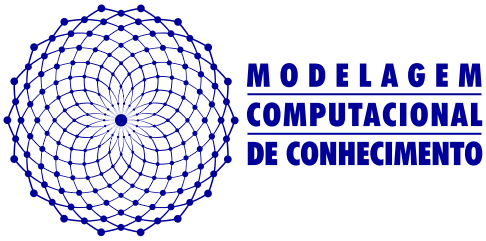


Universidade Federal de Alagoas
Instituto de Computação

Dissertação de Mestrado

Um modelo de Ambiente de Aprendizagem Social para a aprendizagem de programação baseado no conceito de planos de programação e esquemas de concepção de programas

Jalves Mendonça Nicácio
jalves.nicacio@gmail.com

Orientador:

Fábio Paraguaçu Duarte da Costa

Maceió
2017

Jalves Mendonça Nicácio

Um modelo de Ambiente de Aprendizagem Social para a aprendizagem de programação baseado no conceito de planos de programação e esquemas de concepção de programas

Dissertação de mestrado apresentada como requisito parcial para obtenção do grau de Mestre pelo Programa de Pós-Graduação em Modelagem Computacional de Conhecimento do Instituto de Computação da Universidade Federal de Alagoas.

Área de concentração: Ciência da Computação

Orientador: Fábio Paraguaçu Duarte da Costa

Maceió
2017

Catalogação na fonte
Universidade Federal de Alagoas
Biblioteca Central
Bibliotecária Responsável: Janaina Xisto de Barros Lima

N582m Nicácio, Jalves Mendonça.
Um modelo de ambiente de aprendizagem social para aprendizagem de programação baseado no conceito de planos de programação e esquemas de concepção de programas / Jalves Mendonça Nicácio. – 2017.
114 f.: il.

Orientador: Fábio Paraguaçu Duarte da Costa.
Dissertação (Mestrado em Modelagem Computacional de Conhecimento) – Universidade Federal de Alagoas. Instituto de Computação. Programa de Pós-Graduação em Modelagem Computacional de Conhecimento. Maceió, 2017.

Bibliografia: f. 50-52.

1. Ambiente de aprendizagem social (Informática). 2. Scaffolding. 3. Linguagem de concepção. I. Título.

CDU: 004.43

UNIVERSIDADE FEDERAL DE ALAGOAS/UFAL
Programa de Pós-Graduação em Modelagem Computacional de Conhecimento
Avenida Lourival Melo Mota, Km 14, Bloco 12, Cidade Universitária
CEP 57.072-900 – Maceió – AL – Brasil
Telefone: (082) 3214-1364/1825

Membros da Comissão Julgadora da Dissertação de Mestrado de Jalves Mendonça Nicácio, intitulada: "Um modelo de Ambiente de Aprendizagem Social para a aprendizagem de programação baseado no conceito de planos de programação e esquemas de concepção de programas", apresentada ao Programa de Pós-Graduação em Modelagem Computacional de Conhecimento da Universidade Federal de Alagoas, em 23 de agosto de 2017, às 10h00min, na sala de aula do PPGMCC.

COMISSÃO JULGADORA

Prof. Dr. Fábio Paraguaçu Duarte da Costa
Ufal – Instituto de Computação
Orientador

Prof. Dr. Marcus de Melo Braga
Ufal – Instituto de Computação
Examinador

Prof. Dr. Guilherme Ataíde Dias
UFPB – Departamento de Ciência da Informação
Examinador

Maceió, agosto de 2017.

Para minha filha Elis.
Descobri em seus olhos toda a inspiração que precisava para ser um pai melhor e um professor mais devotado.

Agradecimentos

Tão grande quanto o desafio de escrever esta dissertação é dedicar esta página para agradecer àquelas pessoas que fizeram parte desta minha jornada, durante o tempo em que estive mergulhado neste projeto. Sem nenhuma dúvida, a realização desta dissertação de mestrado contou com apoio e incentivo muito importantes, sem os quais este trabalho não teria se tornado realidade.

Inicio meus agradecimentos por Deus, razão de minha existência e que pôs em meu caminho pessoas especiais que sempre estão dispostas a me auxiliar.

Aos meus pais, Jurandir e Ruth, por me terem dado educação, valores e por me terem ensinado no caminho em que devo a andar. Sempre acreditaram em minha capacidade e me incentivaram a ir em frente.

Ao Professor Fábio Paraguaçu, meu orientador e exemplo profissional, por não ter permitido que eu desistisse deste processo e pela confiança. Ao Programa de Pós-Graduação em Modelagem Computacional de Conhecimento da UFAL, representado pelos professores, funcionários e colegas, e em especial aos Professores Arturo Hernández-Domínguez, Cleide Jane de Sá Araújo Costa, Eliana Silva de Almeida, Evandro de Barros Costa e Marcus de Melo Braga, cujas aulas estarão para sempre em minha memória. Meus agradecimentos também se estendem ao Vitor Torres que sempre se mostrou solícito.

À minha família por todo carinho: irmãos, cunhados, tios e sobrinhos que sempre se orgulharam de mim e confiaram em meu trabalho. À minha sogra pelas orações e apoio.

À pequena Elis, minha filha, que entrou na minha vida no meio deste processo e que, sem saber, tanto me ajudou a entender melhor a nobreza da paternidade, me mostrando dia a dia a beleza e a complexidade do ser humano que é capaz de aprender e crescer de forma magnífica.

Por fim, meu profundo agradecimento àquela pessoa que esteve incondicionalmente ao meu lado: minha esposa. Nos não raros momentos mais difíceis sempre me fez acreditar que eu chegaria até o fim. Sou grato por cada gesto de carinho e pelo seu cuidado constante. Obrigado, Elizamar, meu amor! Você acreditou em mim, mesmo quando até eu duvidava.

"Não é o crítico que conta, não é o homem que aponta como o homem forte tropeça, ou onde o realizador das proezas poderia ter feito melhor. Todo o crédito pertence ao homem que está de fato na arena; cuja face está desfigurada pela poeira e suor e sangue; aquele que se esforça corajosamente; aquele que erra, que tenta de novo e de novo, porque não existe esforço sem erros e lacunas; mas quem realmente se esforça para fazer as obras, aquele que conhece grandes entusiasmos, as grandes devoções, quem consome-se em uma causa digna é quem melhor conhece, no final, o triunfo das grandes realizações, e que, na pior das hipóteses, se falhar, ao menos não será enquanto não tiver ousado muito, de tal forma que seu lugar nunca será junto às almas frias e tímidas que não conhecem nem vitória nem derrota."

(ROOSEVELT, 1910)

Resumo

Este trabalho desenvolveu um modelo de suporte educacional para aprendizagem de conceitos de programação. Este modelo foi subsidiado pela implementação de um ambiente computacional de ensino que disponibiliza um conjunto de esquemas de concepção para a formação de modelos mentais de programação através da programação visual. Desta forma, a noção de *scaffolding*, conforme concebida por Vygotsky, foi fortemente utilizada com vistas a promover a aprendizagem dos conceitos iniciais de programação a partir das interações sociais. Um ambiente computacional de aprendizagem social foi construído e o conjunto de esquemas de concepção proposto foi incorporado a este ambiente, com o objetivo de verificar se o uso de tais ferramentas podem melhorar a metacognição do aluno em relação a aprendizagem de programação. Em seguida, um experimento foi conduzido com 37 alunos, ao longo de dois meses, e a análise dos dados obtidos aponta que a utilização do modelo sugerido neste trabalho favorece a compreensão dos conceitos iniciais de programação. Além disso, a interação com o conhecimento por meio destas ferramentas de *scaffolding* aumentou não apenas a participação individual dos alunos, mas também, a colaboração entre eles. Esses aspectos indicam o valor das atividades propostas neste estudo, no tocante a encorajar o maior engajamento dos alunos em disciplinas introdutórias de programação.

Palavras-chave: scaffolding. linguagem de concepção. padrões de programação. auto-explicação. ambiente de aprendizagem social.

Abstract

This work developed a model of educational support for learning programming concepts. This model was subsidized by the implementation of a computational teaching environment that provides a set of design schemes for the formation of mental models of programming through visual programming. In this way, the notion of scaffolding, as conceived by Vygotsky, was strongly used in order to promote the learning of the initial concepts of programming from social interactions. A computational environment of social learning was constructed and the set of proposed conception schemes was incorporated to this environment, with the purpose of verifying if the use of such tools can improve the metacognition of the student in relation to programming learning. Then, an experiment was conducted with 37 students over two months, and the analysis of the data points out that the use of the model suggested in this work favors the understanding of the concepts Programming initials. Moreover, the interaction with knowledge through these scaffolding tools has increased not only individual student participation but also collaboration among them. These aspects indicate the value of the activities proposed in this study, in order to encourage the greater engagement of students in introductory programming disciplines.

Keywords: scaffolding. design language. elementary patterns. self-explanations. social learning system.

Lista de ilustrações

Lista de tabelas

Lista de Abreviaturas e Siglas

AAC Ambientes de Aprendizagem Cognitiva

AAS Ambiente de Aprendizagem Social

IAC Instrução Assistida por Computador

IBM International Business Machines

IEEE Institute of Electrical and Electronics Engineers

STI Sistemas Tutores Inteligentes

TIC Tecnologia de Informação e Comunicação

Sumário

1 INTRODUÇÃO

No contexto da Modelagem Computacional em Educação, este trabalho apresenta um modelo de suporte educacional para aprendizagem de conceitos de programação baseada em aprendizagem cognitiva, cujas mediações do professor ocorrem em ambiente de aprendizagem social utilizando linguagem de programação visual na Web.

Partindo do conceito de utilização de *scaffolding*[1] (WOOD; BRUNER; ROSS, 1976), este ambiente disponibiliza para o aluno um conjunto de esquemas de concepção que o ajuda a realizar as tarefas de programação propostas pelo professor.

A importância deste trabalho pode ser justificada citando o problema abordado em diversos estudos, tais como Soloway e Ehrlich (1984), Almeida et al. (2002), Koliver, Dorneles e Casa (2004), Beaubouef e Mason (2005) e Silva, Cavalcante e Costa (2013), que indicam que alunos em períodos iniciais dos cursos da área de computação apresentam dificuldades na aprendizagem dos conceitos fundamentais relacionados com construção de algoritmos.

Desta forma, esta introdução abordará sucintamente duas grandes questões relacionados ao problema anteriormente citado: (1) Que dificuldades os alunos enfrentam ao aprender a programar? e (2) Quais as principais dificuldades do professor no ensino de programação?

Este capítulo está organizado da seguinte forma: na seção 1.1, serão apresentados tanto os problemas enfrentados pelos alunos, como as dificuldades do professor em utilizar uma metodologia de ensino adequada. Em seguida, na seção 1.3, são apresentados os objetivos do trabalho. Por último, na seção 1.4, apresenta-se como o trabalho está organizado.

1.1 Contextualização e Motivação

Nas sub-seções que se seguem, apresentaremos a problemática abordada nesta dissertação sob dois pontos de vista: a da aprendizagem e do ensino de programação.

1.1.1 Dificuldades para aprender a programar

O relatório da IEEE , que trata das Diretrizes curriculares para programas de Graduação em Computação (IEEE, 2013) afirma que alunos formados em ciências da computação devem ter algumas competências fundamentais. Para este relatório, os profissionais desta área devem incorporar um estilo característico de pensamento: uma habilidade de racio-

[1] O termo em língua inglesa *scaffolding* é utilizado na engenharia civil para denotar a colocação de andaimes e outras estruturas de forma a suportar temporariamente os operários de uma construção. Este termo foi utilizado por Wood, Bruner e Ross (1976) no contexto da aprendizagem para designar um suporte que é dado ao indivíduo no estágio inicial de aprendizagem.

cínio lógico para resolução de problemas, proveniente de experiências obtidas através do estudo e da prática profissional.

Dentre as disciplinas que trabalham esta competência de raciocínio lógico-dedutivo para resolução de problemas, encontram-se disciplinas como introdução a programação ou algoritmos. Segundo Santos e Costa (2006), um algoritmo é a descrição de um conjunto de comandos que, quando obedecidos, resultam em uma sucessão finita de ações.

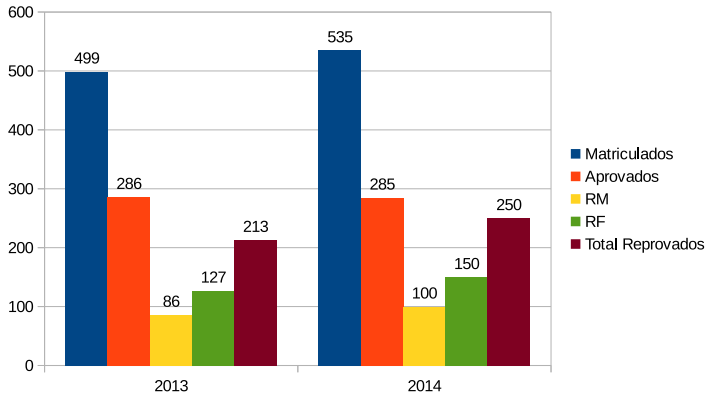
Também o Ministério da Educação (MEC) (SESU, 1999), no documento de Diretrizes Curriculares, caracteriza a habilidade de solucionar problemas como a principal atividade na tarefa de programação. Segundo explica o Ministério, "o estudo de programação não se limita ao estudo de linguagens de programação".

Contudo, por repetidas vezes o processo de ensino e aprendizagem de algoritmos é alvo de pesquisas científicas devido às dificuldades e problemas observados por professores sobre o desempenho dos alunos.

De fato, estudos revelam que um número considerável de alunos em períodos iniciais dos cursos da área de computação apresentam dificuldades na aprendizagem dos conceitos fundamentais relacionados com construção de algoritmos ((ALMEIDA et al., 2002), (KOLIVER; DORNELES; CASA, 2004), (BEAUBOUEF; MASON, 2005), (SILVA; CAVALCANTE; COSTA, 2013)). Tais dificuldades tornam-se mais evidentes quando se observam as taxas de reprovação das disciplinas introdutórias a programação, que variam entre 30% e 40% segundo Beaubouef e Mason (2005).

As informações fornecidas pela Universidade Federal de Pernambuco (UFPE) e Universidade Federal de Alagoas (UFAL) ratificam estes índices[2], conforme ilustra a figura 1 e demonstra a tabela 1.

Figura 1 – Dados sobre aprovação em disciplinas introdutórias a programação na Universidade Federal de Pernambuco (UFPE)



Fonte: Autor.

[2] Os Dados fornecidos pelas Instituições de Ensino a partir do Sistema Eletrônico do Serviço de Informações ao Cidadão (e-SIC). O acesso à informação é garantida pela Lei de Acesso à Informação (BRASIL, 2011)

Segundo os dados fornecidos pela UFPE, 499 alunos se matricularam em disciplinas com ementas básicas de algoritmos e programação no ano de 2013. Destes, 57% foram aprovados, mas 43% dos alunos foram reprovados. Outra informação relevante é que mais de 1/4 dos alunos matriculados desistiram da disciplina. Em 2014, houve um aumento tanto no número de alunos reprovados por média quanto no número de desistências. Ao todo, em 2014, 46% dos alunos matriculados nessas disciplinas foram reprovados.

Já na Universidade Federal de Alagoas, a taxa de reprovação em 2013 foi de 40% no curso de Ciência da Computação do campus de Maceió, mas o mesmo curso apresentou uma melhora no ano seguinte, com a taxa de reprovação reduzida para 24%. No entanto, os índices do curso de Ciência da Computação no campus de Arapiraca foram alarmantes em 2014, apresentando uma taxa de 66% neste ano.

Tabela 1 – Dados sobre aprovação na disciplina de Laboratório de Programação nos cursos de Ciência da Computação na UFAL - Campi Maceió e Arapiraca

Campus	**Disciplina**	**Ano**	**Matriculados**	**Aprovados**	**Reprovados**	**% Reprovados**
Arapiraca	Lab. Prog. 1	2013	76	38	38	50%
Arapiraca	Lab. Prog. 1	2014	47	16	31	65,9%
Maceió	Lab. Prog.	2013	86	51	35	40,7%
Maceió	Lab. Prog.	2014	96	55	17	23,6%

Muitas pesquisas procuram apresentar justificativas para esta insuficiência no aprendizado de programação. Koliver, Dorneles e Casa (2004) explicam que muitas vezes o professor inicia a disciplina com uma perspectiva equivocada a respeito das habilidades de seus alunos tanto para análise e resolução de problemas, como para sua capacidade de sintetização de uma solução para um determinado problema proposto. Segundo esses autores, geralmente os alunos chegam na disciplina sem nenhuma habilidade de abstração.

Almeida et al. (2002) afirmam que o problema é devido a uma forte carga de conceitos abstratos que permeiam todo esse conhecimento envolvido na atividade de programação. Segundo Lemos, Barros e Lopes (2003, p. 3), "há um consenso de que a maior dificuldade nos cursos introdutórios de programação encontra-se na aquisição da habilidade em resolução de problemas dentro do contexto do computador".

1.1.2 Dificuldades para Ensinar a programar

No lado oposto desta problemática, encontram-se os professores que buscam meios eficazes para o ensino de programação de computadores. Também nesta área, diversas pesquisas vêm sendo desenvolvidas, como em Soloway (1986), Brusilovsky e Outros (1994), Delgado et al. (2005), Denhadi (2009), entre outros. Tais pesquisas, em sua maioria, são direcionadas especialmente para alunos iniciantes em programação.

Quando se trata de aprendizado de programação, Weber, Brusilovsky e Stenle (1996) listam uma série de habilidades que devem ser trabalhadas no contexto da sala de aula,

a saber: resolução de problemas; raciocínio lógico aplicado à programação; domínio da sintaxe de uma linguagem de programação e a utilização de um ambiente de programação para teste e depuração.

No entanto, muitos professores podem se sentir confusos no planejamento de suas aulas. As principais dúvidas que eventualmente surgem são: "que linguagem de programação pode ser usada?", ou "Que ferramentas e ambientes podem suportar a aprendizagem? e de que maneira?", ou ainda "Que metodologia de ensino pode ser utilizada de forma que ajude a alcançar os resultados esperados?"

Procurando responder esses questionamentos, uma série de abordagens são aplicadas, procurando utilizar todas as ferramentas possíveis para facilitar a compreensão e aprendizagem dos alunos. Sanches (2002) propõe que o ensino de algoritmos seja feito utilizando o paradigma Orientado a Objeto.

Noschang et al. (2014) apresentam uma ferramenta de apoio para as disciplinas iniciais da área de programação onde os alunos podem implementar e testar suas soluções lógicas utilizando a linguagem Portugol. Rapkiewicz et al. (2006) defendem o uso de jogos computacionais como uma estratégia pedagógica para amenizar os problemas de aprendizagem.

Batalha (2008), por sua vez, procura analisar as possibilidades da utilização de um software de compilação de programas de computador, no processo de ensino e aprendizagem de algoritmos e conclui que o uso do compilador pode fomentar maior motivação, atenção e interação.

O trabalho de Pears et al. (2007) se ocupa em catalogar e classificar diversos estudos sobre o ensino e aprendizagem de programação com o objetivo de auxiliar os professores no momento em que estão planejando suas aulas. Conforme o levantamento feito pelos autores em 180 artigos sobre ensino introdutório de programação, as pesquisas se concentram basicamente em quatro categorias:

i. Estudos sobre o currículo de cursos introdutórios de programação - muitos currículos para ensino introdutório de programação tem sido propostos. Dentre eles, destacam-se: currículos que assumem a computação como um cálculo; currículos que assumem a computação como uma forma de interação; currículos de programação orientado a objeto e currículos de programação funcional.

ii. Métodos pedagógicos que dão suporte ao ensino de programação - Segundo os autores, enquanto o currículo dos cursos introdutórios define o que deve ser ensinado, os métodos pedagógicos tratam da maneira como o ensino e a aprendizagem são gerenciados para facilitar os resultados de aprendizagem desejados (PEARS et al., 2007).

iii. Pesquisas sobre a escolha da linguagem de programação - Segundo os autores, atualmente, as linguagens C, Java e C++ estão no topo da lista como as mais utilizadas,

tanto na indústria como em educação.

iv. Desenvolvimento de ferramentas computacionais para ensino de programação.

O presente trabalho está relacionado com a segunda e quarta categoria. Utilizou-se, no contexto deste trabalho um protótipo de um ambiente de aprendizagem social (CHAN, 1995), onde foi inserido um conjunto de ferramentas cognitivas para auxiliar na aprendizagem dos conceitos iniciais de programação, a saber: linguagem de programação visual (MCINTYRE, 1998), além de um suporte metodológico para ensino de programação baseado em linguagem de concepção (WINOGRAD, 1996), planos e padrões de programação (SOLOWAY; EHRLICH, 1984).

A motivação deste trabalho está relacionada com o fato de que muitos alunos não conseguem desenvolver o raciocínio lógico necessário para o desenvolvimento de programas devido ao alto nível de dificuldade no processo. A baixa motivação dos estudantes, a apatia, e baixa auto estima também são resultantes deste grau de complexidade, culminando, como já foi mostrado, em altos índices de reprovação e evasão. Assim, buscar alternativas para motivar o aprendizado é fundamental para uma melhora no ensino de programação.

1.2 Problema de Pesquisa

Em Dijkstra (1975) é ressaltado que programar, mais que qualquer outra atividade, envolve a habilidade de pensar de forma mais eficiente e muitos estudos concordam que a programação é uma das várias habilidades que são esperadas dos estudantes de computação [(IEEE, 2013), (SESU, 1999)]

Entretanto, conforme demonstrado, o aprendizado de programação não é algo simples de se alcançar, principalmente para os iniciantes em programação, conforme explicam Robins, Rountree e Rountree (2003).

A preocupação com o ensino e aprendizagem de programação vem de longa data. Em 1974, David Gries já afirmava que um curso introdutório deveria se preocupar com 3 aspectos da programação: *(i)* Como resolver problemas?, *(ii)* Como descrever uma solução algorítmica para um problema? e *(iii)* Como verificar se um algoritmo está correto? (GRIES, 1974).

No ensino tradicional de programação, o professor se concentra em uma determinada linguagem de programação, procurando transmitir para os alunos uma série de comandos e códigos desta linguagem enquanto demonstra de forma expositiva a solução linha-a-linha de um problema constante de uma lista de exercícios. No entanto os altos índices de reprovação e evasão levantam a questão de que talvez a abordagem tradicional pode não ser a metodologia de ensino mais adequada.

É inegável que os processos que envolvem ensinar e aprender algoritmos devem ser questionados. Diante de tudo que foi exposto, este trabalho procura responder a seguinte

questão: Como elaborar um modelo que ajude a concepção de programas usando um ambiente de aprendizagem social e linguagem de programação visual?

Como hipótese desta pesquisa apresentamos a seguinte afirmativa: é possível produzir melhores resultados quanto à internalização de conceitos iniciais de programação através da construção de esquemas de concepção que trabalhem com ferramentas cognitivas (tais como linguagem de concepção, planos e padrões de programação) disponíveis em um ambiente de aprendizagem social utilizando linguagem de programação visual.

1.3 Objetivos

São enumerados abaixo os dois objetivos principais do presente trabalho:

1.3.1 Objetivo Principal

i. Definir um conjunto de esquemas de concepção para a formação de modelos mentais de programação através da programação visual, envolvendo conceitos iniciais de programação.

1.3.2 Objetivos Específicos

i. Estabelecer, a partir do conjunto de esquemas de concepção, um modelo de suporte pedagógico para aprendizagem de conceitos de programação para ajudar o professor na elaboração de curso introdutório de programação.

ii. Implementar um protótipo de uma ferramenta computacional para o ambiente de aprendizagem social para ensino de programação com uso de uma linguagem de programação visual.

iii. Realizar uma oficina de programação para experimentação do modelo de suporte pedagógico e do ambiente de aprendizagem social.

Devido ao elevado grau de dificuldade que os alunos dos cursos de computação encontram nas disciplinas introdutórias de programação, espera-se que tanto o suporte educacional quanto o conjunto de esquemas de concepção proposto possam auxiliar os alunos a aprenderem a programar.

Sem valorizar qualquer linguagem de programação convencional, as ferramentas propostas buscam ensinar aos alunos o raciocínio aplicado na elaboração de algoritmos, utilizando linguagem de programação visual. Desta forma, os alunos poderão aprender a resolver problemas de programação sem a necessidade de memorizar comandos de linguagens de programação imperativas.

1.4 Organização do Trabalho

Esta dissertação está dividida em seis capítulos. O conteúdo de cada um é descrito a seguir:

O **capítulo 2** tem como objetivo refletir sobre colaboração a partir da ótica de Vygotsky em ambientes interativos de aprendizagem. Por este motivo, foi introduzido neste capítulo uma descrição da teoria de Vygotsky sobre a Zona de Desenvolvimento Proximal. Também apresentamos os principais modelos de ambientes virtuais de aprendizagem e ambientes de aprendizagem de programação baseado em linguagens de programação visual.

No **capítulo 3** Introduzimos as ferramentas de ajuda a concepção - Planos de programação, modelos mentais de programação e padrões elementares de programação e linguagem de concepção. Tais ferramentas cognitivas que podem fornecer ajuda no processo de aprendizagem dos conceitos relacionados com solução de problemas utilizando programação de computadores.

O método pedagógico para o ensino de programação é apresentado no **capítulo 4**. A abordagem de ensino proposta é baseada no conceito de aprendizagem cognitiva.

O **capítulo 5** propõe uma versão inicial de uma ferramenta denominada DEKSTRA: um modelo computacional do ambiente de aprendizagem social para ensino programação com uso de uma linguagem de programação visual. São descritos o funcionamento e a estrutura do modelo.

O **capítulo 6** explora os aspectos técnicos do ambiente de aprendizagem social proposto. Este capítulo especifica a arquitetura do ambiente, descreve os requisitos e a modelagem do sistema, além de outros detalhes técnicos de implementação do ambiente DEKSTRA.

O **capítulo 7** relata uma oficina de programação realizada com 37 alunos do curso técnico de informática do Instituto Federal de Alagoas, campus Palmeira dos Índios. O experimento relatado neste capítulo foi conduzido ao longo de dois meses, utilizando o conjunto de esquemas de concepção criados a partir deste trabalho.

Finalmente, o **capítulo 8** apresenta as conclusões obtidas e, por fim, aborda a perspectiva dos trabalhos futuros a serem desenvolvidos.

2 FUNDAMENTAÇÃO TEÓRICA

Existem diferentes abordagens pedagógicas e posições teóricas sobre como se dá a aprendizagem e o desenvolvimento cognitivo do ser humano. Destacamos nesta dissertação aquelas teorias que relacionam as interações sociais com o desenvolvimento e a aprendizagem. Neste contexto, destaca-se o trabalho do psicólogo russo Lev Semenovitch Vygotsky (VEER; VALSINER, 1996) por suas influências nos estudos da linguagem, pensamento, aprendizagem e desenvolvimento humanos.

Os trabalhos de Vygotsky foram o ponto de partida para uma linha de pensamento inovadora conhecida como teoria histórico-cultural que defende que o homem não pode ser estudado separado das condições em que vive, constituindo o que Vygotsky denominou psicologia social (VEER; VALSINER, 1996). Para Vygotsky, as marcas da existência social não estão apenas nas coisas, mas na mente do ser humano, que elabora conceitos a partir dos signos com os quais se relaciona (OLIVEIRA, 1997).

Tomando por base a Teoria Histórico-Cultural de Vygotsky, alguns modelos de ambientes interacionais foram criados buscando comprovar como esta teoria sobre aprendizagem pode ser aplicada em um ambiente computacional.

Desta forma, este capítulo apresentará uma revisão bibliográfica que, para fins de estudo, encontra-se subdividida nas sessões seguintes. Primeiramente será apresentada a teoria de Vygotsky sobre a Zona de Desenvolvimento Proximal (ZDP) (VYGOTSKY, 2007). Em seguida apresentam-se resumidamente os tipos de Ambientes Interativos de Aprendizagem mais relevantes para esta dissertação, destacando neste tópico o ambiente de aprendizagem social.

2.1 Zona de Desenvolvimento Proximal

Para Vygotsky (1994) existem duas linhas principais que definem o desenvolvimento do sujeito: a linha cultural e a linha natural. A linha natural é responsável pelas ações psicológicas elementares do ser humano, sendo esta ligada a processos orgânicos e de maturação do homem.

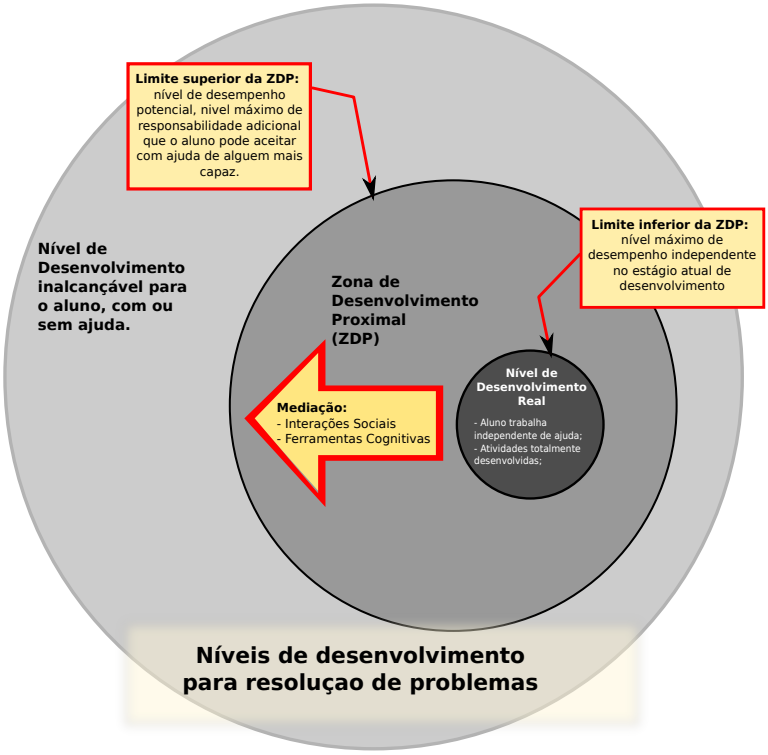
A linha cultural, por sua vez, é responsável pelo desenvolvimento das funções psicológicas superiores que são características peculiares ao ser humano e que segue as leis histórico-culturais para fomentar, na consciência do indivíduo, a sua formação enquanto sujeito.

Levando-se em consideração a perspectiva do desenvolvimento da consciência humana baseado em leis histórico-culturais, podemos perceber que o conhecimento do mundo é sempre intermediado pelas práticas culturais e pela interação com outros indivíduos. Para Vygotsky "o sujeito é interativo, pois adquire conhecimentos a partir de relações intra e in-

terpessoais e de troca com o meio, a partir de um processo denominado mediação"(PRASS, 2007).

