com a disponibilidade relatada por eles para a participação no experimento. Durante a execução do experimento, o GC fez uso de exemplos trabalhados em vídeos, enquanto que o GI fez uso dos mesmos exemplos em vídeos em conjunto com as questões de AE desenvolvidas para estes vídeos.

4.2. Execução

O experimento foi realizado em uma instituição de ensino que atua na educação de nível médio, técnico e superior no Estado de Pernambuco, no primeiro semestre de 2015. Ele foi realizado em dois dias distintos (um para o GC e outro para o GI) e teve a duração de aproximadamente 4 horas. O experimento aconteceu no laboratório de informática da instituição de ensino e consistiu em uma sequência de 5 atividades: (i) questionário inicial; (ii) pré-teste; (iii) minicurso propriamente dito; (iv) pós-teste; e (v) questionário final. Antes da execução do experimento, a primeira autora do artigo explicou que o minicurso em questão se tratava de um experimento e coletou os termos de consentimento dos participantes (ou dos seus pais para aqueles menores de 18 anos). O questionário inicial teve por objetivo principal coletar informações demográficas dos alunos, além de informações sobre os seus hábitos de estudo e sua experiência prévia com vídeos para estudar. O pré-teste foi utilizado para avaliar o conhecimento prévio que os participantes possuíam antes do minicurso ser ministrado. O minicurso propriamente dito teve como principal objetivo de aprendizagem:

[OA1] Aplicar os conceitos de variáveis, estruturas de repetição e de decisão para a construção de um jogo usando a ferramenta *Scratch*.

Para alcançar este objetivo, o minicurso combinou períodos de aula expositiva seguidos por períodos de estudo individual dos exemplos em vídeo, conforme detalhado no Quadro 1. A aula expositiva foi ministrada e os exemplos em vídeo foram gravados pela primeira autora deste artigo. O pós-teste foi utilizado para avaliar o conhecimento adquirido pelos participantes logo depois do minicurso ser ministrado. Por fim, o questionário final teve por objetivo coletar a opinião dos participantes a respeito do material instrucional utilizado e do minicurso realizado.

Quadro 1. Detalhamento das atividades do minicurso.

Período	Conteúdo abordado
Aula expositiva	Introdução a programação com o <i>Scratch</i> Estrutura de repetição – sempre
Estudo dos exemplos em vídeo	Exemplo de aplicação da estrutura sempre
Aula expositiva	Estruturas de decisão – se
Estudo dos exemplos em vídeo	Exemplo de aplicação da estrutura se
Aula expositiva	Estruturas de decisão – espere até
Estudo dos exemplos em vídeo	Exemplo de aplicação da estrutura espera até
Aula expositiva	Variáveis
Estudo dos exemplos em vídeo	Exemplo de aplicação de variáveis

4.3. Material instrucional

Os materiais instrucionais produzidos para este experimento foram: (i) apresentação utilizada para guiar a aula expositiva; (ii) um conjunto de quatro vídeos que exibiam a aplicação dos conceitos apresentados na aula expositiva. Estes vídeos foram gravados pela primeira autora deste artigo; (iii) pré e pós-testes para avaliar o desempenho dos alunos antes e após o minicurso ministrado; (iv) um conjunto de perguntas para promover a AE no GI. Neste experimento, utilizamos o *Scratch* como linguagem de programação e ferramenta de desenvolvimento. Por esta razão, os jogos utilizados como parte do material instrucional desenvolvido foram baseados no *Curriculum Guide do Scratch* [BRENNAN; CHUNG; HAWSON, 2011].

5. Resultados

Os resultados que obtivemos a partir dos dados coletados serão apresentados nas próximas sub-seções.

5.1. Perfil dos participantes

Dentre os 14 participantes, 11 haviam concluído o ensino médio e cursavam o ensino técnico profissionalizante e 3 cursavam o ensino médio integrado ao técnico profissionalizante. Além disso, os 11 participantes que concluíram o ensino médio eram todos da mesma turma e estavam cursando uma disciplina introdutória de LP em C. O fato da maioria dos participantes estarem cursando uma disciplina introdutória de LP não foi um impedimento à realização do experimento, visto que todos estes 11 participantes relataram sentir muitas dificuldades no conteúdo que foi ministrado até o momento da realização do experimento (o que foi comprovado posteriormente com o pré-teste realizado). Os outros 3 participantes, que ainda cursavam o ensino médio, não haviam cursado nenhuma disciplina relacionada à LP.