A partir desta linha de raciocínio, o conceito de Zona de Desenvolvimento Proximal (ZDP) introduzido por (VYGOTSKY, 2007), é definido como uma região dinâmica que representa o espaço entre o **nível de desenvolvimento real** do aluno e o próximo nível (**nível de desenvolvimento potencial**) que o aluno só poderá alcançar com a assistência de alguém mais experiente. A figura 2 apresenta o conceito de ZDP postulado por Vygotsky.

Figura 2 – Zona de Desenvolvimento Proximal



Fonte: Autor.

O Nível de Desenvolvimento Real do estudante corresponde ao nível de desenvolvimento das funções mentais que já amadureceram. Neste nível de desenvolvimento as atividades podem ser realizadas sem ajuda de outras pessoas. Os conceitos pertencentes a este nível já foram internalizados pelo indivíduo, que agora possui maturidade e independência para solucionar problemas deste nível de desenvolvimento. Todas as atividades pertencentes ao Nível de Desenvolvimento Real já foram totalmente desenvolvidas pelo indivíduo.

O próximo nível de desenvolvimento a ser alcançado é identificado como o Nível de Desenvolvimento Potencial (VYGOTSKY, 2007), que pode ser determinado quando um aluno consegue resolver problemas sob a orientação de um adulto, ou em colaboração

com outros colegas mais capazes. As atividades neste nível encontram-se parcialmente desenvolvidas.

Peña-López (2012) destacou que Vygotsky (2007) descriminou dois limites para a ZDP: o limite inferior, que foi definido pelo nível máximo de desempenho independente, e o limite superior, que corresponde ao nível máximo de responsabilidade adicional que o aluno pode aceitar com a ajuda de um instrutor ou outra pessoa mais capaz.

Toda vez que um aluno demonstra um ganho em habilidades e conhecimento, a sua Zona de Desenvolvimento Proximal move-se juntamente com ele. Segundo Dennen e Burner (2008), este espaço entre sua performance atual e sua performance potencial é avaliado através da interação social entre o aprendiz e alguém que é mais experiente: um professor, os pais ou até mesmo um outro aluno mais avançado.

Conforme Prass (2007) afirma, na ZDP um adulto pode oferecer ajuda à criança de diversas maneiras diferentes, destacando-se a imitação de atitudes, os exemplos apresentados a criança, e também, acima de tudo, a colaboração em atividades compartilhadas como fator construtor do desenvolvimento.

Em suma, o aspecto essencial da hipótese de Vygotsky é que ele acreditava que a aprendizagem não deveria acompanhar o desenvolvimento, mas antes deveria liderá-lo. Em outras palavras, um aluno deve constantemente estar alcançando um pouco além de suas capacidades em vez de trabalhar dentro delas.

Foi com base nessas ideias de Vygotsky sobre a ZDP que Wood, Bruner e Ross (1976) introduziram o conceito de processos de *scaffolding* no contexto da educação. O processo de *scaffolding* pode ser definido como o suporte que é fornecido ao aprendiz, permitindo que este consiga realizar uma tarefa que, sem o uso deste suporte, seria muito difícil ou impossível que ele conseguisse realizar sozinho. O método então consiste na retirada gradual deste suporte a medida que o aluno começa a compreender a tarefa que deve ser executada.

Segundo os autores, a ideia do uso de *scaffolding* esclarece o que ocorre na ZPD da seguinte forma:

1. A tarefa não é facilitada, mas a quantidade e o tipo de assistência pode ser variável;
2. A responsabilidade é transferida progressivamente, do educador para o aprendiz;
3. O apoio prestado deve ser temporário, retirado gradualmente, conduzindo o aprendiz à autonomia na realização da tarefa.

Tendo os conceitos de ZDP e *scaffolding* em mente, faremos na próxima seção uma rápida revisão sobre Ambientes Interativos de Aprendizagem, procurando ressaltar em cada um deles as ferramentas utilizadas para dar suporte à aprendizagem do aluno.

2.2 Ambientes Interativos de Aprendizagem

Os primeiros registros do uso de equipamentos eletrônicos para dar assistência ao processo de ensino-aprendizagem surgiram no final da década de 1920, quando Sydney L. Pressey desenhou as primeiras máquinas destinadas a testar automaticamente a inteligência e o conhecimento (SEATTLER, 2004). Um pouco mais adiante, a empresa americana IBM utilizou computadores como um suporte a um curso de Matemática (CROCHIK, 1998).

De fato, segundo afirma Silva (2008), historicamente o uso de computadores na educação ocorreu inicialmente através de ambientes de instrução assistida por computador (IAC), passando por Sistemas Tutores Inteligentes (STI) e, mais recentemente, Ambientes de Aprendizagem cognitiva (AAC) e Ambientes de Aprendizagem Social (AAS). Cada um desses ambientes interativos será brevemente descrito nesta secção.

2.2.1 Ambiente de Instrução Assistida por Computador

Podemos definir esta categoria de ambiente computacional de aprendizagem como aqueles que utilizam as tecnologias da informação e da comunicação (TIC's) como meio de auxiliar a instrução dos conteúdos programados. Um dos modelos de IAC mais conhecidos é, sem dúvida, a *máquina de ensinar* desenvolvida por Skinner Skinner (1975) nos anos 60.

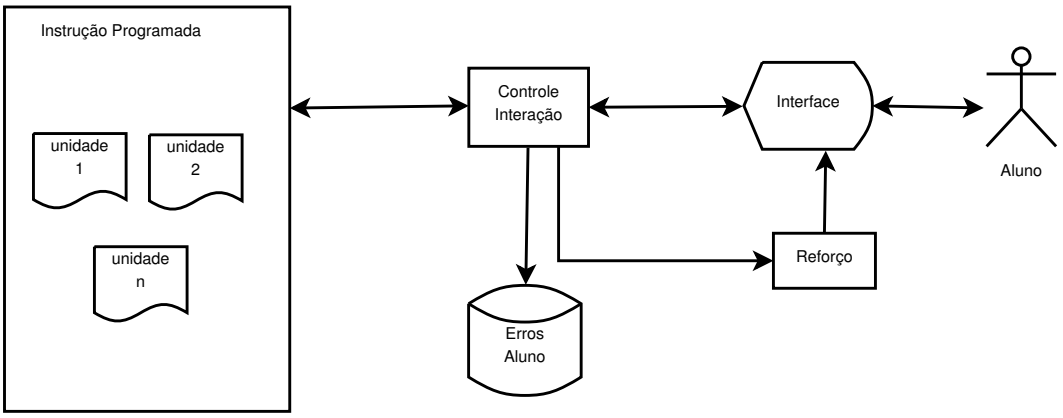
Máquinas de ensinar não eram novidade mesmo no tempo em que Skinner propôs a utilização de tecnologias computacionais para apoiar o professor em sala de aula. De fato, como já foi dito, elas existem desde a década de 1920. O que diferencia a máquina de Skinner daquelas desenvolvidas por Pressey (SEATTLER, 2004) é o fato de ela estar fundamentada em uma das teorias mais polêmicas da psicologia: o Behaviorismo. Apesar do pensamento de Skinner demonstrar claramente a influência de grandes behavioristas como Watson e Pavlov (SKINNER; VILLALOBOS, 1974), Skinner deixou um legado de grandes contribuições para a psicologia comportamental. Destaca-se da pesquisa e trabalho deste autor dois instrumentos educacionais por ele desenvolvidos: a própria *máquina de ensinar* e o modelo de instrução programada.

As *máquinas de ensinar* eram um artefato mecânico composto de cursores onde são apresentados informações ou problemas para o aluno (CROCHIK, 1998). Em alguns modelos, a máquina aguarda a resposta do aluno para uma questão de múltipla escolha. Em outros modelos de máquina, há uma pequena abertura onde o aluno escreve a resposta.

O funcionamento da *máquina de ensinar* era bastante simples, conforme mostra o modelo na figura 3. O material didático era organizado de maneira a que o aluno pudesse utilizar a máquina sozinho, recebendo estímulos à medida que avançava no conhecimento. Grande parte dos estímulos baseava-se na satisfação de dar respostas corretas aos exercícios propostos. A cada resposta certa do aluno, a máquina apresentava um reforço,

como acender uma luz ou emitir algum som. O aluno só poderia avançar para as demais unidades do conteúdo didático da máquina quando acertava todas os questionamentos da unidade anterior.

Figura 3 – Arquitetura do modelo da *máquina de ensinar de Skinner*



Fonte: Autor.

Pode-se perceber que o instrumento elaborado por Skinner leva vantagem em relação à outros instrumentos de aprendizagem já largamente difundidos no meio acadêmico, como por exemplo, os recursos áudio-visuais. Segundo (SKINNER, 1975), sua máquina supera os recursos didáticos tradicionais porque exige do estudante uma participação ativa. Outros pontos fortes do invento de Skinner podem ser considerados, tais como: respeita o ritmo do aluno, dá reforço imediato, podem atender uma demanda maior por educação (CROCHIK, 1998).

2.2.1.1 Suporte cognitivo no IAC

Para fornecer suporte cognitivo à aprendizagem do aluno, Skinner desenvolveu o conceito de Condicionamento Operante. Tal conceito consiste em observar um processo no qual se pretende condicionar uma resposta de determinado indivíduo. O objetivo do condicionamento pode ser para aumentar a probabilidade de ocorrência de uma resposta ou para extingui-la totalmente.

Por exemplo, se o que se pretende é aumentar a probabilidade de ocorrência de certo comportamento no indivíduo, então são apresentados reforços toda vez que o sujeito apresenta a resposta adequada. Percebe-se então que o conceito de reforço está diretamente ligado a ocorrência da resposta. Desta forma, o reforço pode ser positivo quando é apresentado algo ao indivíduo ou negativo quando se retira algo do ambiente.

Skinner propôs mudanças no processo de ensino de sua época, procurando introduzir sua teoria sobre condicionamento operante. Considerando que o professor sozinho não teria condições para dar reforço a todos os alunos ao mesmo tempo, houve então necessi-

dade de se introduzir instrumentos mecânicos que fossem capazes de auxiliar o professor na tarefa de reforçar as respostas dos alunos.

Assim, Skinner (1975) apresenta a ideia do uso das máquinas, até mesmo para a valorização do trabalho dos professores. Para ele, o uso das máquinas garante o aprendizado mecanicista, que os currículos pressupõem, deixando o professor com mais tempo e preocupação para questões fundamentais como ensinar o aluno a pensar. Como ele mesmo afirma:

> Naturalmente, o professor tem uma tarefa mais importante do que a de dizer certo ou errado. As modificações propostas devem libertá-los para o exercício cabal daquela tarefa. Ficar corrigindo exercícios ou problemas de aritmética – "Certo, nove e seis são quinze; não, não, nove e sete não são dezoito- está abaixo da dignidade de qualquer pessoa inteligente. Há trabalho mais importante a ser feito, no qual as relações do professor com o aluno não podem ser duplicadas por um aparelho mecânico. Os recursos instrumentais só virão melhorar estas relações insubstituíveis (SKINNER, 1975).

2.2.2 Sistema Tutor Inteligente

A instrução programada behaviorista se tornou bastante popular nos Estados Unidos nas décadas de 60 e 70. Muitos projetos de IAC neste período foram descendentes diretos das *máquinas de ensinar* skinnianas, refletindo assim a orientação behaviorista predominante na época (SEATTLER, 2004). Embora válido até certo ponto para se perceber a forma como, por exemplo, alguns bebês aprendem, o condicionamento operante defendido por Skinner fragiliza-se diante de novos estudos sobre o cérebro e a mente, bem como diante de outros trabalhos que se tornaram extremamente populares, como a teoria dos estágios de desenvolvimento cognitivo de Jean Piaget (ANTUNES, 2007).

De fato, um dos problemas na teoria do condicionamento operante de Skinner é que não se sabia por quanto tempo a mente guardava essas respostas reforçadas (ANTUNES, 2007). Outro forte argumento é que a ideia behaviorista de tentar explicar o funcionamento mental através de um esquema rígido de estímulo-resposta não é suficiente para descrever os fenômenos mais complexos associados às atividades cerebrais superiores, como, por exemplo, a linguagem humana (TEIXEIRA, 1998).

Entretanto, no final dos anos 70 a tecnologia da educação estava no limiar de uma nova orientação teórica. Como afirma Seattler (2004), ao invés de se falar em reforço, alguns educadores começaram a enfatizar alguns conceitos da então recente psicologia cognitiva. Ao contrário do que se esperava da aprendizagem behaviorista que se focava nos resultados estímulo-resposta, a perspectiva de aprendizagem cognitiva afirma que a mente constrói a sua própria realidade e que a aquisição de conhecimento só é relevante se a informação for aprendida e entendida significativamente.

Iniciava-se então um movimento para a direção de ambientes de aprendizagem que não apenas apresentassem problemas para os alunos, tabulando em seguida suas respostas, mas ambientes que considerassem o aluno como um agente ativo neste processo de aprendizagem (SUPPES, 1967). Surgem então os Sistemas Tutores Inteligentes (STI), como uma resposta às limitações impostas pelos IACs.

Define-se STI como um sistema de computador que se utiliza de técnicas de Inteligência Artificial (IA) para fornecer instruções diretas e personalizadas para alunos, sem intervenção de seres humanos. A sua estrutura é composta por quatro sub-sistemas: Módulo de interface, Módulo do aluno, Módulo do especialista e Módulo do tutor (URBAN-LURAIN, 1996).

A figura 4 representa o que Polson e Richardson (1988) chama de anatomia de um Sistema Tutor Inteligente. O módulo do especialista contém o conhecimento do domínio, a matéria que está em foco em um determinado STI. Ou seja, é o modelo do especialista, que pode ser construído através de um Sistema Especialista.

O Módulo do aluno é definido por um modelo do estudante (*Student Model*), que deve apresentar um diagnóstico do que o aluno conhece, inclusive seus equívocos e lacunas no conhecimento do domínio. O que distingue Sistemas Tutores Inteligentes de ambientes IAC (*instrução assistida por computador*) é justamente o objetivo de tornar o STI capaz de corresponder ao estilo de aprendizagem do aluno, trazendo uma instrução personalizada.

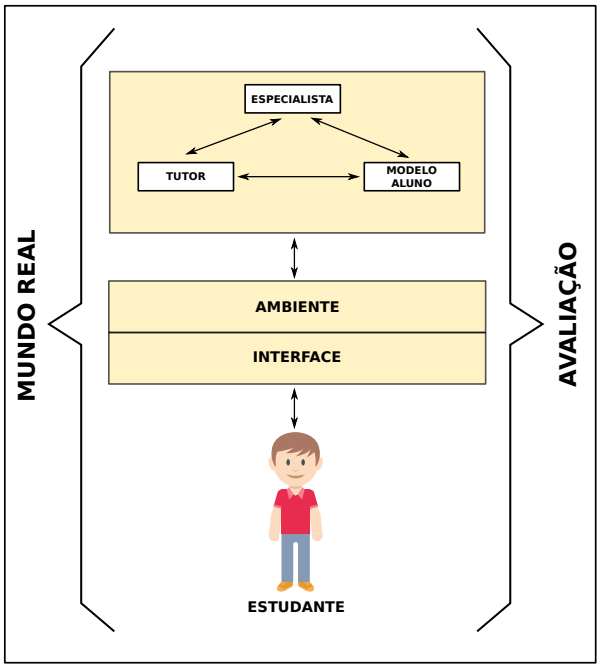
O Módulo do tutor faz a ponte entre o modelo do especialista e o modelo do estudante. Quando o módulo do tutor verifica uma discrepância entre o conhecimento do aluno e o conhecimento do especialista, ele deve agir fornecendo *feedback* para o aluno. Para ser capaz de fazer isso, ele precisa de informações sobre o que um tutor humano em tais situações faria: um modelo de tutor.

Algumas dificuldades surgem na implementação de Sistemas Tutores Inteligentes. Uma dessas dificuldades, segundo Paraguaçu (1997), é o problema da explicação, onde é difícil ou impossível representar um conjunto de conhecimento grande o suficiente para encontrar explicações para todo tipo de situação de um tutorial. Já no contexto da Educação a Distância, outro problema apontado é o problema do isolamento do estudante, onde os alunos acabam trabalhando sozinhos por horas realizando as tarefas indicadas e acabando por gerar uma frustração. Além disso, há um empobrecimento da troca direta de experiências proporcionada pela relação pessoal entre professores e alunos ou aluno e os demais colegas (FARACO; ROSATELLI; GAUTHIER, 2004).

2.2.2.1 Suporte Cognitivo no STI

Fica evidente que, em um STI, a ferramenta que é responsável por fornecer suporte cognitivo para aprendizagem é o módulo do tutor. Segundo Polson e Richardson (1988), as características do modelo do tutor são as seguintes:

Figura 4 – Arquitetura do modelo da Estrutura de um STI, considerando seus subsistemas.



Fonte: Polson e Richardson (1988)

1. Controle da representação do conhecimento a ser instruído, selecionando e sequenciando os assuntos do conteúdo estudado;

2. Capacidade de responder os questionamentos dos estudantes sobre os objetivos e conteúdos da instrução;

3. Estratégias para determinar quando os estudantes precisam de ajuda e para entregar a ajuda apropriada.

De maneira geral, um STI funciona da seguinte forma: o sistema escolhe um problema específico a partir de algum critério de progressão pedagógica existente no modelo do tutor. Em seguida, após o problema ser apresentado ao aluno através do módulo de interface, o aprendiz começa a resolver o problema. Ao mesmo tempo, o sistema também resolve o mesmo problema para, no passo seguinte, poder comparar as duas soluções, a do especialista e do aluno. O sistema então faz uma retroação pedagógica para o aluno, atualiza a informação sobre a progressão do mesmo e gera um novo problema.

2.2.3 Ambientes de aprendizagem baseados em aprendizagem colaborativa

A aprendizagem colaborativa pode ser compreendida pela combinação de alguns aspectos de três diferentes abordagens pedagógicas: a abordagem sócio-construtivista, a abordagem sócio-cultural e a abordagem da cognição distribuída (RIBEIRO, 1999). Tais

abordagens, por sua vez, são oriundas dos pressupostos da teoria histórico-cultural de Vygotsky (2007) sobre Zonas de Desenvolvimento Proximal.

Nesta seção, primeiramente apresentaremos alguns aspectos sobre cognição distribuída (2.2.3.1). Em seguida abordaremos os pontos principais em relação a aprendizagem colaborativa (2.2.3.2).

2.2.3.1 Cognição distribuída

Como as abordagens tradicionais para modelagem da cognição enfocam apenas o que ocorre na mente humana, algumas insatisfações relativas as estas abordagens surgiram, visto que elas não abordam a maneira como as pessoas interagem umas com as outras ou como elas utilizam os objetos em seu ambiente (PREECE; ROGERS; SHARP, 2005). A pesquisa de (HUTCHINS, 1995) e de outros autores surge justamente para suprir tal carência, concebendo desta forma um novo paradigma, a cognição distribuída (DCog).

Segundo (ROGERS, 1997), a cognição distribuída surgiu na segunda metade da década de 80, quando Hutchins reinvidicou que a cognição seria melhor compreendida como um fenômeno distribuído. Do ponto de vista da abordagem da cognição distribuída, a inteligência necessária para realizar uma determinada tarefa está compartilhada entre os indivíduos, o ambiente e as situações de interação entre os indivíduos e os artefatos neste ambiente (RIBEIRO, 1999).

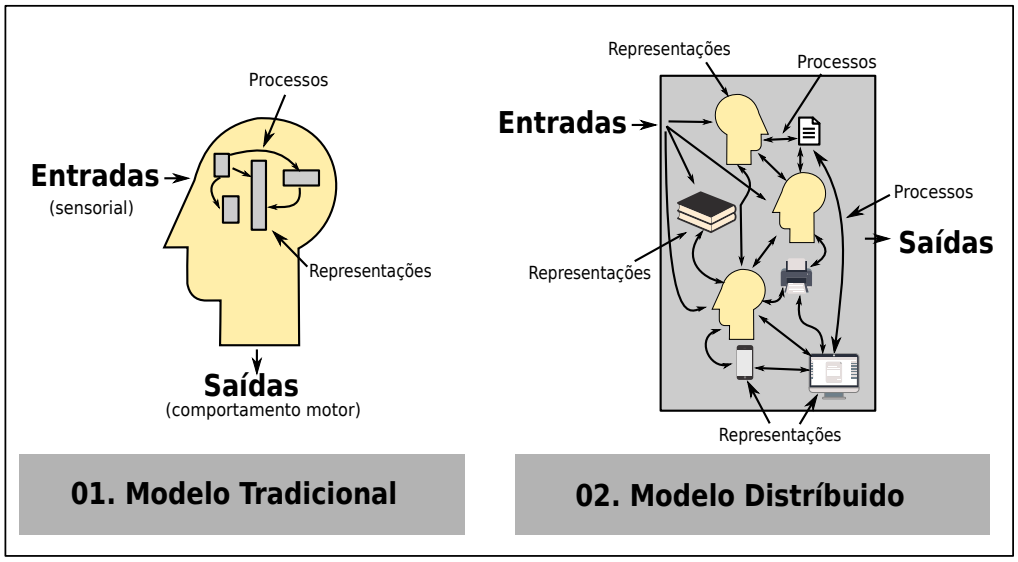
Assim, enquanto as abordagens tradicionais sobre cognição estão concentradas em descobrir o que está acontecendo na mente de cada indivíduo, a cognição distribuída está mais interessada em saber o que está acontecendo entre indivíduos e os artefatos do ambiente (PREECE; ROGERS; SHARP, 2005). Estes mesmos autores ilustram esta diferença a partir da imagem reproduzida na fig. 5, onde é possível comparar os dois modelos de cognição. Ainda na fig. 5 estão representados os elementos fundamentais da cognição distribuída, a saber:

1. Representações - São as diversas mídias por onde a informação e o conhecimento são propagados (por exemplo: mapas, livros, anotações, palavra falada, etc);

2. Processos - A informação é transformada quando transitam entre tipos de mídias diferentes. Essas transformações são as unidades centrais no estudo da cognição distribuída, e são chamadas de *estado representacional* (PREECE; ROGERS; SHARP, 2005).

No campo da cognição distribuída alguns trabalhos foram extremamente relevantes, por demonstrar exemplos de análises de sistemas cognitivos socialmente distribuídos em ambientes de trabalho. Em 1995, (HUTCHINS, 1995) descreve detalhadamente a coordenação de estados representacionais existentes na execução da tarefa de atracar um barco em um porto. Outros trabalhos também merecem destaque, como o estudo do controle do

Figura 5 – Comparação entre o modelo de cognição clássico e o modelo de cognição distribuída.



Fonte: Preece, Rogers e Sharp (2005)

tráfego aéreo (HALVERSON, 1995), e o estudo da cognição em uma cabine de comando de uma aeronave (HUTCHINS; KLAUSEN, 1996).

2.2.3.2 Aprendizagem Colaborativa

A aprendizagem colaborativa parte da ideia de que o conhecimento é o resultado de um consenso entre pessoas de uma comunidade de conhecimento, sendo construído através de diálogos e negociações em algum tipo de atividade como resolução de problemas, projetos, estudos de caso, etc. Para TORRES Patrícia; ALCANTARA e IRALA (2004), aprendizagem colaborativa "é uma estratégia de ensino que encoraja a participação do estudante no processo de aprendizagem e que faz da aprendizagem um processo ativo e efetivo"

Contudo, para Dillenbourg (1999), a definição de aprendizagem colaborativa não é algo trivial. O autor afirma que uma definição mais abrangente para aprendizagem colaborativa, já adicionando uma ressalva sobre a não satisfatoriedade da mesma, é "uma situação em que duas ou mais pessoas aprendem ou tentam aprender algo juntas".

O termo "duas ou mais pessoas" utilizado na definição anterior pode ser interpretado de diversas formas diferentes: como um par de pessoas, como um pequeno grupo de pessoas, ou uma classe com até 30 alunos. Pode ainda ser compreendido de maneira mais abrangente: uma comunidade, com algumas centenas ou milhares de pessoas, uma sociedade e assim por diante.

Analogamente, o termo "aprender algo", pode ser entendido como seguir um curso, resolver problemas, entre outros. Por fim, o elemento "juntos" da definição dada por

Dillenbourg (1999) pode ser traduzido em diferentes formas de interação: face-a-face ou mediados por computador, de modo síncrono ou assíncrono, e se o esforço é verdadeiramente comum ou se o trabalho está dividido de maneira sistemática. Todos estes aspectos são importantes para que se possa compreender o significado de aprendizagem colaborativa.

Apesar da complexidade de sua definição é fato que o contexto das interações sociais passou a ser o foco de muitas pesquisas que analisam como a aprendizagem colaborativa pode influenciar nos processos de ensino-aprendizagem (DILLENBOURG et al., 1996).

2.2.4 Ambientes de Aprendizagem Cognitiva

Em torno da teoria piagetiana, novas pesquisas foram realizadas com o propósito de obter resultados mais significativos na aprendizagem. Nas décadas de 70 e 80, Allan Collins e outros pesquisadores (COLLINS, 2006) estavam particularmente interessados em descobrir como as então novas tecnologias computacionais poderiam ajudar a transformar a escola, acreditando que seria possível a construção de avançados ambientes de aprendizagem baseados no computador. Estas pesquisas levaram à proposta da criação de um ambiente de aprendizagem cognitiva.

Segundo Collins (2006), há duas definições importantes sobre a aprendizagem cognitiva:

i A palavra "aprendizagem" no termo "aprendizagem cognitiva" é destinada para definir os processos de ensino que os especialistas usam na solução de problemas complexos, ou seja, a aprendizagem cognitiva está focada no ensino, onde o conhecimento exposto é usado na solução de problemas do mundo real, criando uma associação entre o conhecimento conceitual e o conhecimento prático. Isto é comum a algumas metodologias de ensino, principalmente àquelas que se propõe a treinar o aprendiz para uma determinada função ou profissão.

ii O termo "cognitivo" está focado nos processos e habilidades cognitivas, ao invés de processos e habilidades físicas. Na aprendizagem tradicional não é possível observar os processos e habilidades cognitivas adquiridas pelo aluno simplesmente porque eles não são visíveis. O professor resume-se apenas a avaliar os objetos concretos, o produto final da aprendizagem aplicada. Collins explica que, na aprendizagem cognitiva, antes que os métodos de aprendizagem sejam apresentados ao aluno para que ele adquira as habilidades cognitivas necessárias, o ambiente de aprendizagem deve ser alterado para que esses processos internos do pensamento sejam visíveis externamente.

Assim, podemos definir a aprendizagem cognitiva como um processo pelo qual os alunos aprendem de uma pessoa mais experiente tanto por meio de habilidades cogniti-

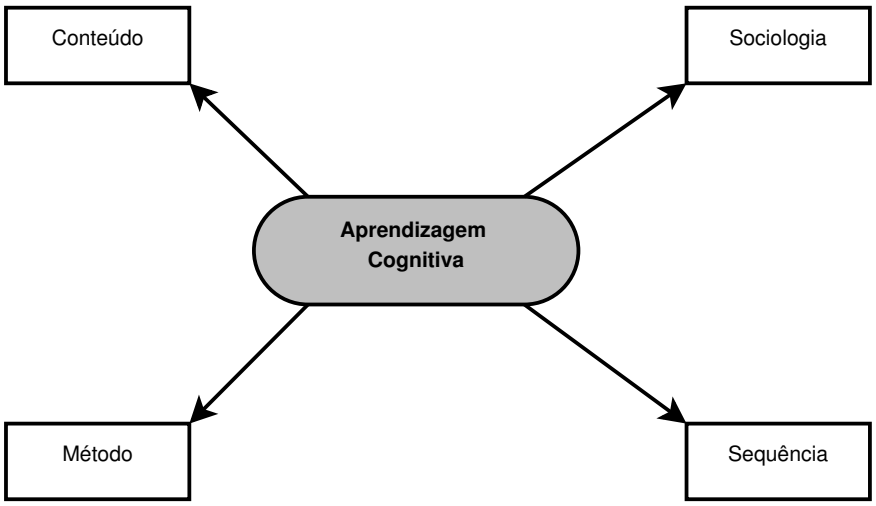
vas e metacognitivas como através processos de aprendizagem bem definidos (DENNEN; BURNER, 2008).

A afirmação de Collins (2006) é que a aprendizagem cognitiva está apoiada na realização de tarefas sequenciadas que ampliam a gradativamente sua complexidade, ilustrando o poder de certas técnicas e métodos para a solução de problemas complexos. Segundo o mesmo autor, a aprendizagem cognitiva se baseia em quatro princípios básicos para a criação de ambientes de aprendizagem cognitiva. Esses princípios estão demonstrados na figura 6: conteúdo, método, sequência e sociologia.

Em relação ao conteúdo, para a aprendizagem cognitiva não é suficiente apenas definir qual o domínio de conhecimento deverá ser apresentado para os alunos. Também é necessário planejar que estratégias de conhecimento poderão ser utilizadas para que se alcance a meta desejada.

Além de planejar o conteúdo e estratégias para a a sua abordagem, Collins (2006) destaca a importância da aprendizagem cognitiva conduzir-se de maneira gradativa e crescente, aumentando-se gradualmente a complexidade das tarefas que serão realizadas e ampliando pouco a pouco a diversidade de habilidades necessárias na resolução de problemas.

Figura 6 – Princípios para a criação de ambientes de aprendizagem cognitiva



Fonte: Autor.

2.2.5 Ambientes de Aprendizagem Social

Interações sociais podem influenciar no desenvolvimento cognitivo do indivíduo (CHAN, 1991). Em Piaget (1965) encontramos que a diferença do ponto de vista entre pares de indivíduos provoca desequilíbrio cognitivo, forçando ao indivíduo a assimilar as mudanças ocorridas.

Esta afirmação se harmoniza com a hipótese de Vygotsky (2007), que afirma que as interações sociais tem um papel fundamental nas estruturas cognitivas internas. Além disso, a aprendizagem em pares pode aumentar a motivação do aluno, fazendo-o buscar com mais afinco a realização das tarefas propostas (CHAN, 1991). Em um ambiente de aprendizagem social, o aluno pode ser levado a se desdobrar, examinar e defender suas ideias quando desafiado pelo outro estudante.

Portanto, segundo Chan (1995), Ambiente de Aprendizagem Social (AAS) é um ambiente de aprendizagem interativa suportado por computador, cuja principal característica é a presença de alguns agentes, quer humanos ou artificiais, que interagem assumindo diversos papéis e se envolvem em uma sequência de atividades sociais e educativas. Desta forma, os ambientes de aprendizagem social possuem como pressuposto a teoria histórico-cultural de Vygotsky (2007) sobre Zonas de Desenvolvimento Proximal (seção 2.1).

De acordo com Paraguaçu (1997), esses agentes interagem utilizando protocolos de comunicação social, ou seja, cooperação, competição ou colaboração. Desta forma, eles podem ser estratificados a partir de duas noções principais: a concepção de agentes lógicos e o estudo dos protocolos de cooperação, colaboração e competição.

Em relação a agentes lógicos, no contexto da EAD, há basicamente dois tipos de agentes artificiais: agentes participativos, que assumem papéis em protocolos de atividades de aprendizagem, e os agentes auxiliares que servem como assistentes pessoais, possuindo algumas capacidades em um determinado domínio, mas que escondem a complexidade da tarefa ou reduz a sobrecarga de informação (CHAN, 1995).

Paraguaçu (1997) define Ambientes de Aprendizagem Social (AAS) como uma quádrupla:

AAS = (Ag, Fcog, Fcom, AAp), onde:

(i) *Ag* é um conjunto de agentes do sistema, quer sejam humanos ou artificiais.

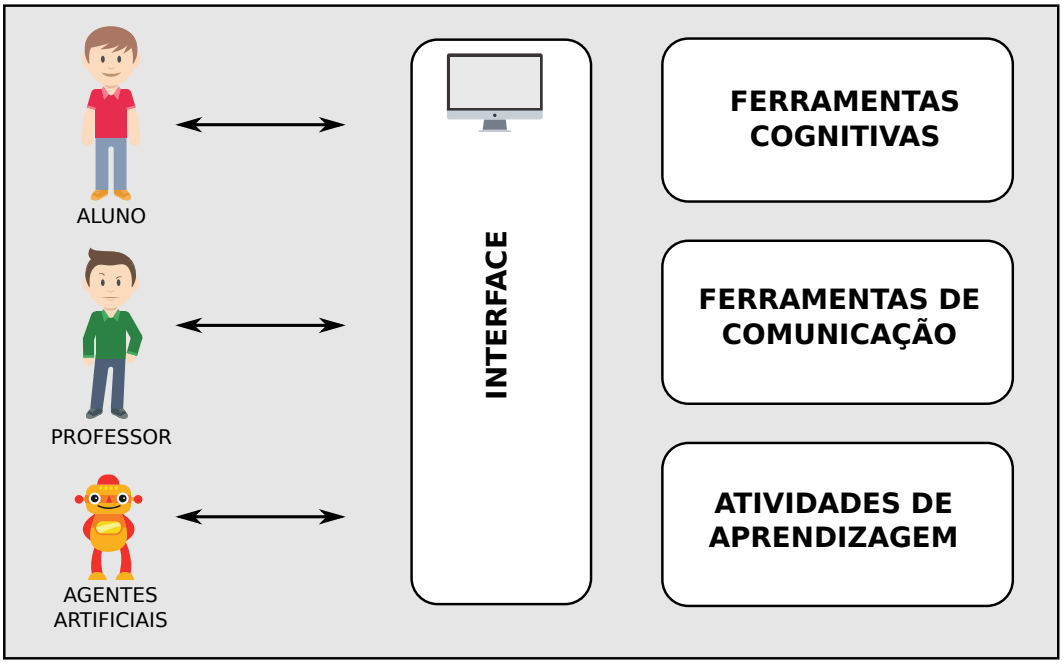
(ii) *Fcog* é um conjunto de ferramentas cognitivas que os agentes podem utilizar: simuladores, calculadoras, etc;

(iii) *Fcom* é um conjunto de ferramentas de comunicação, tais como texto, vídeo, coleção de questões e respostas;

(iv) *AAp* é um conjunto de atividades interativas de aprendizagem a partir das quais o aprendiz se envolve com outros agentes. Tais como: cooperação, colaboração ou competição.

A figura 7 representa os elementos que compõe um ambiente de aprendizagem social. Nas sessões subsequentes abordaremos com mais detalhes cada um desses itens que compõem este tipo de ambiente de aprendizagem.

Figura 7 – Arquitetura do modelo do *ambiente de aprendizagem social*



Fonte: Autor.

2.2.5.1 Conjunto de agentes

Os agentes desempenham papéis tais como: professor, aprendiz ou companheiro. Em Ambientes de Aprendizagem Social é comum a utilização de agentes artificiais, que podem ser definidos como objetos computacionais capazes de, por meio de sensores, perceber o ambiente no qual se encontram inseridos e, através de operadores, podem agir sobre este ambiente (RIBEIRO, 1999). Em relação à sua complexidade, Chan (1995) classifica os agentes artificiais em três níveis diferentes: agentes reativos, agentes de memória e agentes cognitivos.

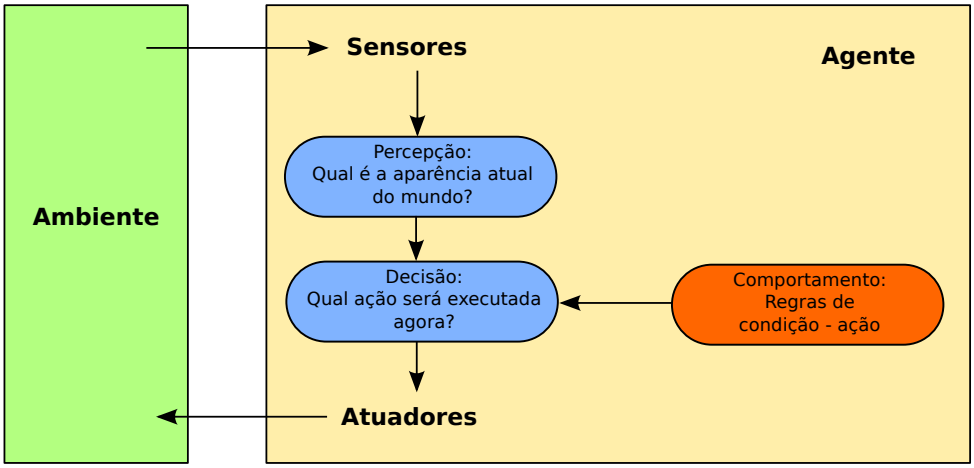
- ❏ Agentes reativos - são agentes mais simples que possuem um conjunto básico de regras que os permitem realizar alguma ação mediante uma percepção atual do estado do ambiente de aprendizagem. Segundo Russell e Norvig (2003), tais agentes selecionam ações com base na percepção atual, agindo de acordo com uma regra cuja condição corresponde ao estado atual definido pela percepção. São agentes de inteligência muito limitada.

  Os agentes reativos mais simples ignoram o histórico de percepções e leva em consideração a última percepção do ambiente. O comportamento do agente reativo é estritamente definida através do esquema "estímulo / resposta", uma noção que vem do campo da psicologia comportamentalista (BORDINI; VIEIRA; MOREIRA, 2001). A Figura 8 apresenta a arquitetura para o modelo de agente reativo simples, adaptada de Russell e Norvig (2003)

- Agentes de memória - são agentes que, para tomar uma decisão de ação, levam em consideração tanto o estado do ambiente quanto o conteúdo de sua memória interna. Russell e Norvig (2003) denominam tais agentes como agentes reativos baseados em modelo. Tais agentes são capazes de monitorar a parte do ambiente que eles desconhecem, tornando-os mais eficientes em ambientes parcialmente observáveis.

- Agentes cognitivos - agentes mais complexos que procuram simular o comportamento humano, que possui crenças, hipóteses e intenções. São os agentes com aprendizagem, capazes de operar em ambientes inicialmente desconhecidos e, com o tempo, tornam-se mais competentes do que o início de sua atuação (RUSSELL; NORVIG, 2003).

Figura 8 – Arquitetura para o modelo de agente reativo simples.



Fonte: adaptado de Russell e Norvig (2003).

2.2.5.2 Ferramentas cognitivas

Paraguaçu (1999) afirma que as ferramentas cognitivas de um ambiente de aprendizagem social servem como apoio para a colaboração, para a realização de tarefas e para a comunicação entre agentes. Em outras palavras, tais ferramentas são intermediárias da interação entre os agentes. Alguns exemplos de ferramentas que podem auxiliar o aluno na aprendizagem de programação segue abaixo:

(i) Espaço de Concepção: é um ambiente virtual para construção de algoritmos durante o processo de aprendizagem de programação.

(ii) Ferramentas Mentais: é uma coleção de objetos criados para auxiliar o aluno a partir de uma série de questionamentos, tais como "Posso atribuir um valor a uma variável?". Tal ferramenta se utiliza da formalização de uma linguagem de concepção

para programação, que será explicado na sessão 3.4. No contexto deste trabalho, ferramentas mentais serão ainda utilizadas para auxiliar o alunos a criar *planos de programação* e adquirir habilidade de decompor *metas de programação* em *padrões de programação* (maiores detalhes nas seções 3.2 e 3.3).

(iii) Linguagem de Concepção: Na interação entre os agentes do sistema é essencial trabalhar diretamente com o conhecimento estudado, como por exemplo, a aprendizagem de programação. Linguagens de concepção permitem apresentar este conhecimento de maneira estruturada, permitindo que as relações entre as partes do conhecimento sejam explicitadas. Como exemplo de linguagem de concepção, podemos citar mapas conceituais e outros esquemas de concepção como padrões de programação (aplicado ao contexto de aprendizagem de programação).

(iv) Blocos de Programação: são objetos do espaço de concepção que podem ser utilizados pelos alunos para resolverem um determinado problema de programação. Os blocos de programação seguem a estrutura básica de comandos de programação, como por exemplo: comandos de atribuição de valor a variáveis, comandos de seleção (SE...SENÃO) e comandos de repetição (REPITA, ENQUANTO).

2.2.5.3 Ferramentas de comunicação

As ferramentas de comunicação de um ambiente de aprendizagem social são aquelas que criam uma interface de comunicação entre os agentes do sistema (humanos ou artificiais). Por exemplo: textos, imagens, vídeos;

2.2.5.4 Atividades interativas de aprendizagem

Em uma atividade interativa de aprendizagem, um agente humano aprendiz irá interagir com outro agente, quer humano ou artificial, para realizar uma tarefa.

Segundo Reis (2001), o que fundamenta a qualidade da aprendizagem colaborativa é justamente o tipo de interação que existe entre os pares, uma vez que as interações sociais nem sempre geram novo aprendizado.

Assim, em um ambiente de aprendizagem social, devem ser incentivadas aquelas interações que podem levar a uma situação de aprendizagem, tais como: processos explicativos, onde o agente mais qualificado fornece uma explicação sobre a tarefa, e processos de controle, onde o agente mais qualificado conduz os demais na solução do problema.

2.2.6 Ambientes de aprendizagem baseados em linguagem visual de programação

Linguagem de programação é um conjunto de regras sintáticas e semânticas que representam um programa de computador (FISCHER; GRODZINSKY, 1993), um método

padronizado com a finalidade de comunicar instruções à máquina (DERSHEM; JIPPING, 1995).

A visão de Edsger Dijkstra sobre linguagens de programação era ainda mais aguçada. Segundo ele, a linguagem de programação pode ser vista como "um veículo para a descrição (potencialmente muito sofisticada) de mecanismos abstratos"(DIJKSTRA, 1976).

Fato é que desde os primórdios da computação, mesmo no tempo de Charles Babbage e sua máquina calculadora programável (BRETON, 1991), havia a preocupação de instruir uma máquina computável a partir de alguma linguagem formalizada. No entanto somente em novembro de 1954 surge a primeira linguagem de programação de alto nível amplamente utilizada: *Fortran* (WEXELBLAT, 1981). A partir daí, diversas outras linguagens utilizando paradigmas variados foram construídas e são utilizadas até o dia de hoje.

Dentre as linguagens de programação desenvolvidas existem aquelas cuja principal finalidade é facilitar o acesso de iniciantes ao conhecimento de programação. É o caso,por exemplo das linguagens: Logo (PAPERT, 1988), Pascal (PACITTI; AKTINSON; TELES, 1983), Portugol (MANSO; OLIVEIRA; MARQUES, 2009) e as linguagens de programação visual.

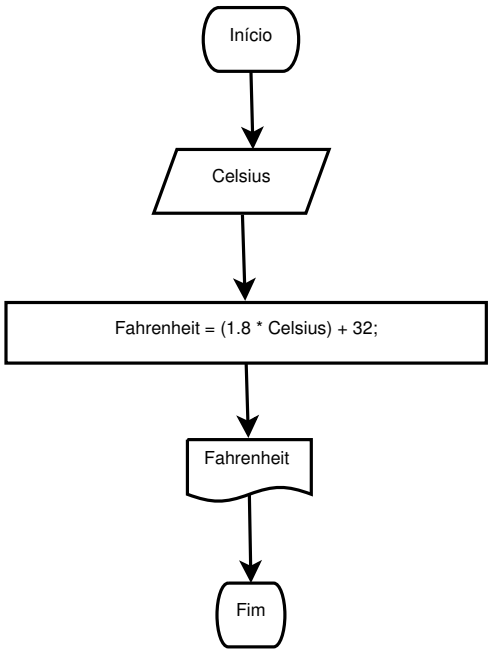
Segundo McIntyre (1998), linguagens de programação visual (VPL) são comumente definidas como linguagens de programação que utilizam expressões visuais, tais como gráficos, desenhos, animações ou ícones, no processo de programação. Estas expressões visuais podem ser entendidas como interfaces que interpretam graficamente o que pode ser transcrito para linguagens de programação textuais em ambientes de programação. Assim, as expressões visuais utilizadas formam uma nova sintaxe de programação completamente gráfica.

Do ponto de vista de Myers (1990), Programação Visual refere-se a qualquer sistema que permite ao usuário especificar um programa em duas (ou mais) dimensões. As linguagens textuais convencionais não são consideradas bi-dimensionais visto que seus compiladores ou interpretadores processam o código-fonte como fluxos unidimensionais.

Há muitos tipos diferentes de ambientes de programação utilizando linguagens visuais. Alguns tipos procuram simplesmente implementar sistemas de ação e resposta, outros tipos incluem técnicas de programação moderna como a abstração e orientação a objetos. Algumas linguagens gráficas representam o fluxo de instruções de um programa. Já outras representam o fluxo de dados de um determinado sistema. Aquelas linguagens que representam o fluxo de instruções de um programa procuram mostrar o caminho de comandos a serem executados pelo programa. O exemplo mais evidente deste tipo de linguagem gráfica é o fluxograma. O ambiente de programação SICAS, proposto por Mendes e Gomes (2000) apresenta um elevado grau de interatividade com o aluno permitindo a concepção de algoritmos de forma visual através de fluxogramas. A figura 9 apresenta um fluxograma que representa os passos para transformar temperatura Celsius

em Fahrenheit.

Figura 9 – fluxograma que representa os passos para transformar temperatura Celsius em Fahrenheit



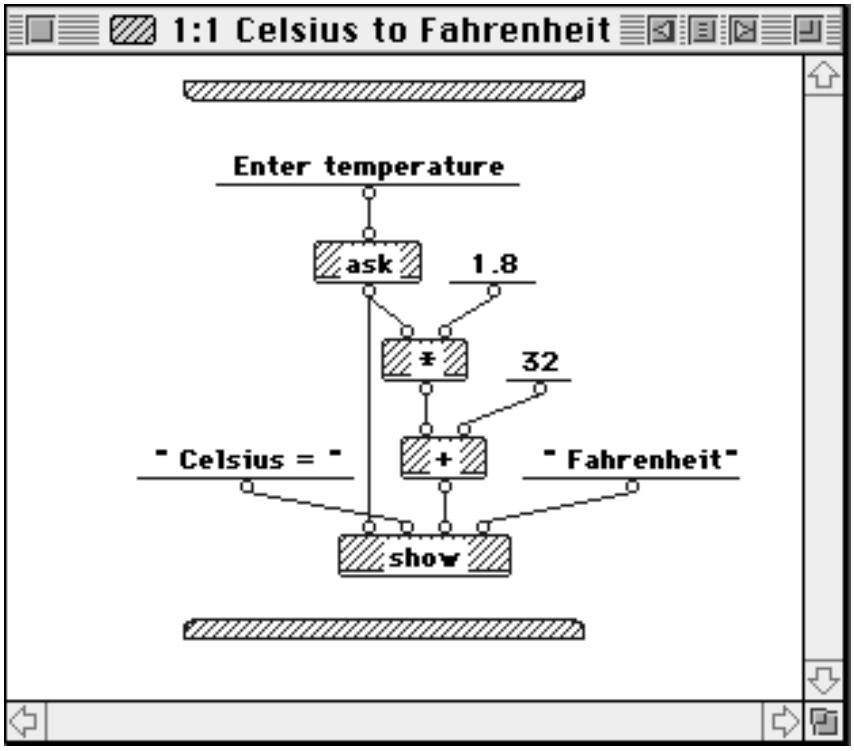
Fonte: Autor.

As linguagens visuais que representam fluxo de dados modelam a programação através da utilização de filtros aplicados aos objetos de dados. O *Prograph*, linguagem produzida pela Pictorius (1996), representa muito bem esse perfil de linguagem visual. O *Prograph* inclui encapsulamento de objetos e funções, além de ser equipado com uma interface gráfica robusta. A figura 10 apresenta um programa feito no *Prograph* que transforma temperatura medida em Celsius para Fahrenheit.

Linguagens de programação visual possuem vantagens e desvantagens quando comparadas com linguagens de programação textuais. Uma das vantagens é que o uso de representações gráficas de objetos pode ajudar no melhor entendimento sobre programação orientada a objeto. Além disso, as linguagens visuais eliminam detalhes de sintaxes que podem ser um problema para o aluno aprendiz, como o uso de chaves, parêntesis, ponto-e-vírgula, etc. Como o programa construído nesses ambientes é representado graficamente, isto melhora a visualização do caminho que o programa está a seguir. Talvez a melhor vantagem seja o uso que linguagens de programas visuais podem fazer de metáforas da vida real. Linguagens de programa visuais como *Scratch* (RESNICK et al., 2009) e *Blockly* (FRASER et al., 2013), por exemplo, representam pequenos pedaços de código como blocos de montar ou peças de quebra-cabeça. Essas peças podem ser encaixadas umas nas outras, tornando explicita a conexão existente entre objetos da linguagem.

Por outro lado, programadores experientes podem se frustrar com o uso de linguagens

Figura 10 – Programa feito com linguagem visual *Prograhp* que transforma temperatura medida em Celsius para Fahrenheit



Fonte: McIntyre (1998)

visuais, uma vez que se tenha consciência de que certas declarações poderiam ser mais concisamente expressas se forem representadas usando texto. Além disso, McIntyre (1998) criticou as linguagens programação visual uma vez que elas podem exibir na tela um número limitado de artefatos primitivos da linguagem. Esta limitação provem do tamanho mínimo que um objeto gráfico deve possuir para que ele se mantenha inteligível pelo usuário.

2.2.6.1 Ambiente de desenvolvimento *Scratch*

Em 2009, o grupo Lifelong Kindergarten, do Instituto de Tecnologia de Massachusetts (MIT) tornou público o *Web site* da ferramenta *Scratch*: um projeto open-source do mesmo grupo, onde os autores desenvolveram uma linguagem de programação visual voltada para crianças (RESNICK et al., 2009).

A linguagem *Scratch* disponibiliza comandos que permitem ao aprendiz trabalhar com conceitos computacionais importantes para iniciantes em programação de computadores, tais como entrada e saída, tipos de dados, variáveis, estruturas de controle, operadores e arrays. Além disso, também permite trabalhar com comandos que conferem a natureza multimídia inerente a esta linguagem.

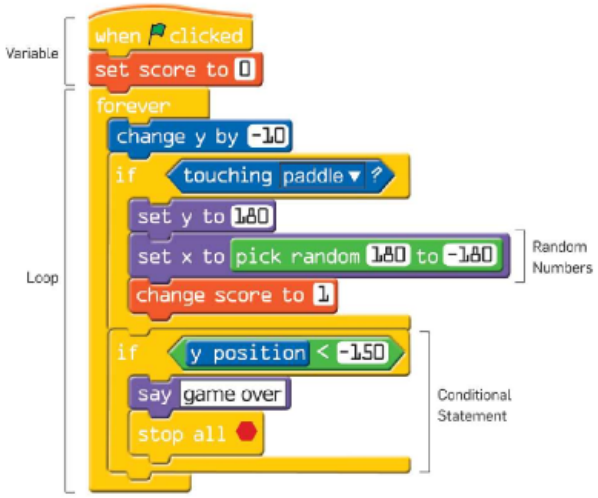
A ferramenta *Scratch* conta com um ambiente integrado de desenvolvimento (IDE)

próprio, que permite a construção visual da lógica de programação, tornando o processo de desenvolvimento de aplicações interativas mais acessível a diversas faixas etárias ou níveis de fluência computacional.

Hoje, o *Scratch* é uma das mais conhecidas ferramentas de programação visual que permite a criação de jogos simples e animações interativas de forma bastante intuitiva, além de promover uma comunidade de desenvolvedores. Na plataforma *Web* do sistema, crianças e adolescentes do mundo inteiro compartilham suas criações com a ferramenta.

Segundo Resnick et al. (2009), há três princípios primordiais nos quais o design do programa se baseia: torná-lo mais flexível para utilização, ser mais significativo para o público, e mais social do que outros ambientes de programação. Os blocos de programação em *Scratch* também possuem uma forma conveniente para facilitar a criação de macroblocos de código, tais como funções, blocos de condição ou laços de repetição. Nesses casos, os blocos de *Scratch* podem ser facilmente aninhados dentro de objetos gráficos em forma de "C", como ilustrado na figura 11, onde os blocos de condição (blocos *if*) e o bloco de repetição (*forever*) contém diversos outros blocos.

Figura 11 – Programa feito em *Scratch* mostrando conceitos computacionais e matemáticos.



Fonte: Resnick et al. (2009)

2.2.6.2 Biblioteca de programação visual *Blockly*

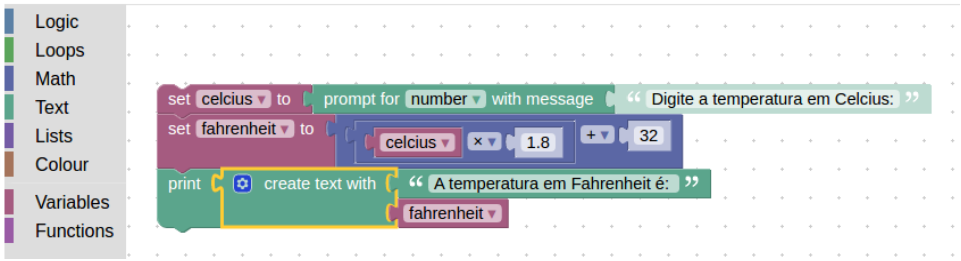
Blockly é uma biblioteca para programação desenvolvida pela Google que adiciona um editor de código visual em aplicativos para a plataforma Web e Android. Semelhante ao *Scratch*, o editor de código do *blockly* também usa peças que se encaixam, como em um quebra-cabeça, para representar todos os elementos básicos de uma linguagem de

programação: ação, sequencia, decisão, iteração e estado. (VIRGINIA POLYTECHNIC INSTITUTE AND STATE UNIVERSITY, 2016)

Blockly é feito de forma que seus blocos de encaixem, mas somente arranjos sintaticamente corretos são permitidos. Esta propriedade liberta o aluno iniciante da preocupação com erros sintáticos, permitindo que ele se concentre na estrutura lógica do código que está tentando criar (FRASER et al., 2013).

O ambiente de programação do Blockly consiste numa área que contém uma caixa de ferramentas, com blocos previamente definidos, e uma área de trabalho. Qualquer bloco da caixa de ferramentas pode ser selecionado e colocado na área de trabalho. A figura 12 apresenta o ambiente de programação do Blockly e, na área de trabalho, o código em Blockly que transforma temperatura medida em celcius para fahrenheit. Note-se que cada um dos blocos tem um entalhe no topo e uma saliência na parte inferior. Esta saliência vai se encaixar no entalhe que fica no topo de outro bloco. Desta forma, os blocos podem ser "empilhados"na sequência em que se quer executar o programa.

Figura 12 – Programa feito em *Blockly* para transformar temperatura medida em Celcius para Fahrenheit



Fonte: Autor.

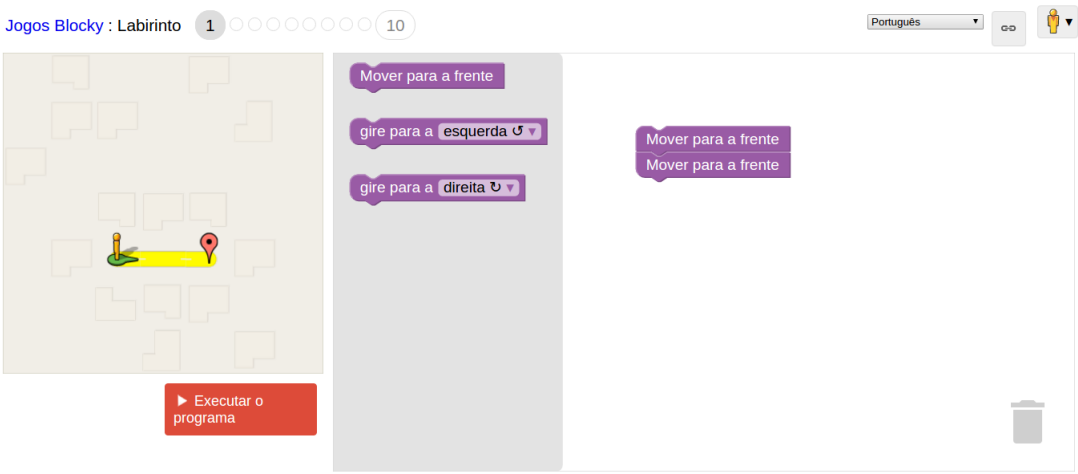
Conforme afirmam Marron, Weiss e Wiener (2012), a principal diferença entre as ferramentas desenvolvidas pela Google e o MIT se encontra na capacidade do Blockly de transformar o código criado através dos blocos em um código funcional escrito em linguagem de programação textual como Javascript. Assim, programas escritos em Blockly podem ser traduzidos e diretamente executados a partir de código padrão de linguagens de programação baseado em texto. Atualmente diversas linguagens são suportadas, sendo as mais populares: Python, Php, Lua e Dart (FRASER et al., 2013).

Outra característica do *Blockly* é a possibilidade de construir blocos personalizados. Uma demonstração desta característica encontra-se no jogo *Blocly Maze* ou Labirinto, que tem como objetivo introduzir os conceitos de ação, sequencia, decisão e iteração (GOOGLE, 2016a).

O Labirinto contém um personagem e um caminho destacado em amarelo que vai do ponto de início, onde se encontra o personagem até ao objetivo sinalizado por um ícone vermelho (ver figura 13). A tarefa do usuário é escrever um código em Blockly que leve

o personagem até ao final do caminho. Observa-se neste ambiente a utilização de blocos personalizados como "*mover para frente*", que movimenta o personagem um passo em frente, e dois blocos do tipo "*gire para a...*", que instrui o personagem a girar 90 graus à esquerda ou direita.

Figura 13 – Nível 1 do programa Maze, feito em *Blockly*.



Fonte: Google (2016b)

2.3 Discussões

Este capítulo apresentou a teoria de desenvolvimento cognitivo proposto por (VYGOTSKY, 2007), denominada Zona de Desenvolvimento Proximal (ZDP). Também foi apresentado um histórico dos ambientes computacionais interativos de aprendizagem, colocando em evidência aqueles ambientes concebidos a partir da abordagem colaborativa de aprendizagem.

Apresentamos os ambientes de instrução assistida por computador (seção 2.2.1), os sistemas tutoriais inteligentes (seção 2.2.2) e os ambientes de aprendizagem colaborativa, procurando dar ênfase aos ambientes de aprendizagem social (seção 2.2.5).

No entanto, para dar prosseguimento ao embasamento teórico, é importante detalhar as ferramentas cognitivas de ajuda a concepção que fazem parte de um ambiente de aprendizagem social: o que são e como tais ferramentas podem ajudar ao aluno a compreender as estratégias para a resolução de problemas de programação. Desta forma, o capítulo seguinte trará o conceito das ferramentas cognitivas utilizadas no contexto deste trabalho: Linguagens de concepção, Planos de programação e Padrões de programação.

3 ESQUEMAS DE CONCEPÇÃO: FERRAMENTAS DE AJUDA A CONCEPÇÃO DE PROGRAMAS

Este capítulo se ocupa em discutir o problema da concepção na aprendizagem de programação, apresentando em seguida algumas ferramentas cognitivas que podem fornecer alguma ajuda no processo de aprendizagem dos conceitos relacionados com solução de problemas utilizando programação de computadores.

3.1 Introdução

Considere um problema de programação qualquer proposto pelo professor. Uma das dificuldades mais comuns pela qual a maioria do alunos iniciantes passam é a "tela em branco": a terrível fase onde o aluno não faz nem ideia de por onde começar a programar.

Tradicionalmente, em uma sala de aula, o professor se encarrega de explicar a construção de um algoritmo, identificando os objetivos que devem ser alcançados e que são extraídos do enunciado do problema. É ainda o professor que deve fornecer o significado de cada linha de código, explicando cada instanciação de variáveis e atribuições de valores em cada ciclo de iteração em estruturas de repetição, entre outras atividades. Todo este conhecimento é comumente transmitido de maneira informal, sem uma documentação apropriada, em aulas expositivas, quer seja utilizando uma ferramenta computacional ou quadro-branco. É fácil inferir que, diante destas circunstâncias, a carga cognitiva exigida do aluno é muito grande, fazendo com que este acabe por não conseguir assimilar todo conteúdo explicado pelo professor.

Lemos, Barros e Lopes (2003) afirmam que, para o aluno aprendiz, existe um abismo entre o enunciado do problema e o respectivo código de uma linguagem de programação que o implementa. Aliás, segundo o mesmo artigo, “há um consenso de que a maior dificuldade nos cursos introdutórios de programação encontra-se na aquisição da habilidade em resolução de problemas dentro do contexto do computador” (LEMOS; BARROS; LOPES, 2003).

Tais dificuldades aqui relatadas apontam para uma possível falha na escolha de uma metodologia de ensino com ferramentas cognitivas adequadas, que sejam capazes de estimular a habilidade do aluno para criar modelos mentais de programação. Esses modelos mentais, conforme veremos nas sessões seguintes, devem reunir todas as tarefas realizadas por um programador para atingir o objetivo de codificar um programa.

Desta forma, a seção 3.2 apresentará os conceitos de modelos mentais de programação, planos de programação e ações primitivas. Em seguida, a seção 3.3 se encarregará de apresentar os Padrões Elementares de Programação como uma ferramenta de concepção para a aquisição da habilidade de construir modelos mentais de programação. Por fim,

a seção 3.4 descreverá o conceito de Linguagem de Concepção de Programas como uma ferramenta de concepção útil para o aprendizado de programação.

3.2 Modelos Mentais de Programação

Esta seção abordará os Modelos Mentais de Programação, e os conceitos de Planos de Programação e Ações Primitivas.

Desde a década de 1980 a ciência cognitiva, e em especial a área de Psicologia da Programação (PPIG, 1987) está preocupada com desenvolvimento de teorias sobre como programadores experientes e aprendizes de programação resolvem problemas. De uma forma geral, muitos trabalhos tem sido realizados com aplicação das teorias da ciência cognitiva.