O GC contou com 6 participantes, no entanto, por problemas técnicos na coleta de dados de um deles, esta análise conta com os dados de somente 5 participantes. As idades deles variavam de 16 a 18 anos (média=17.2 anos). Todos eles já haviam concluído o ensino médio e estavam cursando o ensino técnico profissionalizante. Além disso, eles estavam cursando uma disciplina introdutória de LP em C, porém nenhum deles tinha experiência com o *Scratch*. Por estarem matriculados em uma disciplina de LP, os participantes foram perguntados sobre os seus hábitos de estudo para esta disciplina. As respostas obtidas são apresentadas na Figura 2 (a). Além disso, todos os participantes relataram ter utilizado vídeos disponíveis no Youtube para estudar para a disciplina de LP. Estes vídeos foram utilizados da maneira descrita na Figura 2 (b).

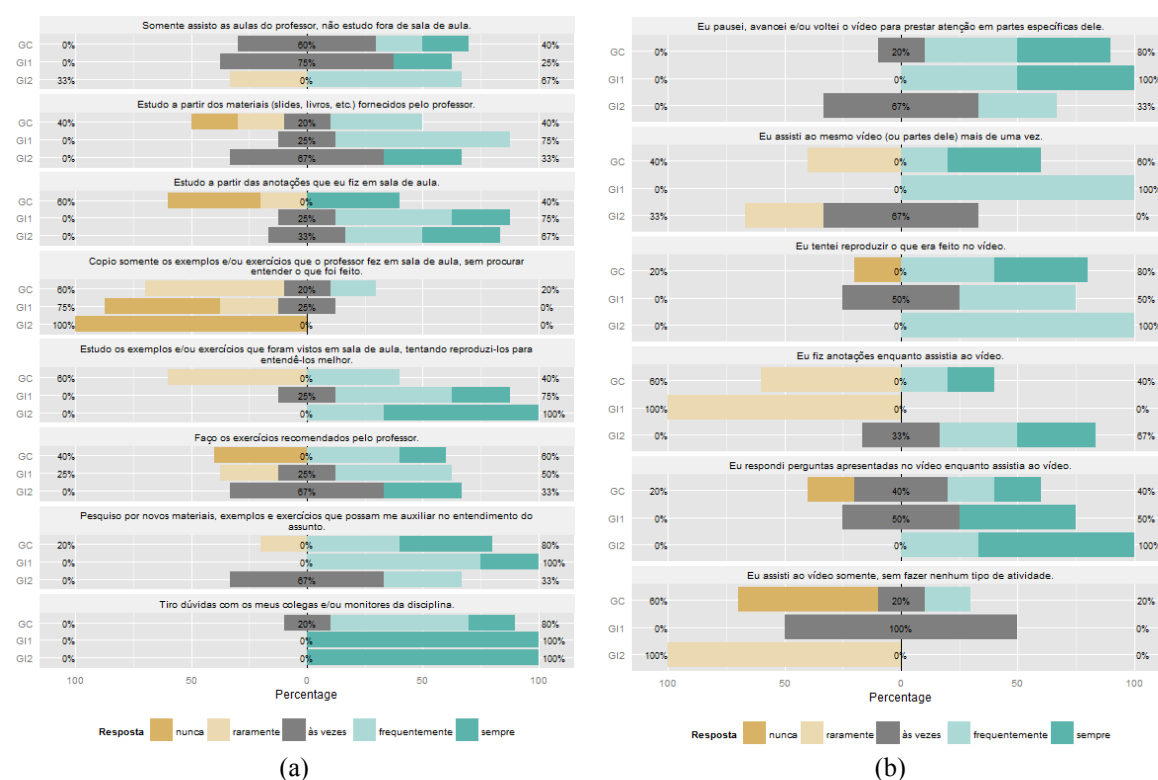


Figura 2. Hábitos de estudo (a) e utilização de vídeos (b) pelos GC, GI1 e GI2.