Sob este prisma, o relatório de Mayrhauser e Vans (1994) indica que mais da metade do esforço realizado para manutenção de programas de computador se encontra na etapa de compreensão do código-fonte. Apesar de seu relatório tratar da manutenção de programas e o esforço realizado para este fim, os autores reúnem em sua pesquisa uma série de modelos cognitivos para compreensão de programas. Segundo os autores, todos os modelos de cognição apresentados possuem elementos em comum. Dentre eles destacam-se os seguintes:

(i) Conhecimento - Há dois tipos de conhecimento que os programadores devem possuir. O conhecimento geral pré-existente, que é independente de um programa específico. Este conhecimento engloba regras de negócios, modelagem de problemas conhecidos, conhecimento prévio de linguagens de programação e ambientes de programação. Quanto mais experiente for o programador, mais conhecimento geral ele terá. Para alunos iniciantes, este conhecimento é ainda pouco adquirido. Outro tipo de conhecimento é denominado por Mayrhauser e Vans (1994) de *Novo Conhecimento*. Trata-se do conhecimento específico aplicado ao programa que se deseja implementar. Vários níveis de abstração podem estar envolvidos com este tipo de conhecimento, desde um olhar estratégico sobre as escolhas de algoritmos adequados até a utilização e compreensão de estruturas de sequência, seleção e repetição na codificação do programa.

(ii) Modelo Mental - segundo Mayrhauser e Vans (1994) é o estado atual de uma representação interna do programa que se pretende implementar. O modelo mental conterá diversos elementos estáticos e dinâmicos, a saber: estruturas de texto (do código-fonte), trechos ou blocos de comandos existentes no código-fonte, planos de programação e hipóteses.

A estrutura do texto de programação faz referência à sintaxe de uma linguagem de programação. Alguns exemplos de unidades de conhecimento sobre a estrutura do texto são:

controles primários (estruturas de sequência, interação e seleção), definição de variáveis, chamada de funções e parâmetros de funções.

Quando várias linhas de código-fonte se agrupam para alcançar algum objetivo, este agrupamento pode ser chamado de bloco de programação ou macro-estrutura. Mayrhauser e Vans (1994) explicam que as macro-estruturas simplesmente nomeiam um bloco de linhas de código. Por exemplo, as micro-estruturas para ordenação de algum elemento consistem em todas as linhas de código de um algoritmo de ordenação. A macro-estrutura neste caso é uma abstração deste bloco de código-fonte e consiste apenas no nome "ordenação". Obviamente, diversas macro-estruturas para ordenação de elementos podem ser criadas, todas elas distinguindo-se em performance, aplicabilidade, quantidade de memória e processamento necessários para executá-las.

A estrutura do texto de programação e as macro-estruturas de comandos do código-fonte são elementos estáticos do modelo mental de programação. Com relação aos elementos dinâmicos do modelo mental, tratam-se de metas construídas pelo programador para a solução de determinado problema e a respectiva construção de *planos de programação* que procuram alcançar as metas para a resolução do problema em questão.

Um fato comumente conhecido entre professores de programação é que, diferentemente dos alunos iniciantes na primeira linguagem de programação, aqueles que já possuem experiência prévia em alguma linguagem de programação sentem menos dificuldades para aprender a segunda linguagem. Há, portanto, um forte indício de que o conhecimento prévio sobre resolução de problemas computacionais, ou seja, a habilidade de construir planos de programação, seja um pré-requisito para o aprendizado de linguagens de programação.

Soloway e outros pesquisadores implementaram vários estudos [(SOLOWAY; EHRLICH, 1984), (BONAR; SOLOWAY, 1985), (JOHNSON; SOLOWAY, 1985)] que mostram que programadores experientes possuem a habilidade de resolver problemas partindo da identificação de metas de programação, seguindo para a criação de planos mentais de programação para, somente depois, transformá-los em algoritmo executável por computador. No processo de resolução de problemas, Lemos, Barros e Lopes (2003) identificaram que há cinco passos que devem ser realizados:

1. Identificar os objetivos ou metas do problema;
2. Selecionar planos de programação que alcancem seus objetivos;
3. Selecionar o conjunto de ações que compõe cada plano;
4. Decompor as ações em sub-planos (Ações primitivas);
5. Codificar cada sub-plano em linguagem de programação.

O algoritmo 1 demonstra, segundo Lemos, Barros e Lopes (2003), de que forma programadores experientes constroem algoritmos para resolver problemas. O algoritmo torna

evidente que existe um caminho cognitivo que deve ser percorrido pelo programador desde o entendimento do problema até a construção de um programa que o solucione.

Algoritmo 1 Como resolver problemas de programação

Entrada: Um problema P
Saída: O código-fonte, escrito em alguma linguagem de programação, que soluciona P

```
1 início
2   identifique o conjunto de metas M que soluciona P
    para cada m_i ∈ M faça
3     selecione o conjunto de Planos de Programação (Pl) que alcança a meta m_i
      para cada pl_j ∈ Pl faça
4       selecione o conjunto de ações abstratas (A) que compõem Pl_j
        para cada a_k ∈ A faça
5         decomponha a_k em ações primitivas (AP)
          codifique cada AP
6       fim
7     fim
8   fim
9 fim
```

3.2.1 Modelo de Compreensão de Programas

Muitos autores propuseram experimentos empíricos para atestar diversos modelos de compreensão de programação, tais como: Brooks (1983), Shneiderman e Mayer (1979), Letovsky (1986) e Soloway e Ehrlich (1984). Todos estes trabalhos definem o modelo mental de programação é construído a partir do processo de compreensão de programas.

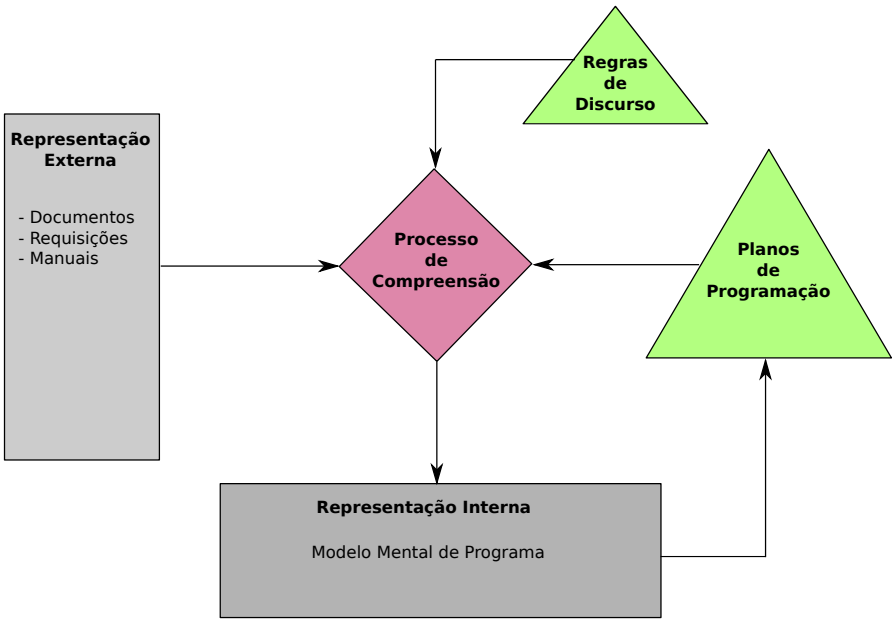
Sob este prisma, no artigo publicado por Soloway, Adelson e Ehrlich (1988), os autores sugerem que os programadores experientes usam dois tipos de conhecimento para programação: Planos de Programação e Regras de Discurso de programação.

Neste trabalho, os autores definiram *plano de programação* como fragmentos de programas que representam estereótipos de sequências de ações. As *regras de discurso de programação*, por sua vez, especificam convenções adotadas na prática de programação e são análogas às regras de discurso utilizadas na conversação. Um exemplo de regra de discurso de programação é a escolha do nome de uma variável, que geralmente segue algum padrão de nomenclatura adotado pelo programador.

Após vários testes, Soloway, Adelson e Ehrlich (1988) concluem que programas de computador são formados por um conjunto de planos de programação que se ajustam a um determinado problema. Esses planos de programação são compostos por regras de discurso de programação que buscam facilitar a leitura e compreensão do código-fonte. Em outras palavras, as regras de discurso de programação tem grande efeito na habilidade de compreensão de programas.

A figura 14 apresenta o modelo de compreensão de programas criado pelos autores. Na figura, os triângulos representam os conhecimentos necessários para programação. Os retângulos simbolizam representações internas e externas do programa. O losango representa o processo de compreensão do programa.

Figura 14 – Modelo de compreensão de programa de Soloway, Adelson e Ehrlich (1988)



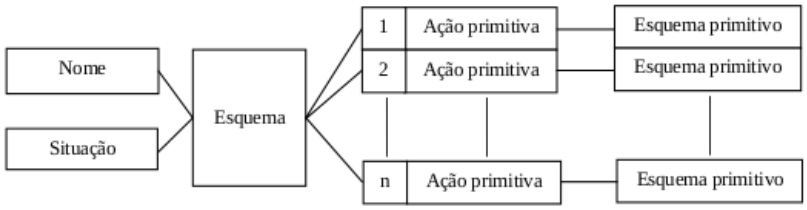
Fonte: Soloway, Adelson e Ehrlich (1988)

3.2.2 Planos de Programação

Lemos, Lopes e Barros (2005) definiram planos de programação como abstrações mentais de um programa, que podem ser traduzidos para uma linguagem de programação. Segundo as autoras, programadores experientes conseguem mapear metas de um problema diretamente em planos de programação. Por outro lado, alunos iniciantes possuem apenas habilidade de criar planos naturais (LEMOS; BARROS; LOPES, 2003). Os planos naturais são diferentes dos planos de programação porque não levam em consideração os aspectos computacionais na solução de um problema. Assim, uma vez que o aprendiz precisa desenvolver a habilidade de mapear planos naturais em planos de programação, seria interessante se, de alguma forma, ele pudesse se apoiar no conhecimento prévio de programadores experientes.

Pensando nisto, Lemos (2004) propõe um modelo de Processo de Construção de Programas para Aprendizes, que considera os planos de programação como elementos-chave para promover a construção de conhecimento cognitivo em programação. Segundo a autora, o modelo consiste em componentes e estratégias para planejamento em programação

Figura 15 – Estrutura genérica de um Plano de Programação.



Fonte: Lemos, Lopes e Barros (2005)

procedimental que visam promover no aprendiz a construção de conhecimento cognitivo em programação.

A Figura 15 apresenta a estrutura genérica que define um plano de programação no modelo apresentado em (LEMOS, 2004) e (LEMOS; LOPES; BARROS, 2005). Segundo os autores, todo plano deve ser identificado por um nome e possuir uma descrição de situação na qual a utilização do plano seria útil. Tais propriedades podem ajudar ao aluno no momento em que ele deve escolher qual plano irá utilizar. Além disso, cada plano está associado a um esquema que representa um bloco de comandos utilizados no plano. O conjunto de *ações primitivas*, que necessariamente é sequenciado, detalha a lógica existente no esquema do plano, subdividindo-o em passos menores. Por último, cada ação primitiva é transformada em esquema primitivo, ou seja, o código-fonte em si.

Como se pretende trazer o conhecimento de programadores experientes para apoiar o desenvolvimento do aprendiz, Lemos (2004) modela os planos de programação a partir de padrões elementares de programação (UNIVERSITY OF NORTHERN IOWA, 2005). Padrões elementares são fragmentos de código que correspondem a certas ações ou operações de alto nível aplicáveis a algum contexto (WALLINGFORD, 1998). O trabalho de Lemos, Barros e Lopes (2003) traz uma contribuição importante para este modelo baseado em padrões elementares, pois apresenta uma biblioteca cognitiva extraída da decomposição dos padrões elementares em ações primitivas. Os padrões elementares criados por Bergin (1999), Astrachan e Wallingford (1998) são usados como base para esta estratificação, conforme podemos ver na **tabela 2**.

A **tabela 3** traça um paralelo entre os padrões sugeridos por Bergin (1999) e planos de programação decompostos em ações primitivas. Por exemplo, o padrão elementar nomeado "Independente de Resposta" da **tabela 2** é utilizado para criar um plano de programação identificado pelo mesmo nome. Em seguida, este plano é decomposto em duas ações primitivas: *i)* insere condição; *ii)* implementa ações para condição verdadeira. Por último, é apresentado um esquema geral que corresponde ao exemplo de código fonte sugerido na tabela 2.

Tabela 2 – Exemplos de Padrões Elementares apresentados em (BERGIN, 1999)

Nome do Padrão Elementar	**Exemplo**	**Contexto**
Independente de resposta (Whether or Not)	`if (measuredHeat() > subBoilThreshold) { shutDownGen(); }`	Você está em uma situação em que uma ação pode ser apropriada ou não dependendo de alguma condição testável.
ação alternativa (alternative-action)	`if (numericGrade > 60){ output("passing"); } else{ output("failing"); }`	Você está em uma situação em que somente uma de duas ações é apropriada dependendo de uma condição. Quando a condição é verdadeira deseja-se executar uma condição, quando falsa deseja-se executar uma outra ação.

3.3 Padrões elementares de programação

Os padrões elementares de programação, também chamados de padrões pedagógicos de programação (PORTER; CALDER, 2004), definem um arcabouço no qual um determinado problema será resolvido. Isto é particularmente útil para o iniciante em programação de computadores, visto que estes padrões de programação formam uma base inicial de micro soluções que podem ser reaproveitadas, adaptadas e combinadas pelos alunos durante a elaboração de seus programas. A resolução de problemas utilizando experiências prévias, procurando adaptar as soluções anteriores para resolver os novos problemas, é discutida por Johnson e Soloway (1985), Mayrhauser e Vans (1994), entre outros.

Segundo Bergin (1998), existem dois movimentos na área de ciência da computação relacionados a padrões, com objetivos diferentes. O primeiro envolve a comunidade de Padrões de Projeto que se reúnem em conferências anuais denominadas PLoP (*Pattern Languages of Programming*). O segundo movimento está envolvido com padrões pedagógicos e enfatizam o padrão de instrução para o ensino de programação. Bergin (1998) afirma que, enquanto o primeiro movimento diz "o que" deve ser ensinado, o segundo movimento explica "como" deve ser ensinado.

Foi desta forma que, na conferência de ChiliPLoP em 1998 (THE HILLSIDE GROUP, 1998), um pequeno grupo de educadores da área de ciência da computação se reuniu para pensar e escrever padrões que fossem adequados para iniciantes aprenderem a programar já nos dois primeiros anos de sua graduação (ASTRACHAN; WALLINGFORD, 1998).

Tabela 3 – Exemplos de Planos de Programação e Ações primitivas criados a partir dos padrões de seleção de Bergin (1999) (LEMOS, 2004)

Nome do Plano de Programação	**Ações primitivas**	**Esquema**

Independente de resposta	Insere condição
	
	Implementa ação(ões) para condição verdadeira
		`if ($condicao){`
		
		`$acao;`
		
		`}`

Ação alternativa	Insere condição
	
	Implementa ação(ões) para condição verdadeira
	
	Implementa ação(ões) para condição falsa
		`if ($condicao){`
		
		`$acao;`
		
		`else{`
		`$acao;`
		`}`

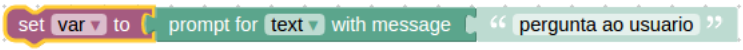
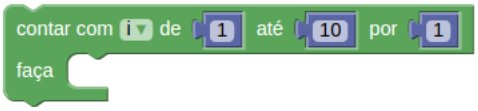
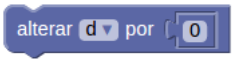
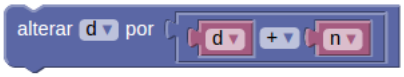
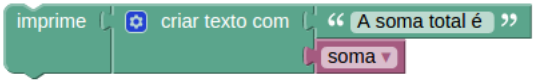
Diversos trabalhos foram produzidos a partir deste encontro. (BERGIN, 1999) apresenta uma linguagem de padrão simples para ser usado na escrita de código que requer a seleção de ações alternativas. (ASTRACHAN; WALLINGFORD, 1998) relaciona um grupo de padrões que auxiliam na identificação de problemas específicos que os estudantes encontram quando aprendem a escrever laços de repetição.

Podemos definir padrões elementares de programação como soluções para problemas computacionais recorrentes, formando blocos de trechos de código para o desenvolvimento de algoritmos (LEAL; FERREIRA, 2013). Astrachan e Wallingford (1998) define padrões elementares como ferramentas de ensino. Já Bergin (1998) afirma que, no âmbito educacional, os padrões podem prover uma forma de um professor mostrar ao aprendiz como resolver problemas em um determinado contexto.

Existe uma relação entre os padrões elementares de programação e as ações primitivas, apresentadas na sessão 3.2. As ações primitivas dizem respeito a menor parte de que é composto o plano de programação. Desta forma, elas fazem parte do modelo mental do programa. Os padrões elementares, por sua vez, fazem parte do modelo implementado

do programa. Trata-se então de fragmentos de programação que mapeiam as ações de programação. A tabela 4 relaciona a abstração (ação primitiva) e o exemplo de código fonte, já utilizando linguagem de programação visual (ver seção 2.2.6).

Tabela 4 – Ações primitivas e padrões elementares

Ações Primitivas	**Padrões elementares**
implementar leitura do teclado	
emitir mensagem para o usuário	
gerar sequência numérica	
inicializar variável para soma iterativa	
definir expressão da soma iterativa	
mostrar resultado na tela com texto concatenado	

3.3.1 Padrões de Programação utilizados neste trabalho

Os padrões de programação utilizados neste trabalho foram inspirados por diversos trabalhos. No entanto, para os padrões de seleção e padrões de repetição, destacam-se os trabalhos realizados por Bergin((BERGIN, 1999), (BERGIN, 1998)) e Astrachan e Wallingford (1998), respectivamente. As subseções que se seguem apresentam os padrões utilizados neste trabalho que se encontram classificados em: padrões entrada e saída, padrões de seleção, repetição e acumulação.

Apresenta-se o nome do padrão, sua definição, um exemplo e um pequeno trecho de código utilizando *Blockly*, linguagem visual de programação apresentada em 2.2.6 e utilizada no protótipo de um Ambiente de Aprendizagem Social para programação (capítulo 6), como exemplo relacionado ao padrão.

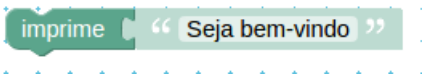
3.3.1.1 Padrões de Entrada e Saída

Padrões de entrada e saída são importantes pois muitas vezes os alunos sentem dificuldade em compreender como realizar as etapas de entrada e saída dos dados. Os padrões

de Entrada e Saída utilizados neste trabalho foram inspirados nos padrões elementares para linguagem C catalogados em (PAES, 2016) e seguem descriminados abaixo.

- ❏ Saída simples - Escreve uma frase simples quando existe a necessidade de informar algo ao usuário. No exemplo da figura 16, a mensagem "Seja bem-vindo"é emitida para o usuário.

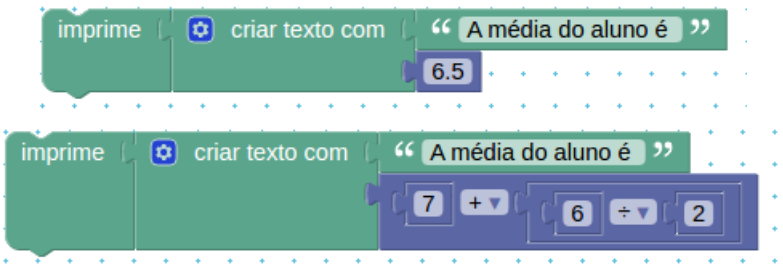
Figura 16 – Padrão de Saída Simples



Fonte: Autor

- ❏ Saída combinada - Escreve uma frase que contenha um texto e uma expressão. A expressão pode ser um valor simples (inteiro ou real), uma variável ou uma expressão matemática. a figura 17 demonstra duas situações de uso deste padrão. Na primeira situação, o texto é combinado com um valor real. Na segunda situação, há uma expressão que será calculada pelo computador antes de imprimir a mensagem.

Figura 17 – Padrão de Saída Combinada



Fonte: Autor

- ❏ Entrada de dados pelo teclado - Esta é a forma mais comum de entrada de dados em disciplinas introdutórias a programação. Na biblioteca Blockly a entrada de dados e a guarda do valor em variável ocorre em um único comando, conforme demonstrado na figura 18. Neste exemplo a variável A recebe o valor digitado pelo usuário a partir do teclado. Note que no mesmo comando, uma saída de texto é emitida, solicitando do usuário a digitação do valor correspondente.

Figura 18 – Padrão de Entrada de dados a partir do teclado

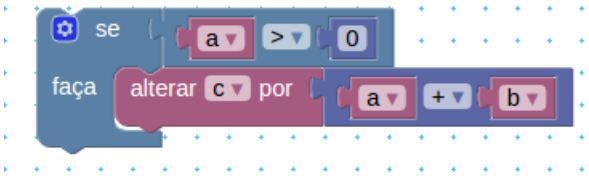
Fonte: Autor

3.3.1.2 Padrões de Seleção

Padrões de seleção permite ao aluno estudar as diversas formas de se utilizar uma estrutura de decisão em um programa. Esses padrões estão relacionados com tomadas de decisão para resolução de problemas dentro de um programa (LEAL; FERREIRA, 2013). Os quatro padrões de seleção listados abaixo também estão descritos no trabalho de Leal e Ferreira (2013).

- Escolha Única - Este padrão de seleção é utilizado quando existe a necessidade de testar se uma condição é apropriada dependendo da condição de teste. No exemplo abaixo, o comando c = a + b só será executado se a condição de teste for verdadeira, caso contrário o programa continua sua execução. A figura 19 mostra um exemplo do padrão escolha simples.

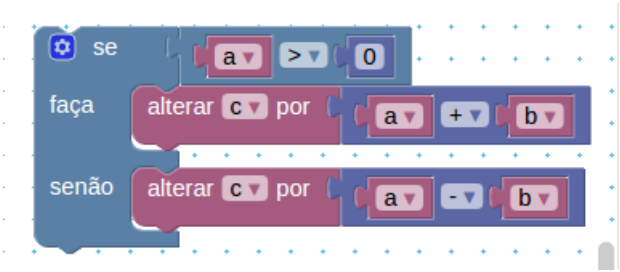
Figura 19 – Padrão de Seleção Escolha Única



Fonte: Autor

- Escolha alternativa - Este padrão é utilizado quando se tem exatamente duas ações possíveis, porém apenas uma é apropriada dependendo da condição testada. Ao contrário do padrão **escolha única**, este padrão contém uma segunda parte, que é executada caso o primeiro teste seja falso. No exemplo abaixo, caso a variável `a` seja maior que 0, será executado o comando `FAÇA`, que é `c = a + b`. Caso contrário, será executado o comando dentro do bloco `SENÃO`, que é `c = a - b`. A figura 20 mostra um exemplo do padrão escolha alternativa.

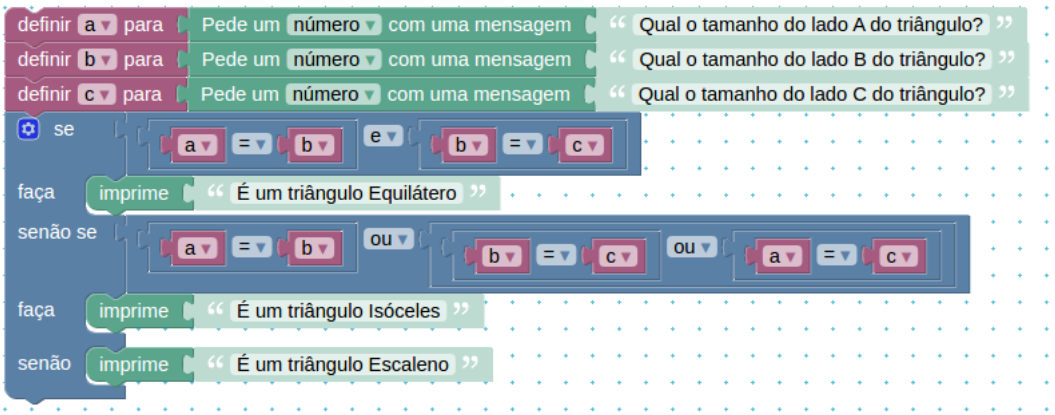
Figura 20 – Padrão de Seleção Escolha Alternativa



Fonte: Autor

- Escolha sequencial - O padrão escolha sequencial é utilizado quando existem mais de duas condições possíveis de teste, desde que haja a necessidade de se fazer testes relacionais entre uma ou mais variáveis e um valor, ou entre uma ou mais variáveis com outras variáveis. A figura 21 mostra um exemplo do uso do padrão escolha sequencial para decisão do tipo de triângulo quanto à igualdade dos lados. No exemplo, dada a entrada dos tamanhos dos lados de um triângulo (variáveis a,b e c), mais que dois testes devem ser feitos com as variáveis. Outro ponto importante é que uma exceção aos testes é necessária. Na figura, esta exceção é representada pelas ações do bloco `SENÃO`.

Figura 21 – Padrão de Seleção Escolha Sequencial



Fonte: Autor

3.3.1.3 Padrões de Repetição

- Meio laço - O padrão meio laço é utilizado quando o programador precisa que alguns comandos sejam repetidos enquanto uma determinada condição se mantiver verdadeira. Este padrão utiliza uma variável de teste para controlar o laço de repetições.

Ao utilizar o padrão de meio laço há sempre a possibilidade do programa entrar em *loop infinito*, um termo em programação usado para definir a condição na qual um programa nunca para de executar. Por este motivo, a condição de parada da repetição deve ser bem especificada.

Segundo Clancy e Linn (1999), este padrão é de difícil compreensão para alunos iniciantes. Na linguagem visual definida pelo Blockly, o padrão de meio laço divide-se em dois tipos:

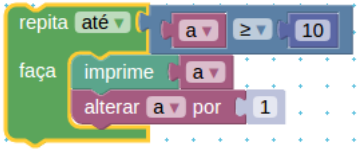
– **Repita enquanto** - repete um ou vários comandos enquanto uma condição for verdade;

Figura 22 – Padrão de Seleção Meio Laço (com *enquanto*)

Fonte: Autor

– **Repita até** - Repete um ou vários comandos enquanto uma condição for falsa;

Figura 23 – Padrão de Seleção Meio Laço (com *até*)



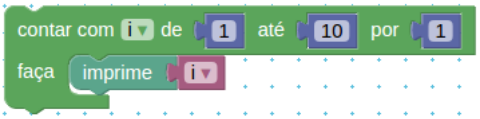
Fonte: Autor

❑ Laço Contado - Este é o mesmo padrão que Astrachan e Wallingford (1998) denominaram de *Process all itens*. O *Laço Contado* deve ser utilizado quando se conhece previamente o número de vezes que será executado um certo conjunto de instruções. Por isto ele é dotado de uma variável de controle que será responsável por contar o número de repetições. A execução do laço é interrompida quando a variável de controle alcança o valor máximo do laço.

Adicionalmente, muitas linguagens de programação, como C/C++, Java, Javascript, etc, implementam um comando complementar para o *laço contado*, denominado *break*, que é capaz de interromper o laço a qualquer momento de sua execução. O Blockly também conta com tal artifício, porém no contexto deste trabalho não trataremos deste comando de interrupção.

A figura 24 apresenta um exemplo de utilização deste padrão. No exemplo a variável *i* assume os valor do número inicial (1) e vai incrementando de acordo com o intervalo especificado (1) até alcançar o número final (10).

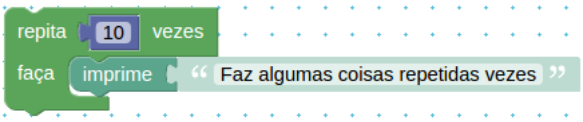
Figura 24 – Padrão Laço Contado (versão com valor de início, valor de fim e intervalo de incremento)



Fonte: Autor.

O Blockly ainda implementa uma versão mais simples do laço contado, conforme ilustra a figura 25, onde o valor inicial e o intervalo de incremento são dados como constantes e igual a 1, ficando desta forma ocultos da interface visual do bloco de repetição.

Figura 25 – Padrão Laço Contado (versão apenas com valor final)



Fonte: Autor

3.3.1.4 Padrões de Acumulação

3.4 Linguagem de Concepção de Programas

Winograd (1996) define linguagem de concepção como uma linguagem visual e funcional de comunicação entre pessoas que usam um artefato. Rheinfrank e Evenson (1996), por sua vez, afirmam que linguagens de concepção estão inseridas em diversos produtos, construções, cidades, etc. Segundo estes autores, a linguagem de concepção que está inserida em um determinado objeto, em geral, não é percebida conscientemente por seu utilizador. Contudo, se implementada de maneira coerente, a linguagem de concepção pode melhorar a interação entre usuário e objeto.

John Rheinfrank estudou profundamente o tema quando participou da equipe de design de máquinas fotocopiadoras da Xerox (RHEINFRANK; EVENSON, 1996). Ele pensava na máquina fotocopiadora como um meio intermediário pelo qual seu projetista pode se comunicar com o usuário. Neste período, Rheinfrank estava particularmente interessado em entender como as pessoas usam máquinas em seu ambientes reais.

Sob este prisma, pode-se definir linguagem de concepção como uma linguagem que é utilizada através de artefatos para comunicar ao usuário alguma informação. A linguagem de concepção, portanto, envolve sistemas de elementos e relacionamento entre esses elementos (WINOGRAD, 1996). É aplicada em um determinado objeto e carrega consigo um conjunto de questões sobre o próprio objeto. Segundo Rheinfrank e Evenson (1996), linguagem de concepção consiste em:

- **Coleção de elementos**. Ex.: forma, textura, cores, ações, metáforas;
- **Conjunto de princípios organizacionais**. É a descrição de como os elementos podem ser compostos para que possam construir coisas que façam sentido;
- **Coleção de situações de qualificação**. São exemplos de como elementos e princípios de composição podem mudar baseado no contexto.

Paraguaçu (1997), explica que há uma distinção entre linguagem natural e linguagem de concepção. Segundo ele, a linguagem natural consiste no uso de palavras e regras gramaticais. Já a linguagem de concepção consiste no uso de elementos de concepção e dos princípios de composição. Segundo o próprio autor, "a linguagem de concepção é utilizada para projetar objetos, especificando o que eles são, o que fazem, sua utilização e sua contribuição à experiência".

No contexto da aprendizagem, a linguagem de concepção pode ser usada como base para a construção de ferramentas que aumentem as capacidades cognitivas do aprendiz. Quando disponibilizadas em uma sala de aula, essas ferramentas, ajudam tanto na compreensão de como utilizar os objetos de aprendizagem quanto na integração desses objetos às atividades dos aprendizes (COSTA; PARAGUAÇÚ; PINTO, 2009).

Formalmente, a linguagem de concepção (LC) foi definida por Paraguaçu (1997) a partir da união de duas sub-linguagens: L1 e L2, de forma que a LC é a união dessas duas linguagens ($LC{=}L1 \cup L2$). Além das sub-linguagens L1 e L2, a linguagem de concepção também está associada a uma metalinguagem de manipulação (LM) que pode ser definida como um conjunto de questões. Tais questões estão associadas tanto à sub-linguagem L1, como a cada bloco de conhecimento, dentro de L2, conforme será explicado na subseção 3.4.2.

Para aplicar a definição de linguagem de concepção no contexto de linguagens de programação, (PARAGUAÇU, 1997) procura definir quais os objetos de L1 e L2 que compõem os elementos de uma linguagem de programação. As próximas subseções detalharão as sub-linguagens L1 e L2 para o contexto de Linguagem de Concepção para Programação (LCP).

3.4.1 Sub-linguagem L1 da Linguagem de Concepção para Programação

A sub-linguagem L1 é o conjunto de todos os objetos de um alfabeto nomeado como A. Este alfabeto, por sua vez, é composto por um conjunto de verbos (V) e por um conjunto de argumentos (Arg) de forma que cada objeto obedece à uma regra de montagem definida que combina verbos com argumentos.

Para ilustrar a sub-linguagem L1 no contexto de programação, considere, por exemplo, a operação de atribuição. Esta operação permite ao programador definir ou redefinir o valor que será armazenado em determinada variável. Independente da linguagem de programação utilizada, este comando sempre obedecerá a seguinte regra de montagem:

```
<variavel> <op> <expressão>
```

Onde <op> representa o operador de atribuição que pode variar sua representação dependendo da linguagem de programação. Duas formas típicas de representação deste operador são: "="e ":=". Dentro de determinada linguagem de programação a forma de representar o operador de atribuição é imutável e, desta forma <op> é um elemento do conjunto de verbos dentro de L1. Por sua vez, <variavel> e <expressão> são elementos do conjunto de argumentos de L1 que podem variar dependendo da situação. Em termos da linguagem L1, uma operação de atribuição pode ser definida como:

```
<argumento> <verbo> <argumento>
```

Conclui-se então que o conjunto de verbos em L1 são palavras chave, ou ações do programador. Além disso uma regra de montagem de L1 associa verbos a argumentos e gera uma **linha de programa** em uma determinada linguagem de programação (PARAGUAÇU, 1997). As ações primitivas da tabela 4 são exemplos de objetos de L1 e os argumentos para cada objeto dentro de L1 são as informações que variam em cada situação. Por exemplo, na ação primitiva "implementar leitura do teclado", o argumento poderia ser o nome da variável que armazena o valor ou a pergunta que será dirigida ao usuário do programa.

3.4.2 Sub-linguagem L2 da Linguagem de Concepção para Programação

A sub-linguagem L2 é formada por blocos de conhecimento (Blc). Cada bloco, por sua vez, pode ser representado por um conjunto de objetos de L1, ou seja, por um conjunto de linhas de programa. Assim, pode-se concluir que um elemento do conjunto Blc representa um **trecho do programa** em uma determinada linguagem de programação.

Desta forma, os blocos de conhecimento podem ser entendidos como objetos que representam as unidades de conhecimento, ou seja, conceitos dentro do domínio de programação. Sendo assim, é lógico associar os objetos de L2 a planos de programação, uma

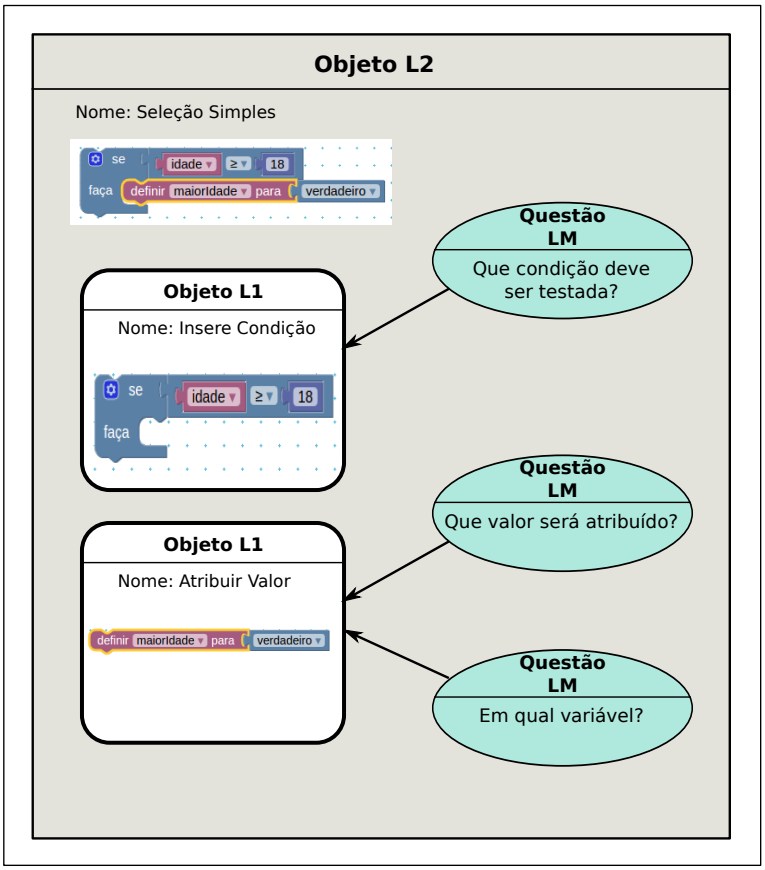
vez que estes são constituídos por um conjunto de ações primitivas e definem conceitos do domínio de uma linguagem de programação.

3.4.3 Metalinguagem de manipulação (LM)

Segundo Paraguaçu (1997), uma metalinguagem de manipulação é associada a uma linguagem de concepção de programas a partir de um conjunto de questões que estão vinculadas aos verbos (dentro de L1) e às linhas de programação (na linguagem L2). O propósito desta metalinguagem é auxiliar o aluno a construir seus primeiros programas utilizando como suporte as questões que estão associadas a cada linha de programação que deve ser implementada pelo aluno.

A figura 26 ilustra exemplos de objetos das sub-linguagens L1 e L2 para uma linguagem de concepção de programas visuais. Como se trata de uma linguagem de programação visual, seus objetos são dispostos espacialmente tanto para direita quanto para baixo, não fazendo sentido falar em linha de programa neste contexto. Na imagem observa-se também questões da metalinguagem de manipulação associadas aos objetos da linguagem L1.

Figura 26 – Representação de objetos das sub-linguagens L1 e L2 de uma Linguagem de Concepção de programas Visuais



Fonte: Autor

3.5 Discussões

Este capítulo apresentou algumas ferramentas capazes de ajudar o aluno a criar esquemas de concepção, levando-o a compreender quais são as estratégias para a resolução de problemas de programação. O modelo mental do programa é uma representação internalizada de um programa que pode ser entendido como um conjunto de metas. Essas metas subdividem o problema e são alcançadas a partir da implementação de planos de programação.

Estudos desenvolvidos por Soloway e Ehrlich (1984), Bonar e Soloway (1985) e Lemos, Barros e Lopes (2003) afirmam que as tarefas para resolver problemas de programação incluem as habilidades de identificação das metas e construção de planos de programação que alcancem essas metas.

Uma vez apresentadas estas ferramentas cognitivas capazes de auxiliar o aluno na aprendizagem de programação, precisamos elaborar um método pedagógico para aplicar tais ferramentas em uma sala de aula. Desta forma, o próximo capítulo apresentará um modelo de ensino de programação, elaborado a partir da abordagem cognitiva, que trabalhará os conceitos de programação utilizando o conjunto de ferramentas detalhados neste capítulo.

4 MÉTODO DE ENSINO PROPOSTO

O objetivo deste capítulo é apresentar um método de ensino para programação utilizando-se como base os fundamentos da abordagem cognitiva de aprendizagem. Uma visão geral desta proposta pedagógica é apresentada na seção 4.1. Logo em seguida serão apresentados, na seção 4.2, os participantes deste método e seus respectivos papéis. Na seção 4.3 são apresentadas as etapas para a implementação do método. Em seguida, na seção 4.4, é descrito o fluxo de trabalho desta abordagem cognitiva, seguido de uma lista de conceitos básicos que devem ser apresentados aos alunos, na seção 4.5.

4.1 Visão geral

Em estudos anteriores (LEMOS; LOPES; BARROS, 2005),(LEMOS, 2004),(LEMOS; BARROS; LOPES, 2003), foi proposta uma abordagem cognitiva para ensino de programação, baseada nos conceitos de padrões elementares de programação e planos de programação (ver seções 3.3 e 3.2). Os elementos-chave desta abordagem são os planos de programação que podem representar soluções para problemas de programação (LEMOS; LOPES; BARROS, 2005). Desta forma, para resolver um problema, os alunos devem manipular estratégias que visam o planejamento de suas ações na prática de programação, com o objetivo de construir conhecimento cognitivo em programação.

Em um método para aprendizagem cognitiva, os alunos são encorajados a utilizar as ferramentas cognitivas que estão disponíveis, como, por exemplo: os planos de programação e os padrões elementares de programação. Tais ferramentas devem ser previamente conhecidas pelo discente.

Por outro lado, o professor deve fornecer ao aluno não apenas o enunciado do problema, mas também o **modelo do problema** que descreve uma solução do problema proposto através de **metas** a serem alcançadas. Cada meta deve ser realizada por um conjunto de **planos de programação** que, por sua vez, é composto de diversas **ações primitivas de programação**. Para facilitar a interação do aluno com cada um desses elementos, uma **linguagem de concepção de programa** deve ser utilizada para associar questões auto-reflexivas aos planos e às ações primitivas que definem o modelo do problema.

4.2 Participantes e papéis

Os participantes da abordagem cognitiva de ensino são o professor e os alunos. Cada aluno pode assumir dois papéis distintos: um aluno pode ser, em um determinado momento, aquele que realiza a tarefa e, em outro, ele pode ser um companheiro que colabora com o colega na resolução de um problema. Adicionalmente, em um ambiente virtual de aprendizagem social, é possível incluir um ou mais agentes artificiais, também assumindo

papéis distintos: ora como agente companheiro do aluno, ora como agente assistente do professor.

4.3 Apresentação das etapas da metodologia proposta

A presente abordagem de ensino pode ser divida nas seguintes etapas:

1. Planejamento
2. Preparação
3. Execução e socialização
4. Avaliação

4.3.1 Fase de planejamento

Na fase de planejamento são definidos todos os conteúdos que serão apresentados aos alunos, procurando selecionar um conjunto de padrões elementares que atenda todas as necessidades para o trabalho em sala de aula. Além disso, deverá ser decidido se a abordagem de ensino será realizada com a utilização de um ambiente computacional para suporte da aprendizagem. Se for este o caso, será necessário tomar todas as medidas para a utilização deste ambiente, realizando ações como, por exemplo, o cadastro de todos os alunos neste ambiente.

A escolha dos problemas é outra importante atividade realizada na fase de planejamento, pois será necessário elaborar, juntamente com cada problema, um modelo mental de programação com suas metas, planos de programação e ações primitivas, de acordo com o que explanado no capítulo 3. Desta forma, o professor deve escolher com bastante atenção todos os problemas que serão trabalhados em sala de aula, para que seja possível abordar todo conteúdo planejado.

Por fim, a escolha do canal de comunicação entre os alunos e o professor é igualmente importante. A partir do meio de comunicação escolhido, os alunos poderão compartilhar seus projetos, possibilitando a cada aluno fornecer ou receber críticas e sugestões de melhoria de outros colegas. O meio de comunicação também será utilizado para o envio dos projetos resolvidos ao professor para correção e *feedback*. Portanto, o meio de comunicação deve ser de fácil acesso, rápida aprendizagem de uso e que permita o compartilhamento de código-fonte de forma simples.

4.3.2 Fase de preparação

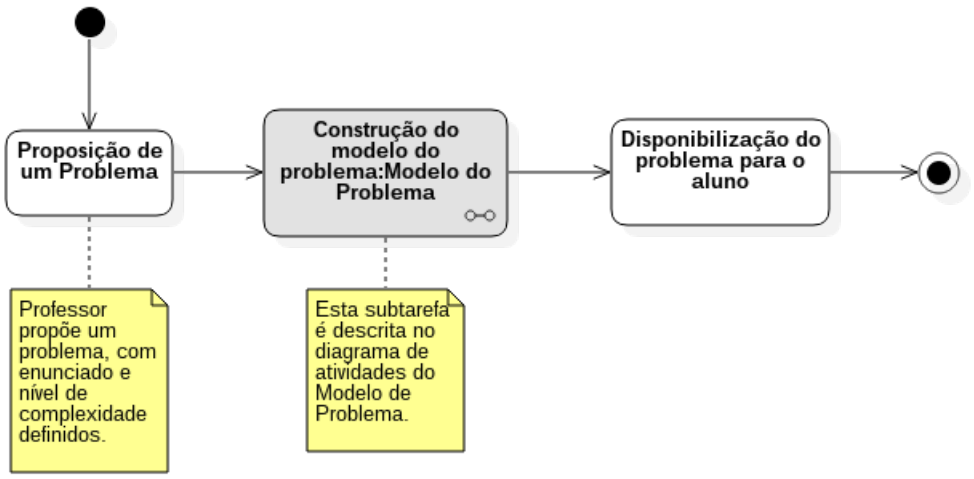
Nesta fase o professor deve preparar todo o conteúdo que será apresentado aos alunos. Primeiramente, o conjunto de padrões elementares que serão utilizados em sala de aula deve estar devidamente organizado, podendo ser entregue ao aluno como um material de

apoio. Também todos os problemas que serão utilizados em sala de aula devem estar organizados e ordenados conforme está descrito na seção 4.5.

A proposição de um problema é uma atividade muito importante na fase de preparação. A figura 27 apresenta a sequencia funcional para proposição de um problema.

Inicialmente o professor deve propor um problema para ser resolvido computacionalmente, fornecendo todas as informações necessárias para sua implementação, tais como enunciado e nível de complexidade. Cada problema proposto pelo professor deve ser acompanhado por um modelo do problema. O professor então disponibiliza este problema juntamente com as metas que devem ser alcançadas.

Figura 27 – Diagrama de Atividades para Proposição de um Problema



Fonte: Autor

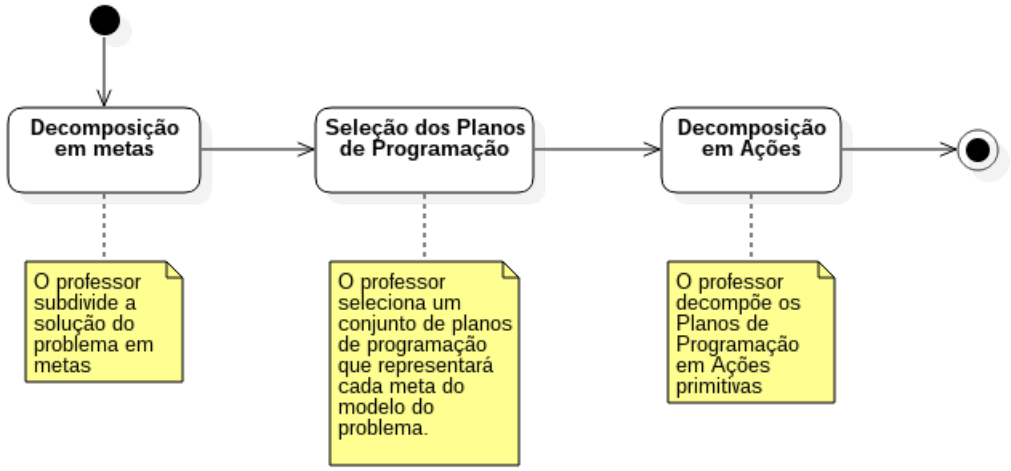
O modelo do problema decompõe o problema proposto em metas que, se atingidas, resolvem o problema. Cada meta é realizada por um ou mais planos de programação. Os planos de programação devem ser conhecidos dos alunos e facilmente transcritos em forma de padrões elementares de programação. A figura 28 apresenta os passos descritos neste parágrafo.

Para resolver um problema proposto, cada aluno deve se utilizar dos conhecimentos previamente adquiridos sobre a linguagem de programação utilizada, bem como do conhecimento sobre padrões elementares. No contexto da aprendizagem social, é salutar a troca de experiências e a colaboração tanto entre alunos quanto entre aluno e professor.

4.3.3 Fase de execução e socialização

Na fase de execução, o professor deve apresentar ao aluno todas as ferramentas disponíveis para o aprendizado de programação. Desta forma, o aluno terá oportunidade de

Figura 28 – Diagrama de Atividades para construção de um modelo de Problema



Fonte: Autor

conhecer os padrões elementares de programação. Conhecimento sobre planos e metas de programação também podem ser abordados.

Como esta proposta de método de ensino tem como base o conceito de aprendizagem cognitiva (ver seção 2.2.4), ela está apoiada na realização constante de atividades que são executadas dentro de uma sequencia crescente de complexidade. A seção 4.4 apresenta em detalhes todo o fluxo de trabalho que envolve os alunos e o professor na resolução dos problemas propostos.

4.3.4 Fase de Avaliação

Na última fase, o professor deverá avaliar os resultados obtidos pelos alunos, procurando observar se os problemas até então resolvidos foram suficientes para o desenvolvimento de cada aluno em determinada etapa do aprendizado. Uma vez que o professor considere que os resultados foram satisfatórios, ele pode iniciar um novo ciclo da metodologia para abordar um novo conteúdo a ser trabalhado em sala de aula.

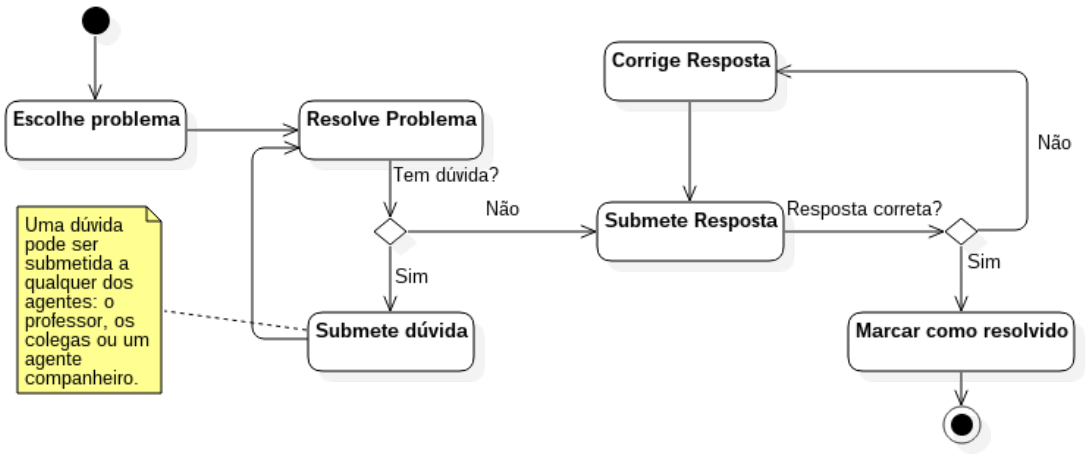
4.4 Fluxo de trabalho

Dentro da abordagem cognitiva, o aluno precisará executar um conjunto de atividades para que ele possa completar todo fluxo de trabalho da abordagem. Estas atividades abrangem desde a escolha do problema que será resolvido até a aceitação de sua resposta pelo professor.

A figura 29 ilustra as atividades realizadas pelos discentes e pelo professor e as subseções seguintes irão abordar todas as ações apresentadas nesta ilustração, detalhando cada

atividade desta metodologia.

Figura 29 – Fluxo de atividades na abordagem cognitiva



Fonte: Autor

4.4.1 Passo 1: Aluno escolhe um problema proposto pelo professor

No início do processo, o aluno deve receber do professor uma série de problemas para resolver. Tais problemas devem ser preparados pelo professor antecipadamente, conforme explanado na seção 4.2, seguindo uma ordem em que o nível de complexidade aumente gradativamente. Depois de escolhido o problema que será resolvido, o aluno deverá observar as metas para a resolução do problema, escolhendo uma para o início do processo.

4.4.2 Passo 2: Resolução do problema pelo aluno

Após a escolha do problema e da meta que será alcançada, o aluno tentará resolver o problema. Neste momento, é importante que o aluno tenha à sua disposição todo conteúdo da disciplina de maneira clara e organizada. O aluno poderá, a qualquer instante, alterar a meta. O aluno, portanto, ajusta o processo de resolução do problema ao seu próprio entendimento e raciocínio, refinando gradativamente sua própria habilidade de desenvolver algoritmos.

4.4.3 Passo 3: Aluno submete dúvidas

O espaço para resolução de problemas é também um espaço de mediação e interações sociais. Portanto, quando necessário, o aluno poderá pedir auxílio ao professor ou até mesmo um colega em nível mais avançado. Neste momento, o professor pode trazer

exemplos ou até questionamentos, procurando com isso revelar ao aluno quais planos de programação são necessários para alcançar uma determinada meta do problema.

4.4.4 Passo 4: Aluno submete resposta

Uma vez finalizado o processo de resolução do problema, o aluno poderá submeter sua resposta ao professor. Este deve analisar a resposta do aluno, propondo melhorias quando for o caso.

4.4.5 Passo 5: Aluno corrige sua resposta após comentários do professor

Esta é uma etapa de refinamento da resposta. O aluno pode ouvir os comentários do professor, suas sugestões ou até exemplos de melhorias no código-fonte da solução. Professor e aluno, então, entram em um processo de negociação até que o resultado seja satisfatório.

4.4.6 Passo 6: Professor aceita a resposta do aluno

Quando a solução implementada pelo aluno encontra-se em um estado de refinamento aceitável pelo professor e todas as dúvidas do aluno sobre o problema estejam suprimidas, o professor poderá aceitar a resposta do aluno, finalizando o processo de resolução do problema.

4.5 Conceitos básicos apresentado aos alunos

De acordo com Collins (2006), a aprendizagem cognitiva está apoiada na realização de tarefas sequenciadas, cuja complexidade é ampliada gradativamente. Tal princípio requer do professor um planejamento para que o conteúdo e as atividades sejam organizadas em níveis diferentes de complexidade.

No contexto de aprendizagem de programação, adota-se aqui a mesma divisão do conteúdo em níveis utilizado por Rocha et al. (2010), conforme listado na tabela 5. No entanto, algumas alterações foram realizadas para que a organização do conteúdo refletisse a necessidade da inclusão dos padrões de programação no contexto de linguagem de programação visual.

4.6 Discussões

Neste capítulo foi apresentado uma abordagem para o ensino de programação para iniciantes que utiliza planos de programação como ferramenta cognitiva para ajudar o aluno na compreensão e na atividade de criação de algoritmos. O ciclo de atividades associado ao método foi apresentado, descrevendo os papéis e as tarefas realizadas por todos os agentes.

Tabela 5 – Relação dos níveis de complexidade para estudo de programação usando linguagem de programação visual

Níveis	**Conteúdo**	**Padrões Elementares**
Nível 01	Conceitos básicos de Programação e da linguagem de programação visual	Ler teclado, Saída simples, Saída combinada, Atribuir valor, atribuir variável, atribuir expressão
Nível 02	Estrutura de Seleção: casos elementares	Escolha única, Escolha alternativa, escolha não relacionada
Nível 03	Estruturas de Seleção aninhadas	Escolha Sequencial, Escolha Aninhada, Série de Possibilidades
Nível 04	Estrutura repetição para laço contado	Laço contado com passo, Laço contado simples (ver seção 3.3)
Nível 05	Estruturas de repetição para "laços com fim indeterminado"	Meio laço - Repita enquanto, Meio laço - Repita até
Nível 06	Trabalhando com coleções de dados homogêneos	Laço para percorrer lista, Laço para buscas em lista

Também foi apresentado neste capítulo o fluxo de trabalho da presente no método de ensino, desde o momento em que o aluno escolhe um problema proposto pelo professor até o momento em que o professor aceita a resposta enviada pelo aluno.

Outro ponto importante da método apresentado é a preparação do conteúdo que deve ser ministrado ao aluno, classificando-o conforme seu nível de dificuldade e elaborando atividades-problema para que o aluno possa desenvolver seus estudos.

Para tanto, duas coisas são importantes para o sucesso na aplicação do método: o conhecimento prévio do aluno sobre os padrões elementares de programação e a elaboração de planos mentais para solução dos problemas propostos, utilizando-se planos de programação baseado nos padrões elementares de programação.

No próximo capítulo será apresentado a arquitetura de um ambiente de aprendizagem social projetado para facilitar tanto a vida do professor, no momento da criação das atividades e seus esquemas de concepção, quanto a vida do aluno, permitindo que ele possa interagir com seus colegas, professor e o agente companheiro artificial enquanto elabora uma solução para o problema escolhido.

5 ARQUITETURA DO AMBIENTE PROPOSTO

Conforme Paraguaçu (1997), a aprendizagem de programação é um processo que envolve ações coordenadas de vários agentes. Estes agentes, por sua vez, utilizam um conjunto de ferramentas cognitivas para construir um programa.

Considerando a definição de ambiente de aprendizagem social (AAS), apresentado na seção 2.2.5, este capítulo apresentará um modelo de ambiente virtual de aprendizagem social. A ideia é que tal ambiente possa facilitar a utilização de uma metodologia de ensino com base nos conceitos da aprendizagem cognitiva, conforme discutido no capítulo 4, e com o objetivo de auxiliar os alunos iniciantes em programação.

O ambiente proposto - denominado *DEKSTRA* - é um ambiente virtual de aprendizagem social construído para apoiar tanto o professor quanto os alunos iniciantes em programação. O nome escolhido para a ferramenta foi inspirado em um dos mais notáveis cientistas da computação, o professor Edsger W. Dijkstra. Seu artigo "Go To Statement Considered Harmful" (DIJKSTRA, 1968) influenciou fortemente para a depreciação do comando "`Goto`" em prol de estruturas de controle, tais como as estruturas de sequência, repetição e seleção. Ele alegava que o artifício era motivo para vários erros de programação.

Os elementos conceituais que compõem o modelo serão apresentados na seção 5.1. Em seguida a seção 5.2 apresentará uma visão geral do ambiente proposto, juntamente com detalhes de sua arquitetura. As ferramentas cognitivas para apoio das atividades de programação serão embasadas no conceito de linguagem de concepção, padrões elementares e planos de programação, apresentado na seção 5.3. Os aspectos técnicos sobre o desenvolvimento do protótipo elaborado a partir do modelo proposto neste capítulo serão apresentados no capítulo 6.

5.1 Elementos Conceituais do Ambiente Proposto

Para facilitar a concepção de problemas e seus respectivos modelos mentais de programação, este trabalho propõe a construção de um ambiente de aprendizagem social que utiliza como ferramentas cognitivas uma linguagem de concepção para programação e padrões elementares de programação.

Trata-se de um ambiente virtual para resolução de problemas de programação capaz de auxiliar o professor a criar uma série de atividades (problemas) para serem aplicados aos alunos. A formulação de um conjunto de questões deve orientar o aluno no processo de concepção do conhecimento, ao ponto que possibilita a reflexão do aluno sobre os aspectos necessários para solucionar tais problemas. Além disso, para a implementação da solução proposta, foi realizada a integração do ambiente com a biblioteca de programação visual

Blockly, apresentado na seção 2.2.6.2.

DEKSTRA foi modelado com base na definição de Ambientes Aprendizagem Social apresentado na seção 2.2.5. Desta forma, podemos dizer que a arquitetura do ambiente é composta por um conjunto de Agentes (Ag), Ferramentas Cognitivas (Fcog), Ferramentas de Comunicação (Fcom) e um conjunto de Atividades Interativas de Aprendizagem (AAp). As definições de cada um destes elementos estão descritas abaixo:

5.1.1 Conjunto de Agentes do Sistema (Ag)

Um agente é qualquer coisa que pode perceber seu ambiente através de sensores e agir nesse ambiente por meio de atuadores. (RUSSELL; NORVIG, 2003). Um agente de software pode receber entradas do teclado, conteúdo de arquivos e pacotes de rede como sensores de entrada e age no ambiente mostrando resultados na tela, gravando em tabelas do banco de dados e enviando pacotes pela rede. O agente artificial do ambiente DEKSTRA é um agente de memória. Agentes de memória também são conhecidos como agentes reativos baseados em modelo (RUSSELL; NORVIG, 2003). Sua principal função é participar dos protocolos de colaboração a fim de auxiliar o aluno na resolução dos problemas. Suas principais características são as seguintes:

- ❏ Utiliza um modelo interno que representa um modelo mental de resolução do problema;
- ❏ É capaz de perceber as alterações realizadas pelo aluno e reagir a essas mudanças;
- ❏ É autônomo;
- ❏ Colabora com o aprendiz;
- ❏ Possui um perfil de companheiro aprendiz.

Além do agente artificial, outros agentes humanos possuem papéis definidos no sistema. O agente aprendiz humano receberá no ambiente, diversas missões para resolução de problemas de programação. Ele contará com a ajuda do agente artificial companheiro que participará sistematicamente da execução das tarefas para cumprimento dos objetivos de aprendizagem. A função do agente professor humano é cadastrar problemas, criando um modelo mental para resolução dos problemas a partir do uso de metas, planos de programação e ações primitivas, além de mediar as ações do aluno no ambiente, permitindo que este adquira todos os conhecimentos necessários para a prática de programação.

5.1.2 Conjunto de Ferramentas Cognitivas (Fcog)

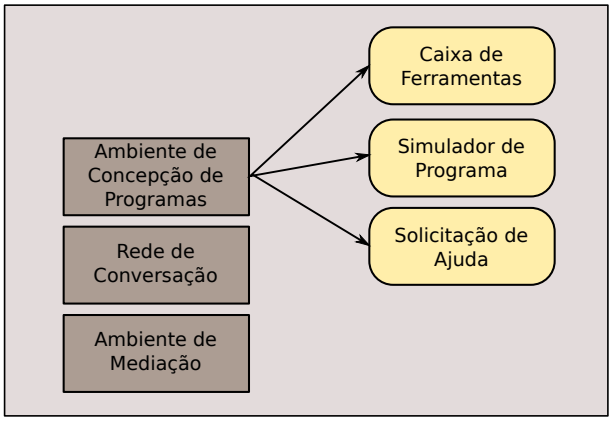
Denomina-se ferramentas cognitivas o conjunto de ferramentas que podem ser utilizadas para a comunicação, colaboração ou para realização de determinada tarefa. COSTA,

PARAGUAÇÚ e PINTO (2009) afirmam que as ferramentas cognitivas são mediadoras da interação entre os agentes. As ferramentas cognitivas podem ser classificadas como: ferramentas para auxílio de realização de tarefas, ferramentas para colaboração e ferramentas para comunicação.

As ferramentas cognitivas utilizadas no ambiente DEKSTRA são as seguintes:

- **Ferramentas para realização de tarefas**: o ambiente de concepção, a caixa de ferramentas para programação visual e o simulador de programa;
- **Ferramentas para colaboração**: a solicitação de ajuda ao agente companheiro, a rede de conversação com outros agentes aprendizes;
- **Ferramentas para comunicação**: o ambiente de mediação entre aluno e professor.