O GI contou com 8 participantes, no entanto, por problemas técnicos na coleta de dados de um deles, os dados de somente 7 participantes puderam ser aproveitados nesta análise. As idades deles variavam de 15 a 19 anos (média=17 anos). Este grupo englobou participantes de dois perfis diferentes, o GI1 com participantes que já haviam concluído o ensino médio e estavam matriculados na disciplina introdutória de LP em C, e o GI2 com alunos que estavam cursando o ensino médio integrado ao ensino técnico e não haviam cursado nenhuma disciplina de LP. Nenhum dos participantes tinha experiência com o *Scratch*. Todos os participantes foram perguntados sobre os seus hábitos de estudo; os 4 participantes do GI1 responderam como estudavam para a disciplina de LP na qual estavam matriculados, enquanto os do GI2 responderam como estudavam para as disciplinas em geral. As respostas obtidas são apresentadas na Figura 2 (a). Além disso, 5 participantes relataram ter utilizado vídeos disponíveis no Youtube para estudar para a disciplina de LP (GI1) ou para estudar quaisquer outras disciplinas (GI2). Estes vídeos foram utilizados da maneira descrita na Figura 2 (b).

5.2. Desempenho dos participantes

Para responder a QP1, foram analisados os pré e pós-testes dos participantes de ambos os grupos. Os testes foram equivalentes e continham 7 questões contabilizando uma pontuação máxima de 10 pontos cada um. As primeiras 4 questões aplicavam os conceitos aprendidos em contextos semelhantes aos exemplos em vídeo estudados e, por isso, valiam 1 ponto cada uma. As 3 questões restantes aplicavam os conceitos aprendidos em contextos diversos àqueles apresentados nos exemplos e, por isso, valiam 2 pontos cada uma. Na correção dos testes, se o participante tivesse atingido o objetivo completo da questão utilizando as estruturas que seriam estudadas, ele ganhava a pontuação completa; se ele tivesse atingido o objetivo parcialmente utilizando quaisquer das estruturas disponíveis no *Scratch*, ele ganhava a metade da pontuação; e, por fim, o participante não ganhava nenhum ponto, se não conseguisse atingir o objetivo da questão. Os resultados obtidos por ambos os grupos no pré e pós-testes podem ser observados na Tabela 1 e na Tabela 2.

Tabela 1. Médias e desvios padrão (entre parênteses) dos resultados dos pré e pós-testes dos GC e GI.

	GC (n=5)	GI (n=7)
Pré-teste	0.30 (0.45)	0.29 (0.27)
Pós-teste	6.20 (3.49)	7.43 (2.76)

Analisando os dados apresentados na Tabela 1, podemos perceber que GC e GI não apresentam uma diferença estatisticamente significativa em relação aos resultados do pré-teste ($U=16.50$; $z=-0.18$; $p=0.86$) e, por isso, eles são comparáveis. Embora a média dos resultados do pós-teste do GI seja superior a média dos resultados do pós-teste do GC, os dados destes grupos também não apresentam uma diferença estatisticamente significativa ($U=13$; $z=-0.77$; $p=0.44$). Analisando os dados apresentados na Tabela 2, podemos perceber que GC e GI1 quando analisados em conjunto não apresentam uma diferença estatisticamente significativa em relação aos resultados do pré-teste ($U=8.00$; $z=-0.59$; $p=0.56$) e, por isso, eles são comparáveis. O mesmo acontece para GC e GI2 em relação ao pré-teste ($U=4.50$; $z=-0.98$; $p=0.33$) e, por isso, eles também são comparáveis. Analisando os resultados do pós-teste, verificamos que GC e GI1 não apresentam uma diferença estatisticamente significativa ($U=6.00$; $z=-1.03$; $p=0.30$). O mesmo acontece para GC e GI2 em relação aos resultados do pós-teste ($U=7.00$; $z=-0.16$; $p=0.87$).