Figura 30 – Ferramentas Cognitivas do ambiente proposto



Fonte: Autor

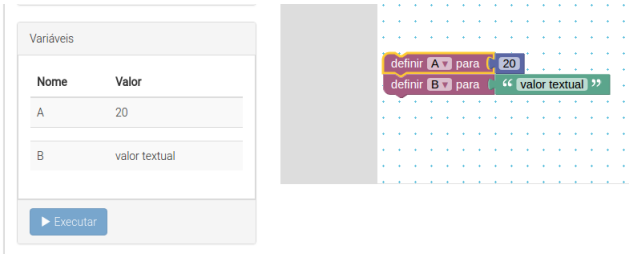
A figura 30 representa o conjunto de ferramentas existentes no modelo proposto. O ambiente de concepção é formado pela a caixa de ferramentas para programação, simulador de programa e solicitação de ajuda ao agente companheiro.

A caixa de ferramentas deve ser utilizada pelo aluno para construção de código-fonte. Tais ferramentas se baseiam no uso de blocos encaixáveis, uma metáfora para programação utilizada no conceito de linguagens visuais de programação, apresentado na seção 2.2.6.

O simulador de programa (figura 31) é uma ferramenta integrada ao ambiente de concepção capaz de simular a execução passo-a-passo do programa criado pelo aluno. O simulador permite ao aluno fazer uma avaliação de seu próprio código, acompanhando, a cada passo da execução, as mudanças dos valores das variáveis do programa.

A solicitação de ajuda ao agente companheiro é uma função disponível no ambiente de concepção onde o aluno pode requisitar a ajuda do agente artificial. O agente deverá então avaliar o código-fonte construído pelo aluno até aquele momento e fornecer para o aluno

Figura 31 – Simulador de Programa



Fonte: Autor

uma questão resgatada da metalinguagem de manipulação existente no modelo mental do problema criado pelo professor. Esta questão deverá permitir que o aluno reflita sobre o que ele deve fazer para resolver uma determinada meta estipulada pelo modelo mental do problema.

A rede de conversação pode ser utilizada pelo aluno para compartilhar com outros alunos tanto os problemas encontrados na resolução de problemas quanto o seu próprio ponto de vista quanto à uma possível resposta a um problema. É um ambiente que permite a interação e troca de conhecimentos entre pares, permitindo que um aluno receba ajuda de outros alunos mais capazes.

O Ambiente de mediação permite a aluno solicitar ajuda diretamente ao professor, possibilitando que o professor colabore com a solução que está sendo construída pelo aluno. É também neste ambiente que o aluno submeterá sua resposta ao problema proposto pelo professor para que este possa avaliar a resposta do aluno.

5.1.3 Ferramentas de Comunicação (Fcom)

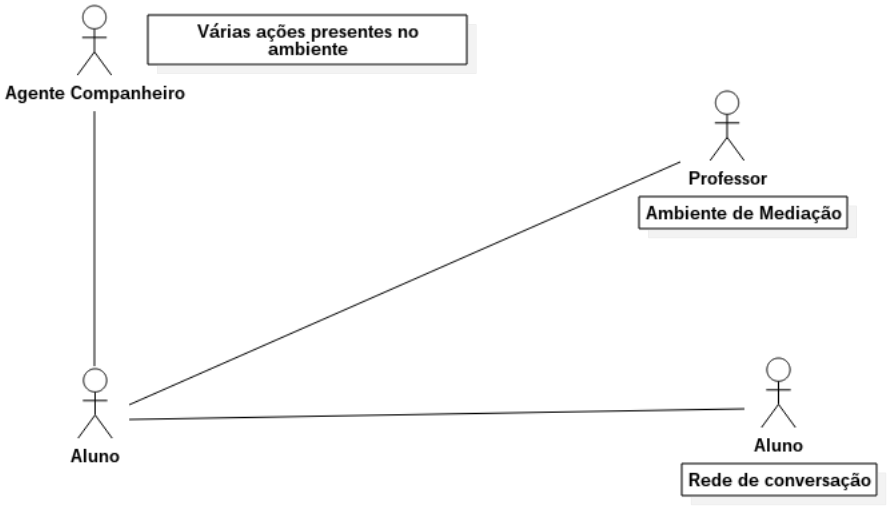
Nesta seção vamos descrever as formas de interação entre os objetos que compõem o ambiente de aprendizagem, observando como eles se relacionam. Também descreveremos cenários interativos e os recursos utilizados nos processos de comunicação entre os agentes humanos.

Por se tratar de um ambiente onde a aprendizagem está baseada nas interações sociais entre os agentes, as ações do sistema envolvem a participação de agentes reativos e agentes humanos através de troca de mensagens. Desta forma, o conjunto de ferramentas de comunicação (FCom) do ambiente DEKSTRA é pode ser descrito em função dos meios de interação entre os agentes do sistema, quer humanos ou artificiais.

A figura 32 apresenta três classes distintas de agentes do ambiente DEKSTA, considerando que cada uma possui funções próprias no ambiente:

1. O Agente companheiro - esta classe é composta pela descrição de um agente reativo baseado em modelo, que vai acompanhar e auxiliar o aprendiz na interação com o ambiente e na realização de uma tarefa de programação. Este agente possui:

Figura 32 – Ferramentas de Comunicação do Ambiente



Fonte: Autor

- ❏ O modelo mental do aluno, que descreve como o aluno entende o problema que ele está tentando resolver.
- ❏ O modelo mental fornecido pelo professor, que descreve um padrão de solução para o problema que o aluno está tentando resolver.
- ❏ Uma base de questões associadas aos elementos da linguagem de programação, formando assim uma meta-linguagem para manipulação dos elementos da linguagem de programação em estudo (ver seção 3.4) .
- ❏ Uma base de regras comportamentais que define a sequencia de ações comportamentais que serão assumidas pelo agente.

2. O Agente Professor - esta classe descreve o agente humano responsável pelas mediações nas discussões entre os alunos, ou quando for solicitado sua ajuda, ou ainda nas avaliações das atividades de programação. Para tais intervenções, o agente professor utilizará o ambiente de mediação ou a rede de conversação entre alunos. Estes recursos utilizados nas intervenções são repositórios de mensagens trocadas entre aluno e professor ou entre aluno e aluno e que descrevem os processos de comunicação ocorridos em cada tarefa de programação.

3. O Agente Aprendiz - esta classe de agente vai descrever o agente humano aprendiz. Ele deve interagir com todos as demais classes de agentes. Naturalmente uma base de informações e um histórico de problemas resolvidos estão presentes no sistema pra que se possa identificar o aluno e seu progresso dentro do ambiente.

5.1.4 Atividades Interativas de Aprendizagem (AAp)

Define-se atividades interativas de aprendizagem como o conjunto de atividades que acontecem entre os agentes, sejam eles humanos ou artificiais.

A partir do ambiente de concepção, uma série de atividades interativas são disponibilizadas para o aluno. A caixa de ferramentas e, mais propriamente, o painel de construção de código-fonte é em si uma atividade interativa, onde o aprendiz interage com os blocos visuais de programação, encaixando-os e movendo-os pela tela.

Além da caixa de ferramenta e o painel de construção de código-fonte, o aprendiz pode interagir com outros colegas a partir do compartilhamento de seu próprio código-fonte, ou comentando o código-fonte de outro colega a partir da rede de conversação.

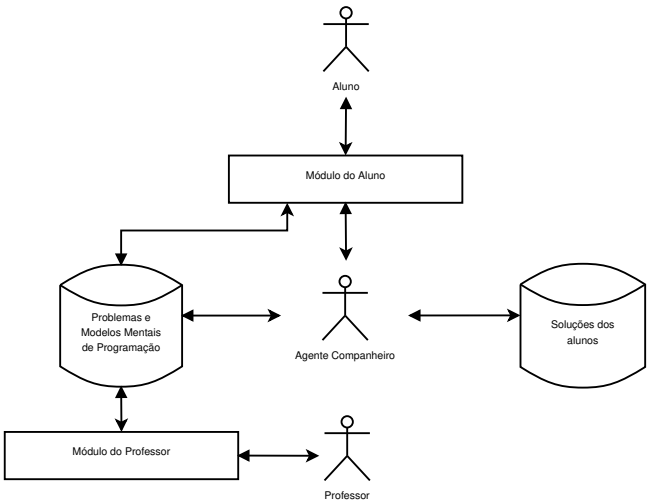
Outro tipo de atividade interativa pode ocorrer no ambiente de mediação, quando o professor acessa o código-fonte do aluno para avaliar o trabalho do aluno.

No processo de interação entre o aprendiz e o agente companheiro, nosso agente artificial fornecerá as metas para a solução de determinado problema de programação. Ele também poderá fornecer auxílio, verificando até que ponto o aluno conseguiu resolver o problema e trazendo uma questão que o auxilie em determinado momento.

5.2 Visão Geral da Solução Proposta

Conforme mostrado na figura 33, o ambiente DEKSTRA consiste basicamente de dois subsistemas: Módulo Professor e Módulo Aluno. As telas iniciais dos dois módulos estão ilustradas nas figuras 34 e 39.

Figura 33 – Proposta para o AAS Dekstra



Fonte: Autor.

Algumas funções estão associadas a ambos os módulos, quais sejam: o acesso ao ambiente de mediação, usado entre agente professor e agente aprendiz, e a rede de con-

versação, entre os agentes aprendizes com a presença do professor. São nesses ambientes que ocorrem as interações entre agentes humanos.

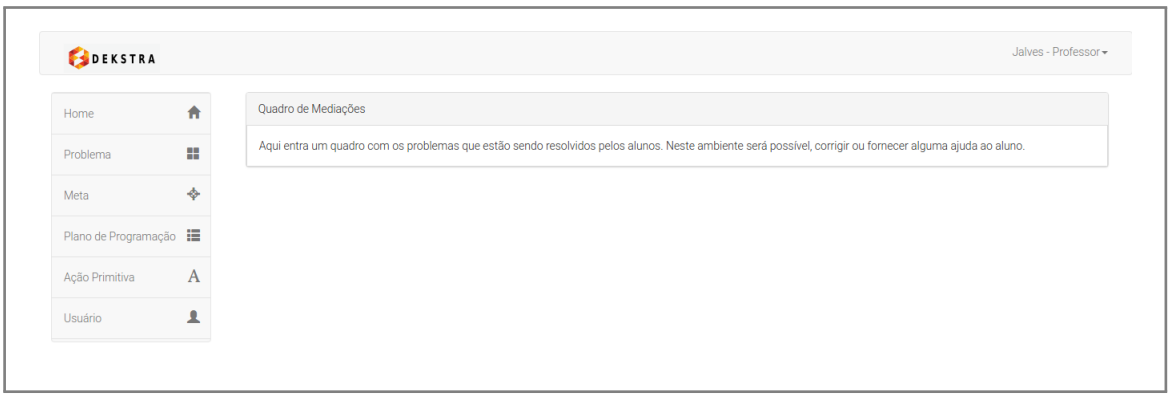
5.2.1 Descrição do Módulo Professor

O Módulo do Professor consiste na interface entre o ambiente de aprendizagem e o agente professor. Este subsistema é responsável pelo cadastro dos problemas que serão resolvidos pelos alunos, bem como o cadastro do modelo mental de implementação de cada problema.

O modelo mental é composto por todas as metas, planos e ações necessários para orientação do aluno na resolução do problema. Além disso, o Módulo do Professor também é responsável pelo cadastro dos usuários (alunos e professores) que utilizarão o sistema.

A tela inicial do módulo professor está ilustrada na figura 34.

Figura 34 – Tela Inicial para o módulo professor

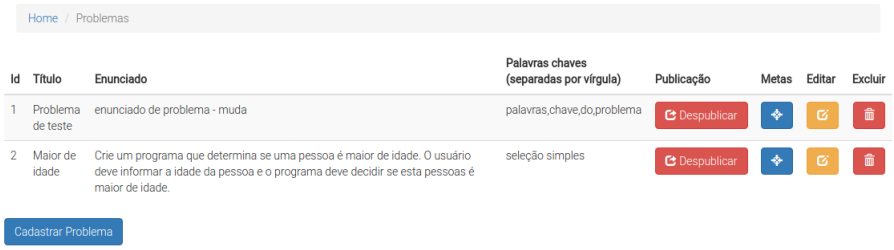


Fonte: Autor.

O módulo professor é formado pelos elementos abaixo:

1. **Problema** - nesta tela o professor pode cadastrar um novo problema. Para cada problema criado, o professor deve criar novas metas subdividem a resolução do problema. Opcionalmente, o professor pode reaproveitar metas criadas para resolução de outros problemas. A figura 35 apresenta a tela de cadastro de problemas, mostrando quais ações estão disponíveis para o usuário.

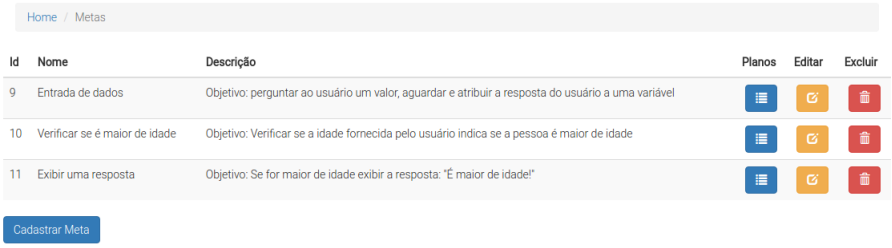
Figura 35 – Tela de cadastro de problemas



Fonte: Autor

2. **Meta** - tela onde o professor pode cadastrar uma nova meta. Uma meta pode ser utilizado em mais de um modelo mental de resolução de um problema. Além disso, uma meta pode estar associada a diversos planos de programação. A ação de associar uma meta a planos de programação é realizado através do botão *Planos* presente nesta tela (ver figura 36).

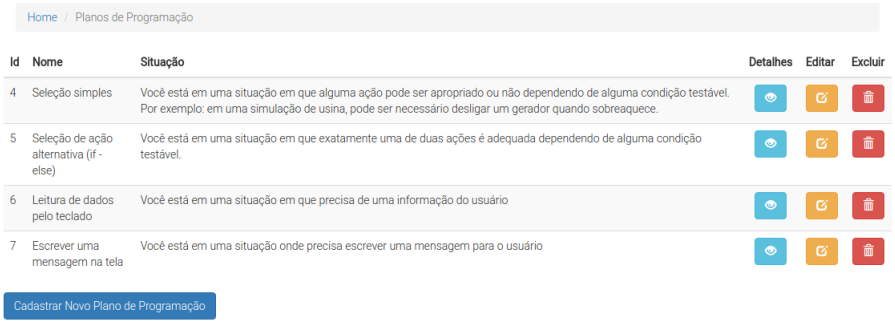
Figura 36 – Tela de cadastro de Metas



Fonte: Autor

3. **Plano de Programação** - tela onde o professor pode cadastrar todos os planos de programação que ele achar necessário. A opção *detalhes*, na figura 37, possibilita associar o plano de programação à ações primitivas. Também é através deste elemento que o professor pode criar questões que sejam pertinentes sobre o uso do plano de programação, levando o aluno a refletir sobre suas ações na realização da tarefa de programação.

Figura 37 – Tela de cadastro de Planos de Programação



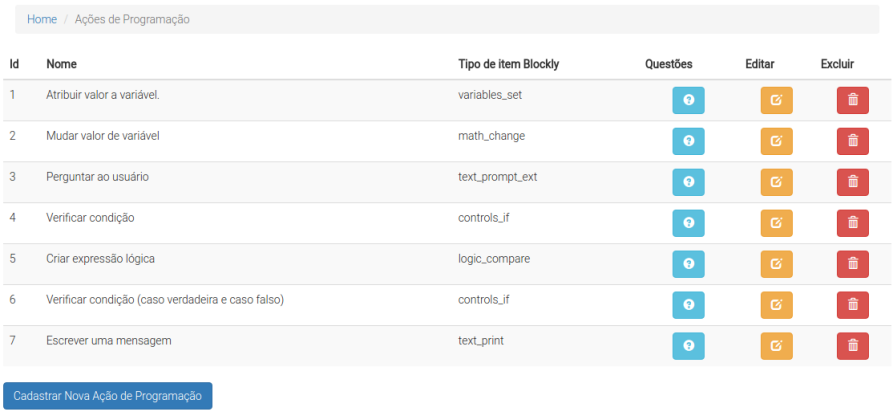
Fonte: Autor.

4. **Ação Primitiva** - Nesta tela é possível cadastrar ações primitivas de programação. As ações primitivas são associadas a tipos de blocos existentes no Blockly. Por exemplo, a ação *Atribuir valor a variável* está associado ao bloco *variables_set* da linguagem visual de programação (confira a figura 38). Esta associação permite um mapeamento do programa desenvolvido pelo aluno, verificando desta forma que ações primitivas estão sendo utilizadas pelo aluno.

   A opção *questões* desta tela permite criar questões sobre cada ação primitiva. Por exemplo, a ação *atribuir valor a variável* possui as seguintes questões associadas:

   - ❏ Para qual variável será atribuído um valor?
   - ❏ Que valor será atribuído ?

Figura 38 – Tela de cadastro de Ações Primitivas



Fonte: Autor.

5. **Usuário** - Nesta tela o professor pode cadastrar todos os usuários do ambiente DEKSTRA. Na versão atual do protótipo, apenas dois perfis são permitidos: pro-

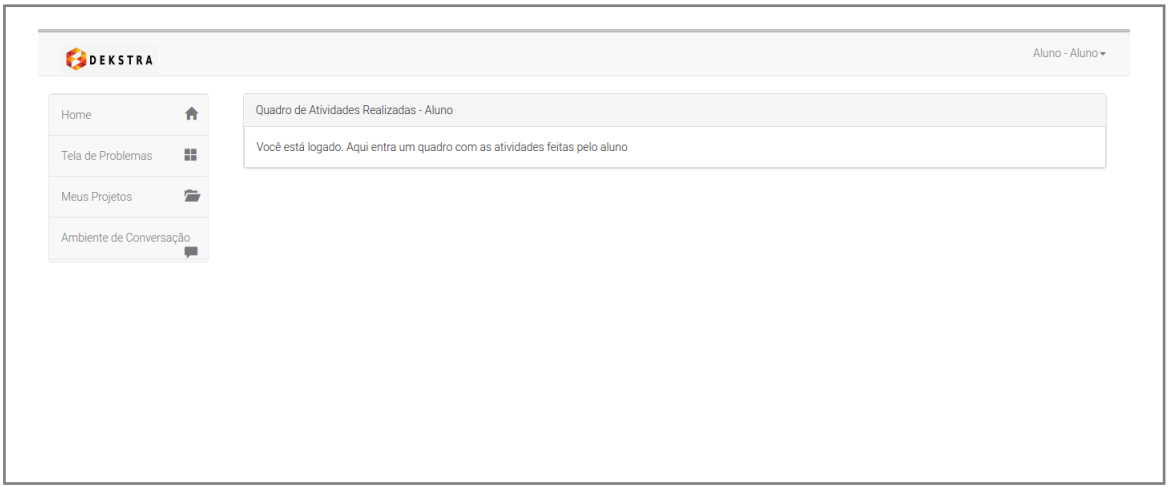
fessor e aluno. Os usuários com perfil professor poderá acessar o módulo de professor, possibilitando assim o cadastro de problemas, metas, planos de programação e ações primitivas.

6. **Ambiente de mediação** - O ambiente de mediação é apresentado na página principal do módulo professor, na área central, facilitando a identificação dos problemas selecionados pelos alunos. O objetivo é permitir ao professor a rápida identificação de submissões de pedidos de ajuda e avaliação das respostas enviadas pelos alunos. Neste ambiente o professor e o aluno podem trocar mensagens e informações de maneira assíncrona.

5.2.2 Descrição do Módulo Aluno

O Módulo Aluno permite a comunicação entre o aluno e o ambiente de aprendizagem. Este módulo possibilita que o aprendiz visualize os problemas cadastrados previamente pelo professor para que, em seguida, ele possa fornecer uma solução implementada em linguagem visual de programação.

Figura 39 – Tela Inicial para o módulo aluno



Fonte: Autor.

O módulo aluno é formado pelos elementos abaixo:

1. **Tela de Problemas** - permite a escolha do problema a ser resolvido. O aluno terá à sua disposição todos os problemas cadastrados pelo professor. Uma vez escolhido o problema que será resolvido, o aluno é encaminhado para o ambiente de concepção. Este ambiente de concepção que é subdivido em 2 camadas funcionais, descritas como se segue:

   a) *Escolha da meta ser cumprida.* Apesar das metas aparecerem em uma certa ordem escolhida pelo professor, o aluno tem liberdade de escolher o que ele fará

primeiro. Esta liberdade deixará o aluno mais a vontade para fazer primeiro o que ele tem mais segurança além de permitir que o aluno crie seus próprios percussos cognitivos para resolver um determinado problema.

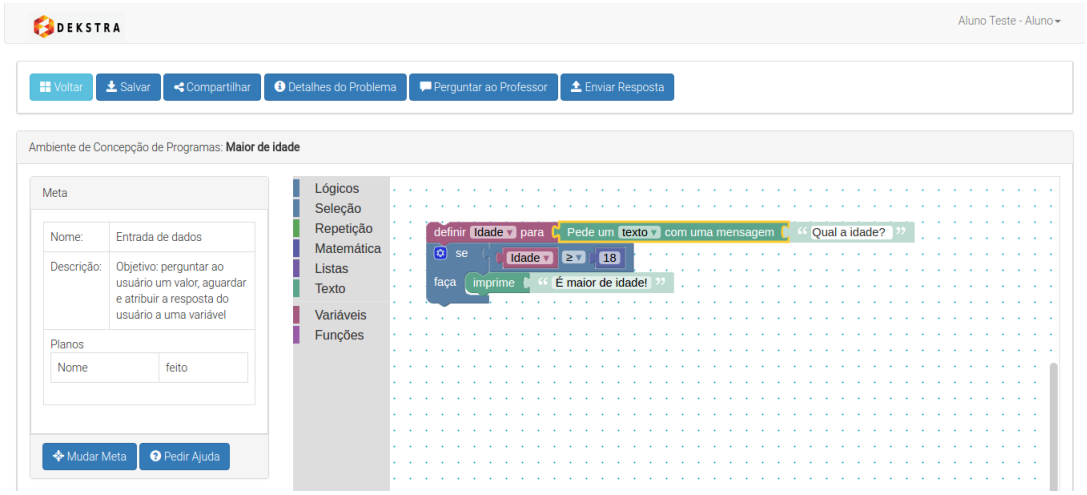
b) *Implementação do problema escolhido.* É o ambiente de concepção propriamente dito. A figura 40 apresenta os elementos existentes no ambiente de concepção que são os seguintes:

- ❏ as ferramentas para auxílio cognitivo: a caixa de ferramentas do Blockly e a ajuda do agente companheiro através de questões reflexivas sobre o problema e sobre a linguagem de programação em si. A ajuda do agente companheiro pode ser requisitada através da opção *Pedir Ajuda*.
- ❏ a área de edição/encaixe dos blocos de programação.
- ❏ a opção *Mudar Meta* - permite ao aluno mudar a meta na qual ele está trabalhando.
- ❏ a opção *Salvar* - Permite ao aluno salvar seu projeto antes de deixar o ambiente de concepção.
- ❏ opção *compartilhar* - permite que o aluno publique seus programas para obter feedback de outros usuários.
- ❏ a opção *Detalhes do Problema* - Informa para o aluno o enunciado do problema e as metas que devem ser alcançadas para resolução do problema, conforme pode ser visto na figura 41. Essas informações podem ser acessadas a qualquer momento no ambiente de concepção.
- ❏ a opção *Perguntar ao Professor* - Permite ao aprendiz solicitar ajuda do professor, criando um fórum de conversação com o professor que terá acesso ao código-fonte criado pelo aluno até aquele momento.
- ❏ a opção *Enviar Resposta* - Permite ao aluno submeter sua resposta para correção do professor.

2. **Opção *Meus Projetos*** - Nesta opção o aluno pode escolher um problema já resolvido ou em resolução para edição. Isso é possível pois no ambiente de concepção, quando o aluno está trabalhando em um determinado problema, ele pode optar por salvar seu trabalho para continuar depois.

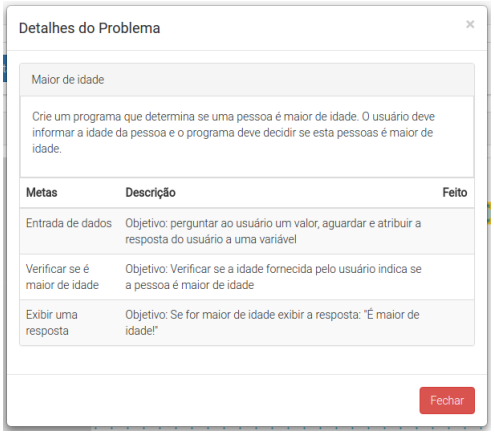
3. **Ambiente de conversação** - A opção *compartilhar*, presente no ambiente de concepção, permite ao aprendiz compartilhar seu trabalho com outros alunos. Uma vez que o trabalho foi compartilhado, é possível visualizar comentários ou comentar alguma solução publicada no ambiente de conversação.

Figura 40 – Ambiente de Concepção para resolução de problemas



Fonte: Autor.

Figura 41 – Detalhes do problema: tela que exibe enunciado do problema e suas metas



Fonte: Autor.

5.3 Uma Linguagem de Concepção de Programas baseada em Padrões Elementares de Programação

Como base para a elaboração do modelo da ferramenta de ajuda a concepção do ambiente de aprendizagem DEKSTRA, foi utilizado o modelo de compreensão de programação proposto por Soloway, Adelson e Ehrlich (1988). Segundo os autores, este é um modelo de abordagem *top down*, onde o aluno deve se focar primeiro na concepção de uma estratégia para solucionar o problema para só então pensar na estrutura de implementação do programa. Em resumo, segundo este modelo, para aprender a programar, o aluno deve ser capaz de criar planos de programação e, para ajudar o aluno a adquirir esta habilidade, o modelo propõe o uso de regras de discurso como uma ferramenta de apoio. Maiores detalhes sobre este modelo foram descritos na seção 3.2.1.

De fato, a construção de planos de programação pode ser considerada uma tarefa não trivial. Lemos, Barros e Lopes (2003) tratam os planos de programação como uma estratégia mental de alto nível. Contudo, segundo as autoras, os *planos de programação* podem ser mapeados em *padrões elementares de programação* a partir das *ações primitivas de programação*. Estes últimos (ações primitivas) compõe os primeiros (planos de programação) de forma que é possível construir planos de programação a partir de uma coleção de ações primitivas. A relação entre estes conceitos pode ser observada na figura 15 apresentada na seção 3.2.2.

Uma linguagem de concepção para programação é proposta por Paraguaçu (1997), conforme demonstrado na sessão 3.4. O quadro abaixo mostra um exemplo desta linguagem. Contudo esta linguagem de concepção representa as interações com uma linguagem de programação textual. A linguagem de concepção proposta neste trabalho deve interagir com uma linguagem de programação visual.

Tabela 6 – Linguagem de Concepção para linguagem de programação textual

Bloco de Programação (L2)	**Linha de Programação (L1)**	**Meta-linguagem de concepção**
Ação alternativa		"Posso construir um bloco de código que sempre executa uma de duas ações apresentadas, dependendo da avaliação de uma condição?"
	if(media > 6)	"Qual condição será avaliada?"
	output("aprovado");	"Que mensagem deve ser apresentada?"
	else	"O que acontece se a condição não for verdadeira?"
	output("reprovado");	"Que mensagem deve ser apresentada?"

Desta forma, é necessário adaptar a definição da Linguagem de concepção de programas para o contexto desta pesquisa. A escolha por trabalhar com linguagem de programação visual pode ser justificada pelas seguintes características:

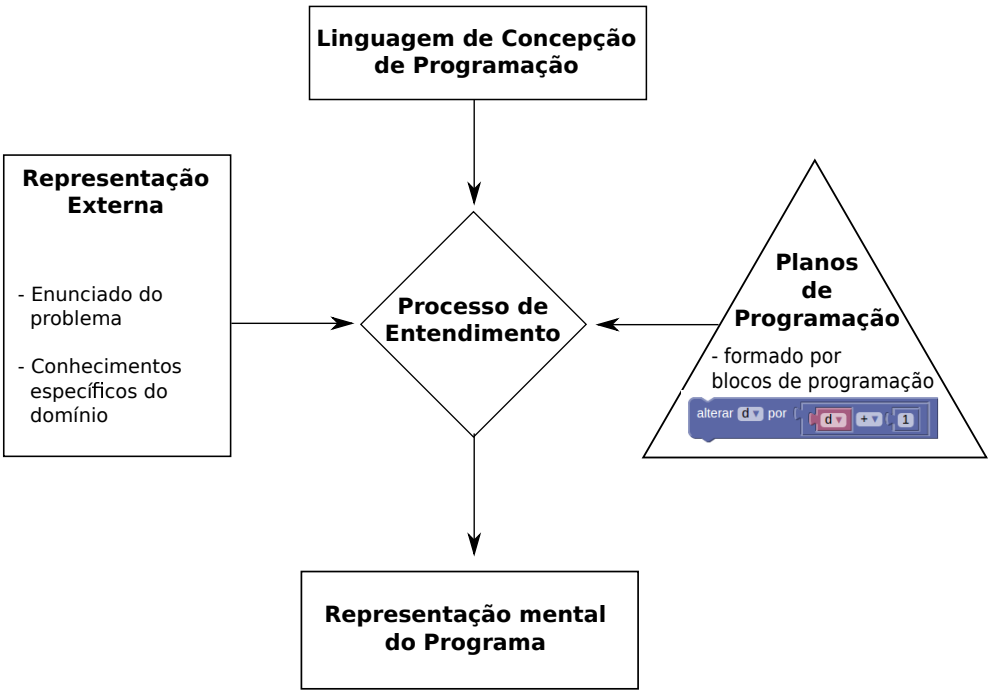
(i) Uso de cores para reforçar a classificação de diferentes tipos de ações de programação que podem ser escolhidas pelo usuário.

(ii) Uso de Metáfora: Quebra-cabeça ou blocos de encaixe. A vantagem é que as características de uma peça de encaixe já são conhecidas. Seu formato define se a peça encaixa ou não em determinada posição.

(iii) Inibe erros de sintaxe. Tais erros são naturalmente evitados pelo simples uso da metáfora de quebra-cabeça: se algo não encaixa é porque está errado.

(iv) Programação bidimensional. O programa não se desenvolve apenas na vertical ou linha-a-linha, como nas linguagens de programação textual. Mas também na horizontal, com a utilização de encaixes horizontais ou internas a um determinado bloco de programação.

Figura 42 – Modelo da ferramenta cognitiva baseada em Linguagem de Concepção



Fonte: Autor.

5.4 Discussões

Este capítulo apresentou a arquitetura de um ambiente de aprendizagem social para o ensino de programação para alunos iniciantes, denominado DEKSTRA. O ambiente aqui proposto utiliza como ferramentas cognitivas: uma linguagem de concepção de programas visuais baseada em padrões elementares de programação, o ambiente de concepção, a caixa de ferramentas para construção de código-fonte visual e o simulador de programa.

O ambiente DEKSTRA foi projetado para dar suporte à metodologia de ensino de programação abordada no capítulo 4, facilitando a interação entre professor e aluno e entre o aluno e seus colegas. Além disso, um agente companheiro (agente artificial) está presente no ambiente à disposição do aluno para ajudá-lo quando for solicitado.

Para poder interagir com aluno de maneira apropriada, o agente companheiro deve ter acesso tanto ao modelo mental de um problema que é disponibilizado pelo professor, quanto à solução proposta pelo aluno, mesmo que incompleta. De posse desses dois modelos, o agente poderá então colaborar com aluno a partir das questões cadastradas

pelo professor no momento em que este registra os planos de programação e as ações primitivas necessárias para resolver determinado problema.

De forma geral, o ambiente DEKSTRA é formado por dois módulos: Professor e Aluno. Este capítulo apresentou as características gerais desses módulos, no entanto alguns aspectos técnicos não foram abordados até o momento. No capítulo seguinte será apresentado diversos desses aspectos técnicos, tais como: requisitos funcionais, projeto arquitetural, modelagem de dados, tecnologias utilizadas e maiores detalhes sobre a implementação do agente companheiro artificial.

6 ASPECTOS TÉCNICOS DO PROTÓTIPO DESENVOLVIDO

6.1 Introdução

Neste capítulo apresentaremos aspectos técnicos sobre a construção de um protótipo para o ambiente de aprendizagem social proposto. Apresentaremos os requisitos funcionais do sistema (seção 6.2), o projeto arquitetural do sistema (seção 6.3), a modelagem dos dados do sistema (seção 6.4) e as tecnologias utilizadas para a construção do protótipo (seção 6.5). Além disso, na seção 6.6 serão apresentados detalhes de implementação do agente artificial companheiro.

6.2 Requisitos Funcionais e não-funcionais do sistema

Na sub-seção de requisitos funcionais serão apresentadas as funções a serem implementadas neste ambiente de aprendizagem virtual. Quanto aos requisitos não-funcionais, este trabalho descreverá as condições de comportamento e as restrições do sistema. Também definiremos os usuários do ambiente e suas atribuições.

6.2.1 Requisitos Funcionais

Diante da análise realizada sobre as características de um ambiente de aprendizagem social, apresentadas na seção 2.2.5 do capítulo 2, foram levantados os seguintes requisitos funcionais:

1. **Prática do ensino usando padrões de programação para alunos iniciantes:** O ambiente deve auxiliar na prática de programação através do uso de planos de programação gerados a partir de padrões de programação anteriormente definidos e ensinados em sala de aula, de acordo com a metodologia pedagógica proposta no capítulo 4.

    a) *Manutenção de Metas de programação*: o ambiente deve permitir adicionar, remover, modificar e configurar metas de programação. Para um dado problema, o professor poderá cadastrar uma ou mais metas de programação.

    b) *Manutenção de Planos de programação*: o ambiente deve possibilitar adicionar, remover, modificar e configurar planos de programação. Para uma meta de programação, o professor poderá criar novo plano de programação ou utilizar um plano já existente no sistema. Deverá ainda adicionar questões relevantes sobre o plano que está cadastrando, para que o agente artificial companheiro possa fornecer ajuda ao aluno através destas questões.

c) *Manutenção de Ações Primitivas*: O sistema deve permitir adicionar, remover, modificar e configurar ações primitivas. Cada ação primitiva possui de possuir uma ou mais questões relevantes sobre a ação primitiva. Além disso, uma ação primitiva deve estar atrelada a um objeto de programação da biblioteca *Blockly*.

d) *Visualização de Planos de programação*: O sistema deve permitir que o aluno visualize o plano de programação a partir do ambiente de concepção de programas.

e) *Utilização de planos de programação a partir de uma linguagem de concepção de programas visuais*: Sugerir ao aluno, através de questões de reflexão, que plano de programação ou ação primitiva deve ser utilizado em um dado momento da realização da tarefa.

2. **Resolução de Problemas**

a) *Visualização de Problemas*: O sistema deve disponibilizar para o aluno uma *Tela de Missões*, onde serão colecionados todos os problemas disponibilizados pelo professor para resolução.

b) *Escolha do problema a ser resolvido*: o sistema deve permitir ao aluno escolher um dos problemas disponibilizados pelo professor para ser resolvido.

c) *Escolha da meta do problema a ser resolvida*: O sistema deverá fornecer ao aluno a opção de mudar de meta que está sendo resolvida a qualquer momento, dentro do ambiente de concepção de programas.

d) *Resolver Problemas*: O sistema deve disponibilizar para o aluno um *Ambiente de concepção de programas*. A resolução do problema será feita utilizando a linguagem visual de programação.

e) *Compartilhar Solução*: A solução implementada pelo aluno, mesmo que parcial poderá ser submetida para avaliação e ajuda dos colegas. Esta solução compartilhada será visualizada em um ambiente de conversação.

f) *Visualização dos comentários:* Os comentários publicados pelos colegas e professor podem ser visualizados através do ambiente de conversação

g) *Submeter Solução para correção*: O sistema permitirá ao aluno submeter a solução implementada para a avaliação e considerações do professor, através de um *ambiente de mediação*.

h) *Execução do problema:* A solução implementada poderá ser testada pelo aluno através de uma ferramenta de *simulação de programa*, existente no ambiente de concepção.

3. **Diálogo com o agente:** No espaço de concepção, um agente inteligente poderá fazer questões ao aluno quando for requisitado, procurando evidenciar o percusso cognitivo para solucionar o problema.

6.2.2 Requisitos Não-funcionais

Os requisitos não-funcionais identificados na análise da solução proposta são os seguintes:

1. **Dimensionamento da base de Problemas**: A base de problemas deve ser ampliada, possibilitando a oferta ao aluno de situações que englobem, além daquelas questões indicadas pelo professor, problemas extras para que o aluno possa praticar. Para tanto, é necessário ampliar a base de problemas.

2. **Tempo de resposta do ambiente**: O ambiente deve limitar seu tempo máximo de resposta em 30 (trinta) segundos para as atividades que necessitem da comunicação entre o cliente e o servidor.

3. **Interface de fácil uso**: A interface do ambiente deverá possibilitar aos usuários uma experiência interativa satisfatória e que traga motivação ao aluno para a utilização das ferramentas disponibilizadas pelo ambiente de aprendizagem proposto.

6.2.3 Atores do sistema

O ambiente proposto possui 2 (dois) usuários distintos:

- **Professor**: agente humano que utiliza o sistema para adicionar, remover ou modificar os problemas e os modelos mentais de programação (metas, planos e ações primitivas). Além disso, o professor é responsável por cadastrar os alunos da disciplina, corrigir as soluções dos problemas e emitir os comentários.

- **Aluno**: agente humano que interage com o sistema para resolução dos problemas de programação. Este agente pode dialogar com o agente companheiro virtual, através da solicitação de ajuda. Ele também poderá visualizar os planos de programação para o problema atual e acessar os problemas resolvidos anteriormente.

Por se tratar de um protótipo inicial do ambiente proposto, o sistema DEKSTRA não implementará controle de turmas. A figura 43 ilustra o diagrama de casos de uso que demonstra a relação entre os atores do sistema e os requisitos funcionais definidos.

6.3 Projeto Arquitetural

O ambiente DEKSTRA se baseia na arquitetura cliente-servidor, na qual cada instância de um cliente envia requisições de dado para o servidor conectado e espera pela

resposta. Por sua vez, o servidor recebe tal requisição, processa a informação e devolve o resultado para o cliente.

Para implementar essa arquitetura foi utilizado o *framework PHP Laravel* (OTWELL, 2016), que facilita a implementação de aplicações Web seguindo essa estrutura arquitetural. Na Seção 6.5.1 essa tecnologia está descrita com detalhes, sendo apresentadas as suas principais vantagens.

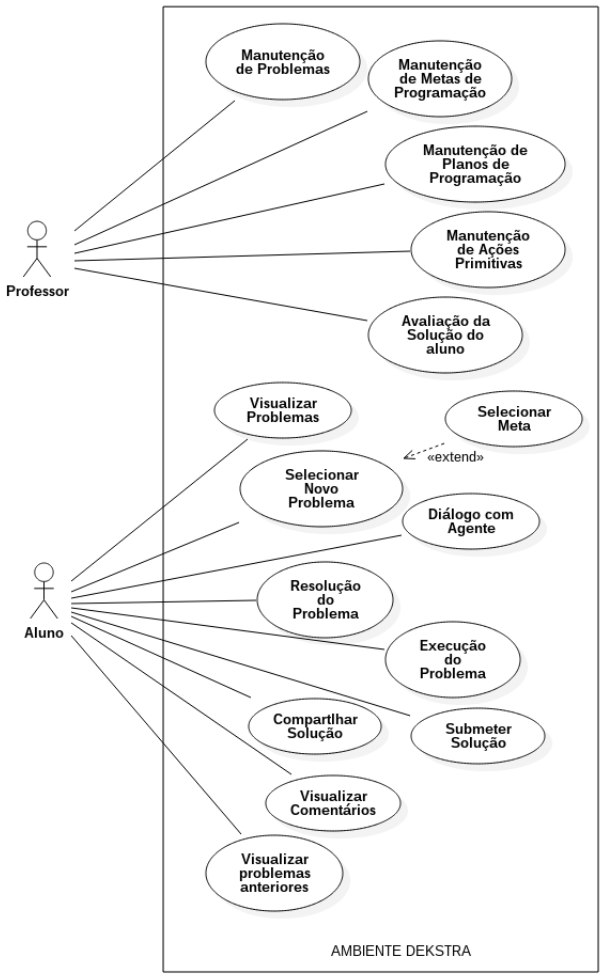


Figura 43 – Diagrama de Casos de Uso do ambiente DEKSTRA

Dentro do contexto da arquitetura cliente-servidor, o lado cliente do sistema é dividido em duas visões de acesso: a visão do aluno e a visão do professor. O lado servidor, por sua vez, é constituído por diversos módulos de cadastro desenvolvidos neste trabalho. Além disso foram utilizados dois frameworks de terceiros e as bases de dados. Na figura 44 está representado a arquitetura do sistema.

De acordo com o projeto arquitetural (Figura 44), no lado cliente do ambiente DEKSTRA, a visão do aluno é representada por uma interface Web que lhe permite visualizar o problema a ser resolvido, consultar os problemas que já foram resolvidos, solicitar ajuda do agente companheiro, solucionar o problema, executá-lo e submeter a solução para o

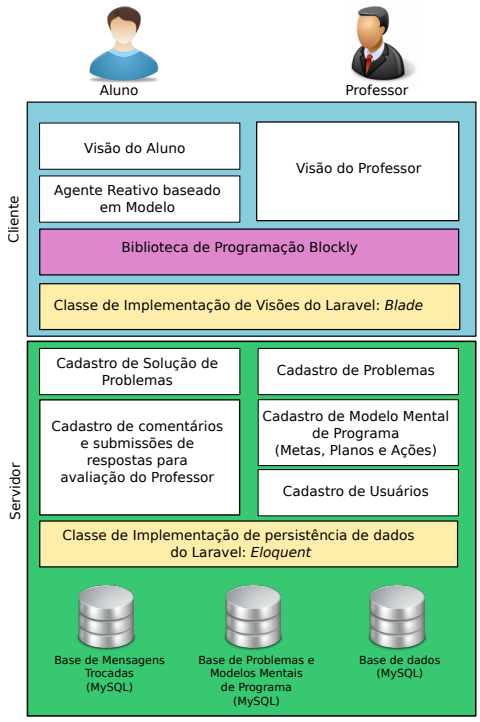


Figura 44 – Projeto arquitetural do ambiente DEKSTRA

professor. Além disso, a visão do aluno também permitirá o compartilhamento da solução do aluno e a visualização dos comentários emitidos por ele mesmo ou pelos colegas.

Também é na visão do aluno que o agente inteligente companheiro encontra-se situado. Ele reage às ações do aluno no ambiente de concepção, percebendo as alterações que este provoca no ambiente e respondendo com ações adequadas à situação. Uma descrição mais detalhada sobre o agente inteligente companheiro encontra-se na seção 6.6.

Por sua vez, a visão do professor permite que este agente mantenha as informações necessárias para o funcionamento do ambiente – problemas e modelos mentais de programas. Além disso, o professor poderá analisar as soluções submetidas pelos alunos e emitir comentários que podem ser visualizados pelos alunos.

Do lado servidor, os módulos de cadastro gerenciam a manutenção dos problemas, modelos mentais de programação, dos comentários e dos usuários. Neste, o professor pode inserir, remover e editar cada uma das informações citadas.

As implementações pertencentes ao *framework Laravel* encontram-se tanto no lado cliente quanto no lado servidor e são utilizadas para facilitar a construção das telas de interfaces com o usuário. Para a persistência de dados, foi utilizada a tecnologia *Eloquent*, parte integrante do framework Laravel, a qual é responsável pelo mapeamento objeto-relacional dos dados, possibilitando a implementação de um sistema orientado a objetos e armazenar os dados em um banco de dados relacional. tais implementações serão descritas na seção 6.5.1.

Outra ferramenta de terceiros utilizada é a biblioteca *Blockly*. Essa biblioteca, incorporada ao ambiente DEKSTRA, permite que os alunos construam suas soluções utilizando

linguagem de programação visual. A biblioteca *Blockly* permite a representação do código visual criado utilizando a linguagem XML, possibilitando desta forma que o sistema salve as implementações do aluno em banco de dados, facilitando a recuperação e o compartilhamento do código-fonte visual criado. Além disso, a biblioteca *Blockly* permite que o código-fonte visual seja convertido para diversas linguagens de programação tradicionais.

6.4 Modelagem dos dados do Sistema

Na modelagem conceitual do sistema, foram identificadas as seguintes entidades de dados:

- **Ação** - representam ações primitivas do usuário e estão atreladas a objetos pertencentes à biblioteca Blockly. Uma ação primitiva pode estar associado a um ou mais planos de programação.
- **Plano** - representam planos de programação esquematizados a partir do estudo de padrões elementares de programação.
- **Questão** - representa questões associadas a ações ou a planos.
- **Problema** - representa problemas que são propostos pelo professor. Todo problema possui: um título, um enunciado, uma classificação de nível de complexidade e um conjunto de palavras-chave para indexação. Um problema pode ser subdividido em uma ou mais metas.
- **Meta** - representa uma meta a ser alcançada para resolver um problema de programação. Uma meta tem um nome que a identifique e uma descrição. Cada meta pode ter um ou mais planos de programação.
- **Solução** - representa um conjunto de soluções para problemas resolvidos pelo aluno. Cada solução está associado a um único problema e aluno.
- **Usuário** - representam usuários do sistema que possuem nome, email, perfil (professor ou aluno), login e senha.
- **Conversação** - representa um conjunto de postagens de mensagens trocadas entre alunos. Cada conversação está vinculada a uma solução criada por um aluno. Outros alunos podem responder a esta postagem inicial, formando assim um grupo de mensagens que tratam do mesmo tópico: a solução compartilhada pelo aluno que postou a primeira mensagem.
- **Mediação** - Representa mensagens trocadas entre alunos e professores. O objetivo da mediação, representado pelo parâmetro *tipo*, deve ser: tirar uma dúvida ou submeter resposta para avaliação.

Tanto a entidade **Mediação** quanto **Conversação** podem armazenar esquemas de código-fonte (armazenados no banco de dados no formato XML). Estes esquemas podem ajudar o agente humano que escreve a mensagem a representar sua resposta usando blocos de código-fonte do *Blockly*.

A figura 45 ilustra um Diagrama de Tabelas Relacionais que apresenta as entidades de dados transformados em tabelas. O diagrama também demonstra o relacionamento entre cada tabela do banco de dados.

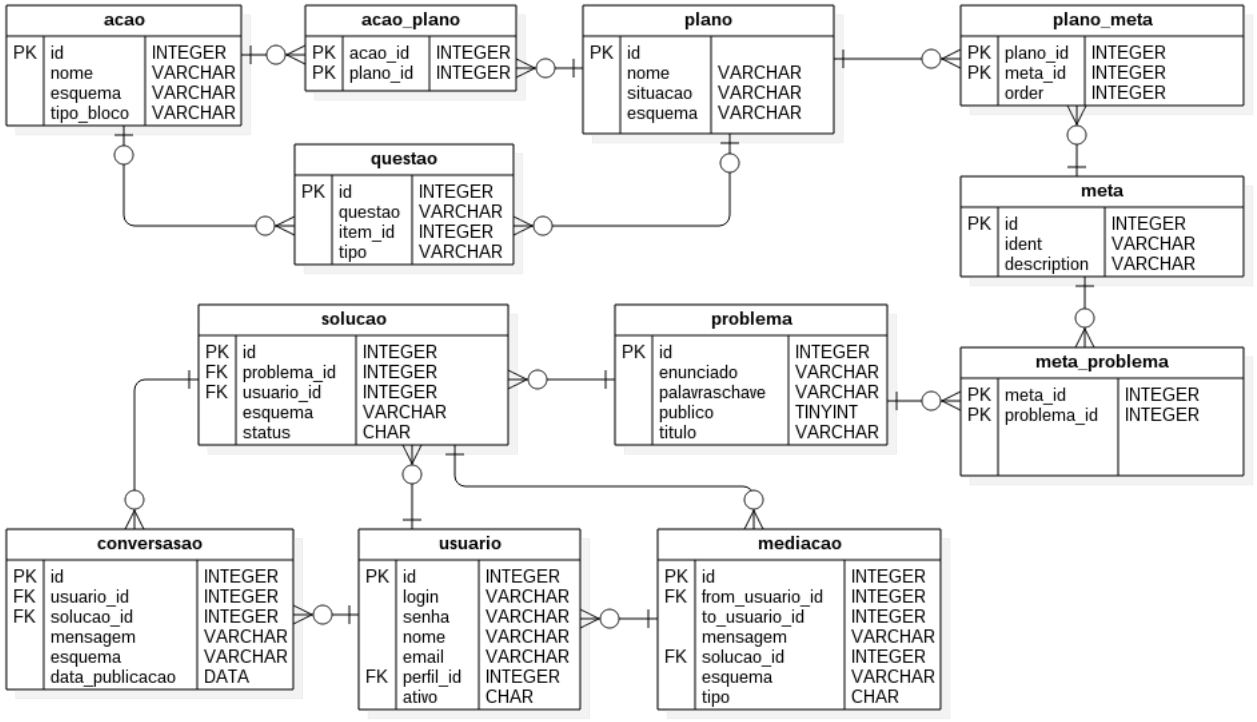


Figura 45 – Modelo de dados do ambiente Dekstra

6.5 Tecnologias utilizadas

Para o desenvolvimento do protótipo do ambiente de aprendizagem social foi utilizada a linguagem de programação PHP, seguindo a arquitetura Cliente - Servidor. O framework de PHP *Laravel* (OTWELL, 2016) foi utilizado para acelerar o processo de desenvolvimento do protótipo.

Além do Laravel, outro framework foi utilizado para a implementação de algumas funcionalidades na camada de interface. Essas funcionalidades visam ampliar a interação do usuário com o sistema. Desta forma, o framework de JavaScript *Vue.js* foi utilizado para cumprir esta função. Nesta sessão abordaremos com mais detalhes esses dois frameworks.

6.5.1 Framework PHP Laravel

A linguagem PHP (Hypertext Preprocessor) é uma linguagem de script largamente conhecida e está frequentemente associada ao desenvolvimento de sistemas para Web. De acordo com site w3techs.com (Q-SUCCESS, 2017), que coleta e processa informação sobre o uso de tecnologias Web, PHP é a linguagem de script mais usada na Internet, com cerca de 82% de aceitação.

Criada por Rasmus Lerdorf em 1994, a linguagem PHP foi originalmente denominada *Personal Home Page Tools* (PHP.NET, 2017). Quando a versão 3 da linguagem foi disponibilizada em 1998, ela foi rebatizada como *Hypertext Preprocessor*, uma vez que a linguagem tinha se tornado mais refinada e começava a ser utilizada profissionalmente.

Atualmente muitos sistemas gerenciadores de conteúdo (CMS) são desenvolvidos utilizando a linguagem PHP, dentre os quais se destacam: Wordpress, Joomla, Drupal. Também a rede social *Facebook* é desenvolvida utilizando esta linguagem (SKVORC, 2013).

Devido ao rápido crescimento e aceitação por parte dos desenvolvedores, muitos frameworks baseados em PHP foram construídos. Segundo Skvorc (2013), os frameworks mais utilizados são: CodeIgniter, Symfony, Phalcon e Laravel.

De forma geral, um *framework* é uma de implementação sobre uma determinada linguagem que encapsula alguns padrões de arquitetura de software, entre outras implementações, para resolver diferentes problemas. Como exemplo de padrões de arquitetura de software pode-se citar: o padrão *Modelo-Visão-Controle* (MVC), cujo objetivo é separar uma página Web (Visão) da lógica de negócios (Modelo) (DEACON, 2009).

Em Junho de 2011 Taylor Otwell lançava a primeira versão do Laravel, um framework PHP voltado para aplicações web. O Framework Larevel é open-source e está liberado sobre a licença do MIT. Em pouco tempo sua popularidade e aceitação levou este framework a ser conhecido como um dos melhores frameworks de PHP da atualidade (BEAN, 2015).

Segundo Bezerra e Schimiguel (2016), Alguns pontos fortes do framework Laravel são:

- ❏ Facilidade de aprendizado;
- ❏ Documentação clara;
- ❏ Sua modularidade é tão robusta, que permite que o desenvolvedor criar sua aplicação utilizando o padrão de projeto desejado, por exemplo: O desenvolvedor pode utilizar o padrão *Facade*, ou o padrão *MVC*, dentre outros.

Dentre as implementações disponíveis no framework Laravel, podemos citar:

- ❏ Módulo de Routing - Utilizado para criar rotas HTTP.
- ❏ Módulo Middleware - Utilizado para filtro de requisições HTTP.

- ❏ Módulo Responses - Utilizado para respostas HTTP, como por exemplo: JSON,File.
- ❏ Módulo Cache - Utilizado para o cache entre sua aplicação e o banco de dados.
- ❏ Módulo Eloquent - ORM ou Object Relational Mapping, módulo que faz o relacionamento do banco de dados em forma de objetos.
- ❏ Módulo Mail - Responsável por todas as funções como por exemplo, enviar emails.

6.5.2 Framework Javascript Vue.js

A linguagem JavaScript foi desenvolvida em 10 dias no ano de 1995 por Brendan Eich, inicialmente chamado de Mocha (SEVERANCE, 2012). A linguagem foi renomeada mais tarde, no mesmo ano, quando a Sun e a Netscape resolveram distribuir seu navegador Web, Netscape, juntamente com o JavaScript (NETSCAPE, 1995). A proposta da Sun era que o JavaScript se tornasse uma alternativa à única linguagem de programação na época que executava nos navegadores Web dos clientes: a linguagem Java, que possibilita o desenvolvimento de applets[1].

Percebendo-se a necessidade da existência de um padrão que definisse uma linguagem de script interpretada que executasse em outros navegadores diferentes do Netscape, em 1997 o padrão ECMA-262, chamado de ECMAScript, foi implementado baseado na versão do JavaScript da época. O ECMAScript tornou-se o padrão para diversas implementações de scripts, inclusive para o próprio JavaScript que atualmente está baseado na versão 6 do padrão, conhecido como ES2015 (KRILL, 2015).

O uso de JavaScript em páginas da Web interfere no comportamento dinâmico da mesma. Uma página da Web é por natureza estática. Servidores Web são capazes de construir páginas Web, no entanto essa construção acontece de uma única vez. Depois que a página está pronta, ela é apresentada ao usuário através de um navegador Web. Quando o usuário clica em um link, esta requisição leva a um novo processamento do servidor Web que, em seguida, retorna uma nova página para o usuário. Este paradigma utilizado em páginas Web tem sido alterado completamente com o aparecimento de alguns frameworks populares de JavaScript.

Com o uso de AJAX[2], por exemplo, tornou-se possível atualizar apenas partes das páginas web, sem a necessidade de recarregar a página inteira. Outro exemplo é a biblioteca de funções JQuery[3], que pode modificar dinamicamente qualquer elemento de uma página Web.

No contexto deste trabalho foi utilizado o framework **Vue.js** para desenvolvimento de algumas funcionalidades na camada de interface com o usuário. Schmitz (2016) define

[1] Applets são pequenas aplicações executadas em janelas do navegador Web.

[2] AJAX: Asynchronous JavaScript and XML. Permite fazer requisições ao servidor Web sem necessidade de recarregar a página inteira.

[3] JQuery: Uma biblioteca JavaScript capaz de manipular o Modelo de Objeto de Documento e realizar *requests* ao protocolo HTTP

Vue.js como um framework de linguagem JavaScript utilizada no desenvolvimento de interfaces web para criação de componentes Web reativos. Este framework provê uma estrutura de dados reativa e componentizada, possuindo também uma API (Application Programming Interface) bastante simples.

A principal proposta de *Vue.js* é transformar a camada visual da página em um sistema orientado a dados. *Vue.js* recorre à reatividade para montar uma estrutura em que os elementos da página estão diretamente conectados a uma camada de dados que controla todo seu comportamento. Isso permite que o programador se concentre na lógica da aplicação ao invés de ter que se preocupar em modificar determinado elemento.

6.6 Implementação do Agente Companheiro

O agente companheiro implementado no ambiente DEKSTRA foi desenvolvido de acordo com o conceito de agente reativo simples, explicado na seção 2.2.5.1. Seu objetivo é auxiliar o aluno no desenvolvimento de programas dentro do ambiente de concepção do DEKSTRA. As seções seguintes descrevem seu funcionamento.

6.6.1 Diagrama de Classes do agente companheiro

A figura 46 apresenta o diagrama de classes dos componentes que integram o sistema do agente companheiro. Aqui está representado toda estrutura e relações das classes que servem de modelo para os objetos do sistema do agente companheiro.

A classe Ambiente guarda o estado atual do ambiente de concepção. O estado atual é obtido através da percepção das ações do aluno. a codificação gerada pelo aluno é capturada do ambiente de concepção e repassada ao estado a partir da propriedade *xml-Percepção* (passado em formato XML).

As classes Meta, Plano e AcaoPrimitiva definem um estado do sistema. De acordo com o modelo do problema, um estado é formado por muitas metas, uma meta é formada por vários planos e um plano é formado por várias ações primitivas.

A classe Agente define os métodos ou ações do agente. Um agente deve verificar se uma meta já foi alcançada, deve listar as metas que ainda não foram cumpridas e deve executar o programa do agente que contém as regras que o ajuda a decidir que ação será executada.

6.6.2 Diagrama de atividade: Ação pedir ajuda

A decisão mais complexa para o agente é quando o usuário requisita sua ajuda para continuar resolvendo um determinado problema. Neste caso, o agente deve disponibilizar para o aluno uma questão pertinente ao problema que o aluno está resolvendo. Esta questão disponibilizada pelo agente é retirada do próprio modelo do problema, criado

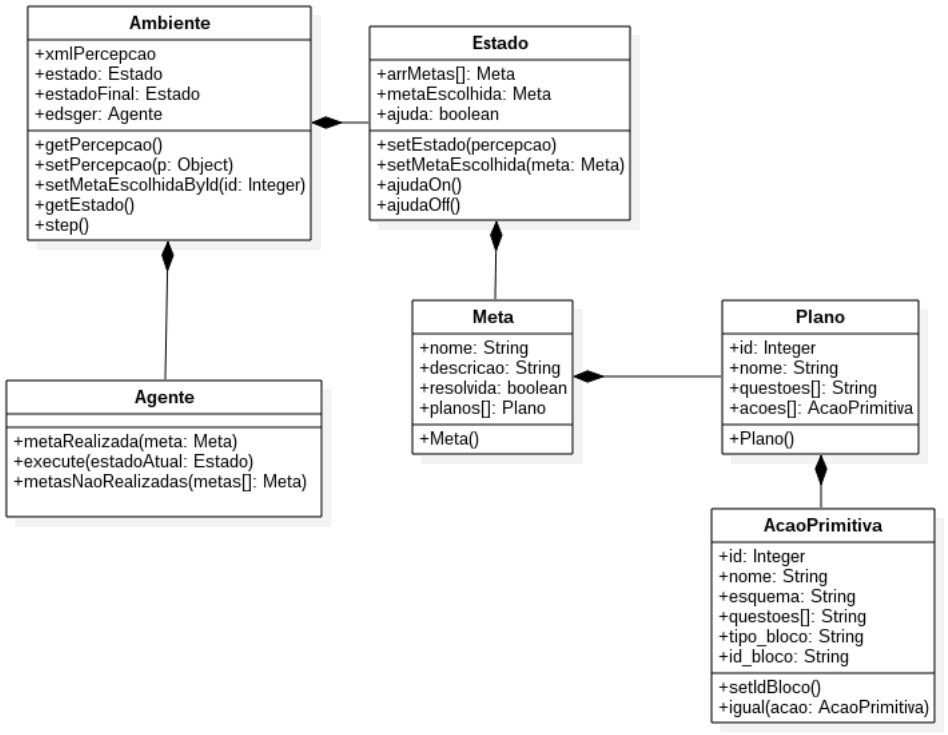


Figura 46 – Diagrama de Classe do agente reativo baseado em modelo

pelo professor. A figura 47 apresenta um fluxo de atividades que ocorre quando um aluno pede ajuda ao agente companheiro.

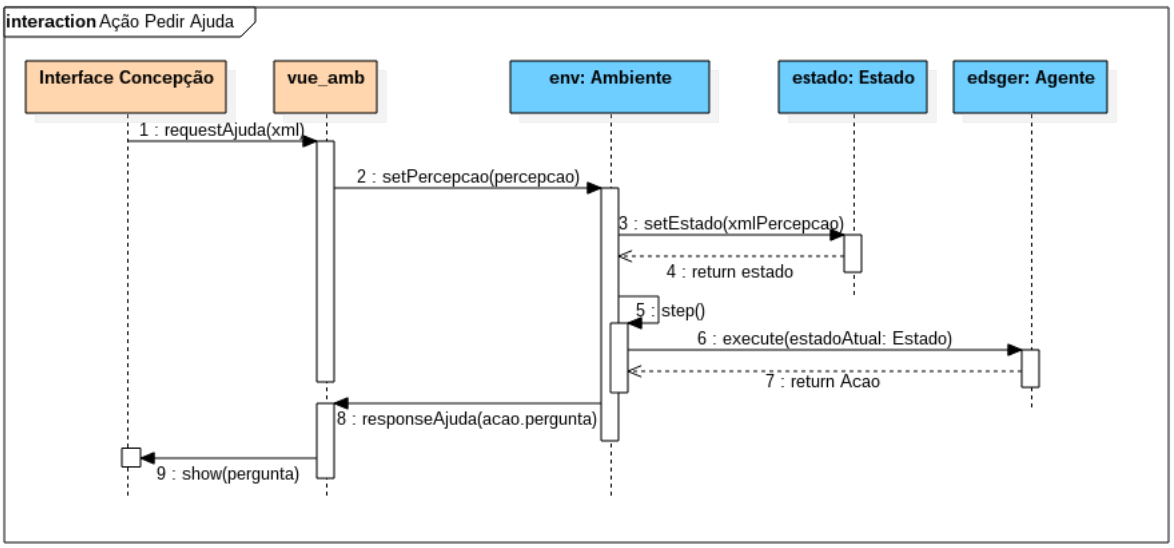


Figura 47 – Diagrama de atividade do agente reativo baseado em modelo - ação Pedir Ajuda

De acordo com a imagem, inicialmente o usuário solicita ajuda do agente através de um botão existente na interface do ambiente de concepção, gerando uma requisição de

ajuda (1) que é enviada para um objeto de controle reativo da interface, denominado de *vue_amb*. O objeto de controle *vue_amb* foi implementado utilizando o framework *vue.js*.