Tabela 2. Médias e desvios padrão (entre parênteses) dos resultados dos pré e pós-testes dos diferentes grupos instrucionais, GI1 e GI2.

	GC (n=5)	GI (n=7)	
		GI1 (n=4)	GI2 (n=3)
Pré-teste	0.30 (0.45)	0.13 (0.25)	0.5 (0.00)
Pós-teste	6.20 (3.49)	8.50 (1.91)	6.00 (3.46)

5.3. Opinião dos participantes

Para responder a QP2, analisamos as respostas dos participantes do experimento ao questionário final (adaptado de [RENKL, 2002]). Percebemos na Figura 3 (a) que os participantes de todos os grupos, GC, GI1 e GI2, tiveram uma opinião positiva a respeito da qualidade e utilidade dos exemplos e das explicações fornecidas pela professora nos vídeos. Percebemos também na Figura 3 (b) que a maioria dos

participantes do GI1 e GI2 teve uma opinião positiva a respeito da qualidade e utilidade das questões empregadas para promover a AE dos vídeos. As avaliações neutras também se fizeram presentes em algumas das respostas, porém elas foram em menor número.

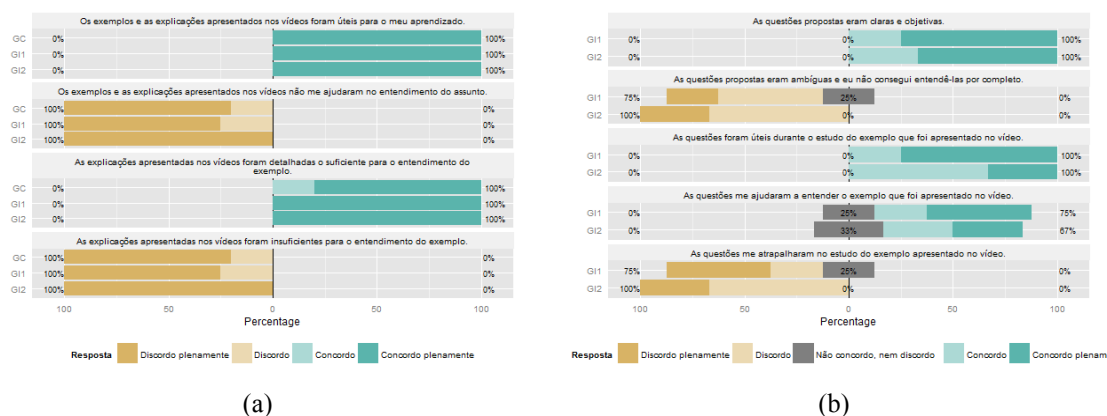


Figura 3. Opiniões dos GC, GI1 e GI2 a respeito dos exemplos em vídeo (a) e dos GI1 e GI2 a respeito das questões para AE nos exemplos em vídeo (b).

5.4. Discussão

Embora não tenha sido possível encontrar uma diferença estatisticamente significativa para os resultados obtidos no pós-teste, é interessante notar que, quando comparamos somente as suas médias, GI apresentou um resultado ligeiramente superior ao GC. Essa diferença entre os grupos é evidenciada quando analisamos os dados de GI separados pelo nível de instrução de seus participantes. Ao comparar GI1 com GC, cujos participantes possuem o mesmo nível de instrução, percebemos que a diferença das médias de GI e GC é superior a diferença das médias de GI1 e GC. Percebemos também que o desvio padrão do GI1 é inferior ao do GC, o que sugere que a aquisição de conhecimento por parte dos participantes do GI1 foi mais uniforme do que a dos participantes do GC.

Por outro lado, ao compararmos GI2 com GC, cujos participantes estão um nível de instrução acima dos participantes do primeiro grupo, percebemos que estes dois grupos apresentam desempenho e desvio padrão equivalentes, o que sugere que o conhecimento dos seus participantes no domínio de linguagem de programação foi nivelado com o minicurso. Desta maneira, esses resultados juntamente com as opiniões positivas dos participantes a respeito dos exemplos em vídeos e das questões para AE utilizados na nossa pesquisa são promissores e sugerem que novas investigações sejam realizadas.

6. Conclusões e trabalhos futuros

Neste trabalho, apresentamos uma abordagem para a AE de exemplos gravados em vídeo e também um experimento para analisar os efeitos da abordagem proposta. Apesar de não termos encontrado uma diferença estatisticamente significativa nos dados obtidos, quando comparamos as médias e os desvios padrão dos grupos encontramos que (i) GI teve um desempenho superior ao de GC; (ii) GI1 teve um desempenho superior e mais uniforme do que GC; e (iii) GI2 e GC tiveram um desempenho

equivalente, o que sugere que seus conhecimentos no domínio de programação foram nivelados. Além disso, tivemos uma avaliação positiva dos participantes no que diz respeito a qualidade e utilidade do material instrucional produzido. Juntos esses resultados são encorajadores para prosseguirmos com a nossa pesquisa que inclui o refinamento do material produzido e a realização de novos experimentos como trabalhos futuros.

Referências

- BENNEDSEN, J.; CASPERSEN, M. Exposing the Programming Process. In: BENNEDSEN, J.; CASPERSEN, M., *et al* (Ed.). **Reflections on the Teaching of Programming**: Springer Berlin Heidelberg, v.4821, 2008. cap. 2, p.6-16. (Lecture Notes in Computer Science). ISBN 978-3-540-77933-9.
- BENNEDSEN, J.; CASPERSEN, M. E. Failure rates in introductory programming. **SIGCSE Bull.**, v. 39, n. 2, p. 32-36, 2007.
- BIELACZYK, K.; PIROLI, P. L.; BROWN, A. L. Training in Self-Explanation and Self-Regulation Strategies: Investigating the Effects of Knowledge Acquisition Activities on Problem Solving. **Cognition and Instruction**, v. 13, n. 2, p. 221-252, 1995/06/01 1995.
- BRENNAN, K.; CHUNG, M.; HAWSON, J. Creative computing: A design-based introduction to computational thinking. **Retrieved May**, v. 9, p. 2012, 2011.
- CASPERSEN, M. E. **Educating Novices in the Skills of Programming**. 2007. 323 DAIMI Aarhus University
- CASPERSEN, M. E.; KÖLLING, M. STREAM: A First Programming Process. **Trans. Comput. Educ.**, v. 9, n. 1, p. 1-29, 2009.
- CHI, M. T. H. et al. Self-explanations: How students study and use examples in learning to solve problems. **Cognitive Science**, v. 13, p. 145-182+, 1989.
- CHI, M. T. H. et al. Eliciting Self-Explanations Improves Understanding. **Cognitive Science**, v. 18, n. 3, p. 439-477, 1994.
- CLARK, R. C.; NGUYEN, F.; SWELLER, J. **Efficiency in Learning: Evidence-Based Guidelines to Manage Cognitive Load**. Wiley, 2005. ISBN 978-0-7879-7728-3.
- CRIPPEN, K. J.; EARL, B. L. The impact of web-based worked examples and self-explanation on performance, problem solving, and self-efficacy. **Comput. Educ.**, v. 49, n. 3, p. 809-821, 2007.
- KÖLLING, M. **Introduction to Programming with Greenfoot: Object-oriented Programming in Java with Games and Simulations**. Prentice Hall, 2010. ISBN 9780136037538.
- RENKL, A. Worked-out examples: instructional explanations support learning by self-explanations. **Learning and Instruction**, v. 12, n. 5, p. 529-556, 10// 2002.
- ROBINS, A. V.; ROUNTREE, J.; ROUNTREE, N. Learning and Teaching Programming: A Review and Discussion. **Computer Science Education**, v. 13, n. 2, p. 137-172, / 2003.
- SILVA FILHO, R. L. L. E. et al. A evasão no ensino superior brasileiro. **Cadernos de Pesquisa**, v. 37, p. 641-659, 2007.
- VIHAVAINEN, A.; MILLER, C. S.; SETTLE, A. **Benefits of Self-explanation in Introductory Programming**. Proceedings of the 46th ACM Technical Symposium on Computer Science Education. Kansas City, Missouri, USA: ACM: 284-289 p. 2015.