A requisição produzida pela interface entrega ao objeto *vue_amb* uma cópia do esquema criado pelo aluno até o momento da solicitação de ajuda. De posse deste esquema, o objeto de controle envia para o ambiente do agente uma percepção do estado atual do programa do aluno (2).

O estado do ambiente é atualizado (3 e 4) e o ambiente do agente companheiro pode finalmente executar o método *step()* (5), responsável por realizar uma consulta ao agente (6). Em seguida, o agente companheiro usará seu programa de agente para decidir que ação é mais adequada (7).

Percebendo que o usuário solicitou sua ajuda para solucionar o problema proposto pelo professor, o agente então retornará uma ação do tipo *"perguntar <questão>"*(7), enviando assim uma das questões que foram previamente cadastradas pelo professor. Portanto, é responsabilidade do programa do agente decidir que questão é mais adequada para ser exibida ao aluno em determinado momento.

6.7 Discussões

Este capítulo apresentou aspectos técnicos relacionado ao protótipo implementado do ambiente DEKSTRA. Foram apresentados os requistos funcionais e não-funcionais do sistema (6.2), o projeto arquitetural (6.3), a modelagem dos dados do sistema (6.4), uma descrição das tecnologias utilizadas (6.5) e alguns detalhes sobre a implementação do agente companheiro (6.6).

Dando continuidade à presente pesquisa, o próximo capítulo apresentará os materiais e métodos utilizados para a realização de uma oficina de ensino de programação utilizando a metodologia de ensino descrita no capítulo 4 e o ambiente DEKSTRA apresentado nos capítulos 6 e 5. Também serão apresentados os resultados obtidos através deste experimento.

7 EXPERIMENTO E ANÁLISE DOS RESULTADOS

Este capítulo apresenta um relato de experiência do uso do Ambiente de Aprendizagem Social DEKSTRA, que foi aplicado a um grupo de alunos do primeiro ano do curso médio de nível técnico em informática do Instituto Federal de Alagoas.

O principal objetivo deste experimento foi identificar quais efeitos o uso das ferramentas propostas nesta dissertação provocaria sobre este grupo de alunos em relação ao aprendizado dos conceitos elementares de programação de computadores.

Além do conjunto de esquemas de concepção presentes no ambiente DEKSTRA, também foi utilizado no experimento o modelo de suporte pedagógico proposto no capítulo 4.

7.1 Método para a Avaliação

Esta seção detalha como se procedeu todas as etapas do experimento proposto com o objetivo de verificar a eficácia dos esquemas de concepção aplicados para dar suporte ao ensino de programação e através do uso do ambiente DEKSTRA.

7.1.1 Seleção do grupo de alunos participante do experimento e divulgação

Como se esperava comprovar o grau de eficácia dos esquemas de concepção e do ambiente DEKSTRA, optou-se por envolver alunos sem experiência em programação. Sendo assim, os alunos selecionados corresponderam àqueles que cursavam o primeiro ano matutino do curso médio de nível técnico em informática do Instituto Federal de Alagoas.

A oficina foi planejada para acontecer entre os meses de maio e junho do ano de 2017, momento em que a turma selecionada estaria cursando o primeiro período avaliativo da disciplina Introdução a Programação, disciplina introdutória de programação deste curso.

A divulgação da oficina se deu por meio de convite realizado em sala de aula e divulgação em lista de discussão da turma. Foram ofertadas 40 vagas como atividade extracurricular e voluntária, das quais 37 foram preenchidas. As inscrições foram realizadas mediante preenchimento de formulário com informações básicas, tais como nome e e-mail. Essas informações foram utilizadas para criar uma conta de usuário para cada aluno no ambiente DEKSTRA.

7.1.2 Delimitação do conteúdo da oficina e planejamento das aulas

A oficina planejada abordou alguns conceitos iniciais da disciplina Introdução a Programação. Desta forma, utilizou-se 10 horas para a realização da oficina, distribuída em 5 aulas, ao longo de duas semanas e meia. Nas duas primeiras aulas foram ensinados os conceitos centrais de programação utilizando a abordagem por padrões elementares e as

demais foram utilizadas para exploração e resolução de problemas através do ambiente DEKSTRA.

A organização das aulas e sua distribuição na carga-horária da oficina estão ilustradas na tabela 7. Vale ressaltar que não era objetivo da oficina explorar todos os conceitos abordados em uma disciplina de introdução à programação, mas verificar se é possível produzir uma melhora na aprendizagem do aluno através da aplicação de um suporte pedagógico baseado nos esquemas de concepção proposto neste trabalho, juntamente com um ambiente virtual apropriado.

Tabela 7 – Distribuição dos tópicos e carga horária da oficina

Aula	**Conceitos Explorados**	**Carga horária**
01	Padrões Elementares de Programação	2h
02	Introdução ao ambiente DEKSTRA, apresentação das principais funcionalidades	2h
03	Variáveis, entrada, saída e atribuição	2h
04	Estruturas de controle de decisão	2h
05	Desenvolvimento de teste final para aferição de aprendizagem	2h
	Carga horária total	10h

De acordo com o método de ensino proposto no capítulo 4, algumas etapas devem ser seguidas para a aplicação do mesmo. Na *etapa de planejamento* foram escolhidos quais conceitos iniciais de programação seriam apresentados aos alunos. Em seguida, na *etapa de preparação*, foi necessário o cadastro de todos os problemas e dos respectivos planos de programação para que fosse possível disponibilizá-los no ambiente virtual.

As etapas 3 (*execução e socialização*) e 4 (*avaliação*) ocorreram durante as 04 primeiras aulas da oficina. A quinta e última aula foi planejada para que os alunos pudessem aplicar seus conhecimentos na resolução de quatro problemas finais, elaborados para avaliar se houve alguma melhora no aprendizado.

7.1.3 Instrumentos para avaliação do experimento

O primeiro instrumento de avaliação utilizado no experimento foi um questionário preliminar, que foi aplicado uma semana antes do início da oficina, com o objetivo de identificar, entre outras coisas, o grau de intimidade do o aluno com conceitos computacionais e, principalmente, se o aluno já tinha cursado alguma disciplina, oficina ou curso relacionado a algoritmos e programação.

Além do questionário, também foram elaborados testes para avaliar a capacidade de resolução de problemas dos alunos. Testes semelhantes foram realizados antes e na última aula da oficina (pré-teste e pós-teste). Foi solicitado aos alunos que respondessem as questões por meio de algoritmos usando pseudocódigo, até então, única forma conhecida

por eles para expressar a resolução de problemas computacionais. A tabela 8 mostra quais conceitos foram abordados em cada questão dos testes.

Além destes, um último questionário foi aplicado para medir o grau de satisfação dos alunos em relação à utilização do ambiente DEKSTRA na oficina. Este questionário procurou identificar as dificuldades que os alunos encontraram ao usar a ferramenta.

Tabela 8 – Conceitos abordados por questão

Conceitos	**01**	**02**	**03**	**04**
Variáveis (tipos de dados)	X	X	X	X
Entrada/Saída		X	X	X
Atribuição	X	X	X	X
Operadores	X	X	X	X
Estrutura de decisão	X		X	
Estrutura de repetição		X		
Compreensão/ leitura de algoritmo		X	X	

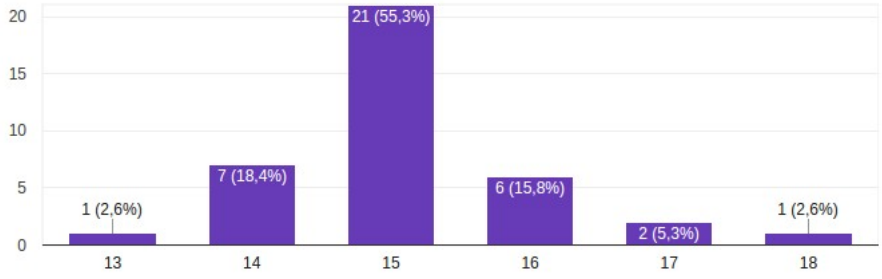
7.2 Resultados

De acordo com o que foi descrito na Seção 7.1, vários instrumentos de aferição foram utilizados durante o experimento. As próximas sub seções mostrarão os resultados obtidos nesta pesquisa.

7.2.1 Perfil do grupo de participantes

Mais de 50% dos participantes possuíam 15 anos no período em que responderam o questionário preliminar. A distribuição dos demais participantes em relação à idade é mostrado na figura 48.

Figura 48 – Distribuição dos participantes por idade



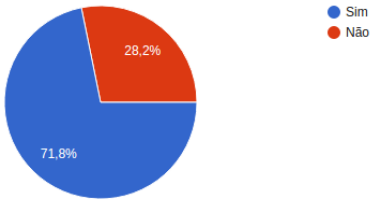
Fonte: Autor.

Para verificar o nível de experiência do participante em relação ao uso do computador, cinco perguntas foram feitas:

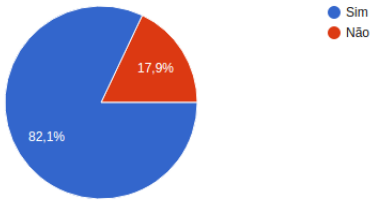
1. *Você tem computador em casa?*
2. *Tem utilizado o computador para trabalhos da escola?*
3. *Realizou ou realiza algum curso de informática básica fora do IFAL?*
4. *Estudou o curso fundamental em escola informatizada, que usava o computador nas aulas?*
5. *Tem facilidade no uso do computador?*

Com relação as duas primeiras perguntas, a figura 49 demonstra o resultado obtido. É importante notar que alguns participantes, mesmo sem possuir computador, ainda podem fazer uso de um computador da própria escola para realização dos trabalhos solicitados pelos professores. Desta forma, mesmo com 28,2% dos participantes não possuindo computador, somente 17,9% não utilizam computador em suas tarefas escolares.

Figura 49 – Experiência com computador



(a) Possui computador?

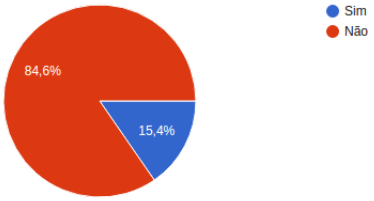


(b) Usa computador para trabalhos da escola?

Fonte: Autor.

Ainda procurando identificar a experiência prévia dos participantes quanto ao uso do computador, verificou-se também se os participantes tiveram contato com computador na escola de ensino fundamental que frequentaram. O resultado, como demonstrado na figura 50, é que pouco mais de 15% dos participantes estudaram em escola informatizada no ensino fundamental.

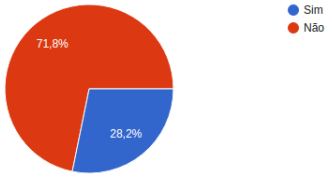
Figura 50 – Estudou em escola informatizada no ensino fundamental?



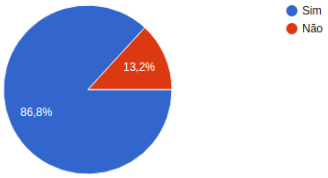
Fonte: Autor.

Também com relação à cursos técnicos de informática básica, a maioria informou que não realizou ou realiza curso de informática básica. No entanto, quase 87% dos entrevistados declararam ter facilidade no uso do computador (fig. 51).

Figura 51 – Afinidade com computador



(a) Realizou curso de informática básica fora do IFAL?

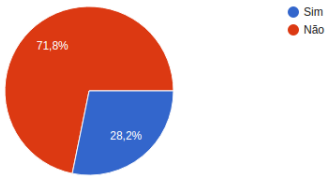


(b) Tem facilidade no uso do computador?

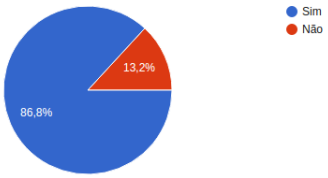
Fonte: Autor.

Com relação à experiência prévia com programação, a pesquisa procurou saber se o participante já programava antes de ingressar no IFAL, ou se já tinha estudado alguma disciplina de programação no IFAL antes. Conforme podemos ver na figura 52, a grande maioria nunca teve contato com programação antes desta oficina.

Figura 52 – Experiência prévia com programação



(a) Experiência prévia com programação



(b) Cursou disciplina de programação antes?

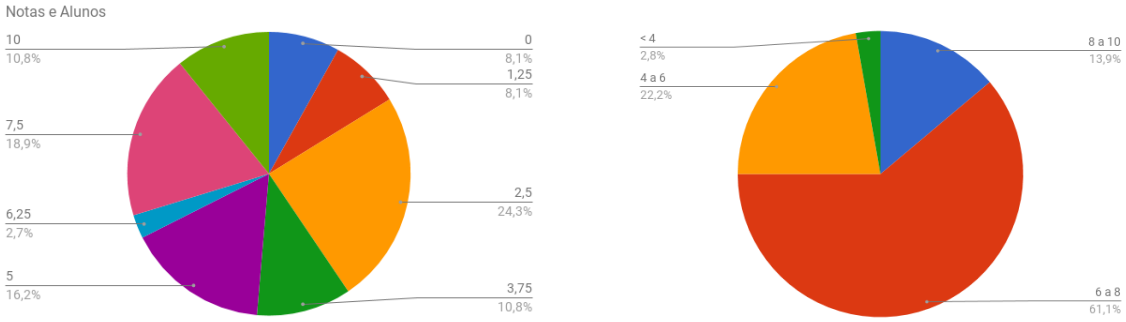
Fonte: Autor.

7.2.2 Desempenho nos testes de Algoritmos

Conforme dito anteriormente, antes da oficina foi aplicado um pré-teste nos participantes para averiguar o conhecimento prévio dos alunos. Como resultado, mais de 67%

dos estudantes obtiveram média abaixo de 6,00, conforme pode ser verificado na figura 53a. Neste gráfico, as médias dos alunos se encontram agrupadas. Nota-se, por exemplo, que quase 25% dos participantes obtiveram média 2,5 no pré-teste.

Figura 53 – Resultados do pré-teste e pós-teste

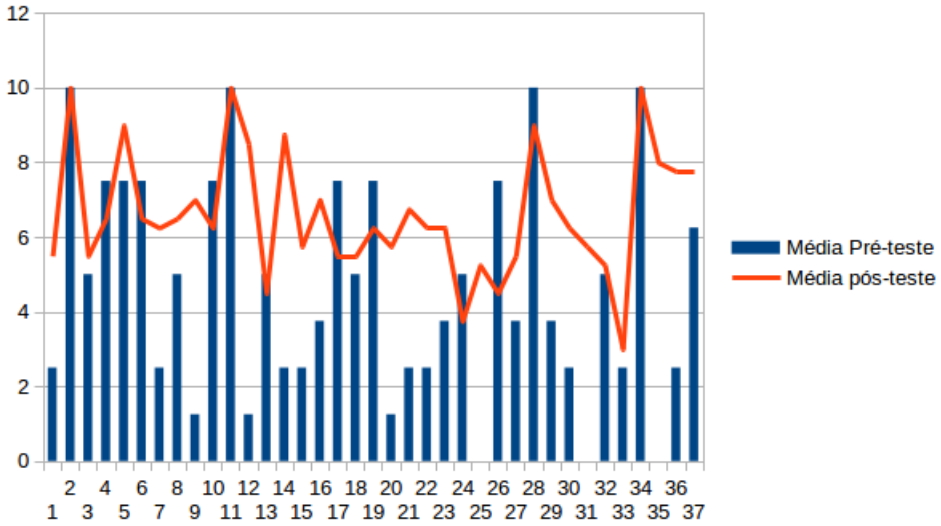


(a) Médias das notas dos alunos no pré-teste (b) Médias das notas dos alunos no pos-teste

Fonte: Autor.

Quanto aos resultados obtidos no pós-teste, o gráfico apresentado na figura 53b mostra que mais de 60% dos participantes alcançaram media entre 6 e 8 pontos. Outro ponto importante é a diminuição do número de participantes com média abaixo de 4, que foi de 2,8%. Como pode ser observado, é possível identificar uma melhoria em relação ao desempenho da maioria dos alunos no pós-teste se comparado ao pré-teste. O gráfico apresentado na figura 54 evidencia a melhora destes resultados.

Figura 54 – Comparativo entre as médias do pré-teste e pós-teste



Fonte: Autor.

7.3 Avaliação dos resultados

Apesar de não possuírem conhecimentos prévios de programação, a grande maioria dos participantes conseguiu apresentar soluções para todos os problemas propostos na oficina. As dúvidas que surgiam, geralmente sobre estruturas de decisão, eram trabalhadas a partir do uso de questões e planos de programação. Também era comum para os participantes se utilizarem das metas de programação, previamente informadas no módulo do professor.

Durante a oficina, foi perceptível o interesse dos alunos, que se motivaram a resolver os desafios apresentados através do ambiente DEKSTRA. Muitos participantes se mostraram empolgados em utilizar o ambiente interativo de concepção de programas disponibilizado pela ferramenta. Foi extremamente gratificante despertar o interesse dos alunos em programação através do uso de uma ferramenta lúdica.

No entanto, algumas dificuldades foram enfrentadas como, por exemplo, a infraestrutura da escola onde foi realizada a oficina. Algumas aulas foram prejudicadas pela inconsistência de disponibilidade de Internet, uma vez que o ambiente DEKSTRA funciona em plataforma WEB. Para evitar maiores transtornos, foi solicitado uma atenção especial para a equipe técnica de suporte computacional da instituição. Ainda assim, ocorreram momentos de falha durante a oficina, ocasionando, inclusive, perda de alguns projetos dos alunos.

Porém conclui-se, a partir das avaliações dos estudantes que a oficina foi propícia para o aprendizado, estimulante e dinâmica, além de ter favorecido o aumento do interesse na área de computação.

8 CONSIDERAÇÕES FINAIS

Este trabalho procurou conhecer quais as principais causas que levam ao alto índice de reprovação em disciplinas introdutórias de programação. Visando a solução do problema, diversos trabalhos, tais como Koliver, Dorneles e Casa (2004), Almeida et al. (2002), Beaubouef e Mason (2005) e Silva, Cavalcante e Costa (2013), apontam para dificuldades enfrentadas pelos alunos na aprendizagem dos conceitos fundamentais relacionados com construção de algoritmos.

Muitas pesquisas procuram apresentar justificativas para esta insuficiência no aprendizado de programação, como por exemplo, Koliver, Dorneles e Casa (2004) que explicam que muitas vezes o professor inicia a disciplina com uma perspectiva equivocada a respeito das habilidades de seus alunos. Outro problema citado é a forte carga de conceitos abstratos que permeiam todo esse conhecimento envolvido na atividade de programação (ALMEIDA et al., 2002).

Diversos métodos de ensino de programação foram propostas ao longo dos anos. Sanches (2002), por exemplo, propõe que o ensino de algoritmos seja feito utilizando o paradigma Orientado a Objeto. Noschang et al. (2014) sugerem o uso de uma ferramenta de programação onde os alunos podem implementar e testar suas soluções lógicas utilizando a linguagem Portugol. Rapkiewicz et al. (2006) defendem o uso de jogos computacionais como uma estratégia pedagógica de ensino de programação.

Do ponto de vista do professor, muitos questionamentos tem sido realizados em relação à complexidade do ensino de programação. As principais dúvidas que eventualmente surgem são: "que linguagem de programação pode ser usada?", ou "Que ferramentas e ambientes podem suportar a aprendizagem de programação? e de que maneira?", ou ainda "Que ferramentas de suporte ao ensino podem ser utilizadas de forma que ajude a alcançar os resultados esperados?".

O trabalho de Pears et al. (2007) procurou catalogar e classificar diversos estudos sobre o ensino e aprendizagem de programação, demonstrando que diversas pesquisas acabam por propor uma das seguintes coisas: uma metodologia de ensino para programação ou uma ferramenta computacional para ensino de programação, ou uma combinação das duas coisas.

Durante a realização deste trabalho foi possível estudar e analisar vários métodos de ensino de programação. Também foi possível verificar diversos que ferramentas cognitivas eram utilizadas em diversos ambientes interativos de aprendizagem.

De forma a seguir um raciocínio coerente, optamos por descrever um conjunto de esquemas de concepção para ajudar o aluno iniciante a aprender a programar. Em seguida, tal conjunto de esquemas de concepção foi inserido em um método de ensino que foi elaborado em consonância com os conceitos de aprendizagem cognitiva.

8.1 Considerações Finais

Foi nesta direção que propusemos uma integração do conjunto de esquemas de concepção a um ambiente de aprendizagem social, denominado DEKSTRA, construído para facilitar a utilização tanto do método de ensino quanto do conjunto de esquemas de concepção criado. Em Paraguaçu (1997) foi possível encontrar uma descrição mais detalhada sobre aprendizagem social de forma a se relacionar com a teoria histórico-cultural de Vygotsky (2007) sobre Zonas de Desenvolvimento Proximal.

O método de propõe que o professor trabalhe o conteúdo da disciplina, classificando-o de acordo com o nível de complexidade, e disponibilize para o aluno uma série de problemas de programação. Cada problema de programação deverá possuir um modelo mental do problema, que mapeia tanto as metas para resolução do problema quanto um conjunto de planos de programação que o aluno deverá conhecer para criar uma solução para este problema de programação.

Por fim, a metodologia proposta também aponta para a necessidade de se criar questões para cada plano de programação e para cada ação primitiva que compõe um plano de programação. Tais questões serão importantes para orientar o aluno sobre que caminho ele deve seguir e serão utilizadas pelo agente companheiro quando for solicitado pelo aluno.

É no ambiente DEKSTRA que tanto o professor quanto o aluno encontrarão suporte para a realização de suas atribuições. O Módulo do professor é projetado para facilitar a criação e edição de problemas de programação, incluindo a criação do modelo mental do programa (esquemas de concepção). O módulo do aluno objetiva oferecer tarefas interativas, onde o aprendiz possa manipular ferramentas cognitivas para construir uma solução para o problema proposto pelo professor.

A arquitetura propõe três agentes: o companheiro artificial, o aprendiz humano e professor humano. Seus papéis são descritos na arquitetura e tem por objetivo estimular o processo interativo no ambiente, permitindo que o aprendiz possa interagir com outros alunos, o professor e com o agente companheiro no momento da realização da tarefa.

Para procurar avaliar os efeitos tanto do método de ensino proposto quanto do protótipo de ambiente de programação implementado neste trabalho, uma oficina foi realizada ao longo da execução de um curso de programação básica, em disciplina introdutória de programação. Para tanto, a oficina adotou o ensino de programação por meio de padrões elementares de programação, aplicando todas as etapas previstas para a metodologia de ensino discutida no capítulo 4.

Neste experimento, os alunos aprenderam nas aulas teóricas os padrões elementares e problemas exemplificados pelo professor e nas aulas práticas usaram o ambiente DEKSTRA para solucionar os problemas sugeridos pelo professor, usando uma linguagem de programação visual. Os resultados do experimento realizado mostraram uma melhora no aprendizado do aluno, principalmente naqueles casos onde o aluno programava pela

primeira vez.

8.2 Trabalhos Futuros

Como sugestão para trabalhos futuros, julga-se fundamental efetuar-se um experimento mais prolongado e meticuloso, com objetivo de obtenção de resultados para a análise do nível de auxílio que pode ser fornecido pelo ambiente na resolução de problemas de programação por meio da aplicação da metodologia cognitiva proposta.

Outra atividade que pode dar continuidade a este trabalho pode ser a melhora nos aspectos de interface. No Módulo do aluno, por exemplo, os aspectos na interface do DEKSTRA podem ser aprimorados, conforme sugestões que podem ser apresentadas por participantes de avaliações futuras. Tais melhorias podem ser implementadas visando uma melhor aceitação do aluno no uso do ambiente.

Referências

ALMEIDA, E. S. d. et al. Ambap: Um ambiente de apoio ao aprendizado de programação. **X Workshop sobre Educação em Informática.**, 2002.

ANTUNES, C. **Novas maneiras de ensinar, novas formas de aprender**. Porto Alegre: Artmed Editora, 2007. ISBN 9788536313139. Disponível em: <http://books.google.com.br/books?id=4zHtUMM_z7oC>.

ASTRACHAN, O.; WALLINGFORD, E. Loop patterns. **Patterns Languages of Programs**, 1998. Disponível em: <https://users.cs.duke.edu/~ola/patterns/plopd/loops.html>.

BATALHA, G. d. S. **O uso de compilador em ambiente de aprendizagem de algoritmos**. Dissertação (Mestrado) — Universidade Estácio de Sá, 2008.

BEAN, M. **Laravel 5 Essentials**. [S.l.]: Packt Publishing Ltd, 2015.

BEAUBOUEF, T.; MASON, J. Why the high attrition rate for computer science students: some thoughts and observations. **ACM SIGCSE Bulletin**, v. 37, n. 2, p. 103–106, Jun. 2005.

BERGIN, J. **Simple Design Patterns**. 1998. Disponível em: <http://csis.pace.edu/~bergin/papers/SimpleDesignPatterns.html>.

_____. Patterns for selection version 4. 1999. Disponível em: <http://csis.pace.edu/~bergin/patterns/Patternsv4.html>.

BEZERRA, P. T.; SCHIMIGUEL, J. Desenvolvimento de aplicações mobile cross-platform utilizando phonegap. **Revista Observatorio de la Economía Latinoamericana**, 2016. Disponível em: <http://www.eumed.net/cursecon/ecolat/br/16/phonegap.html>.

BONAR, J.; SOLOWAY, E. Preprogramming knowledge: A major source of misconceptions in novice programmers. **Human-Computer Interaction**, v. 1, n. 2, p. 133–161, 1985. Disponível em: <http://dblp.uni-trier.de/db/journals/hhci/hhci1.html#BonarS85>.

BORDINI, R. H.; VIEIRA, R.; MOREIRA, Á. F. Fundamentos de sistemas multiagentes. In: **Anais do XXI Congresso da Sociedade Brasileira de Computação (SBC2001)**. [S.l.: s.n.], 2001. v. 2, p. 3–41.

BRASIL. **Lei nº 12.527, de 18 de novembro de 2011.** http://www.planalto.gov.br/ccivil_03/_ato2011-2014/2011/lei/l12527.htm: Diário Oficial [da] República Federativa do Brasil, 2011.

BRETON, P. **História da Informática**. São Paulo: [s.n.], 1991. P. 68-69. ISBN 85-7139-021-5.

BROOKS, R. Towards a theory of the comprehension of computer programs. In: **International Journal of Man-Machine Studies**. [S.l.: s.n.], 1983. p. 543–554.

BRUSILOVSKY, P.; OUTROS. Teaching programming to novices: A review of approaches and tools. In: WORLD CONFERENCE ON EDUCATIONAL MULTIMEDIA AND HYPERMEDIA. **Educational Multimedia and Hypermedia,**. Vancouver, 1994. Disponível em: <http://files.eric.ed.gov/fulltext/ED388228.pdf>.

CHAN, T. W. Integration-kid: A learning companion system. In: ORGANIZATION, I. J. C. on A. I. (Ed.). [S.l.: s.n.], 1991. v. 2.

CHAN, T.-W. Artificial agents in distance learning. **International Journal of Educational Telecommunications**, Association for the Advancement of Computing in Education (AACE), Charlottesville, VA, v. 1, n. 2, p. 263–282, 1995. ISSN 1077-9124. Disponível em: <https://www.learntechlib.org/p/15163>.

CLANCY, M. J.; LINN, M. C. Patterns and pedagogy. In: **Proceedings of the SIGCSE 1999 - Technical Symposium on Computer Science Education,**. [S.l.]: ACM Press, 1999. p. p. 37–42.

COLLINS, A. The cambridge handbook of the learning sciences. In: _____. New York: Cambridge University Press, 2006. cap. Cognitive Apprenticeship.

COSTA, C. J. d. S. A.; PARAGUAÇÚ, F.; PINTO, A. d. C. Experiências interativas com ferramentas midiáticas na tutoria on-line. **Em aberto**, v. 22, n. 79, p. p. 121–137, Jan 2009.

CROCHIK, J. **Computador No Ensino E a Limitacao Da Consciência**. Casa do Psicólogo, 1998. ISBN 9788573960389. Disponível em: <http://books.google.com.br/books?id=mhKnKJS-1V4C>.

DEACON, J. Model-view-controller (mvc) architecture. 2009. Disponível em: <http://www.jdl.co.uk/briefings/MVC.pdf>.

DELGADO, C. et al. Identificando competências associadas ao aprendizado de leitura e construção de algoritmos. In: XIII WEI. **Anais do XXV Congresso da Sociedade Brasileira de Computação**. [S.l.], 2005. p. p. 2371–2382.

DENHADI, S. **A Cognitive Study of Learning to Program in Introductory Programming Courses**. Tese (Doutorado) — Middlesex University - School of Engineering and Information Sciences, 2009.

DENNEN, V. P.; BURNER, K. J. Handbook of research on educational communications and technology. In: _____. [S.l.]: Taylor & Francis US, 2008. cap. The Cognitive Apprenticeship Model in Educational Practice, p. 425–439.

DERSHEM, H. L.; JIPPING, M. J. **Programming Languages. Structures and models**. 2nd. ed. Boston: PWS Publishing Company, 1995.

DIJKSTRA, E. **On the Teaching of Programming, i.e. on the Teaching of Thinking**. [S.l.: s.n.], 1975.

DIJKSTRA, E. W. Go to statement considered harmful. **Comm. ACM**, v. 11, n. 3, p. 147–148, 1968. Letter to the Editor.

_____. **A Discipline of Programming**. New Jersey: Prentice Hall, 1976. ISBN 0-13-215871-X.

DILLENBOURG, P. Collaborative-learning: Cognitive and computational approaches. In: _____. [S.l.]: Elsevier, 1999. cap. Introduction: What do you mean by collaborative learning?, p. 1–19.

DILLENBOURG, P. et al. Learning in humans and machine: Towards an interdisciplinary learning science. In: _____. [S.l.]: SPADA, E. and REIMAN, P., 1996. cap. The evolution of research on collaborative learning, p. 189–211.

FARACO, R. A.; ROSATELLI, M. C.; GAUTHIER, F. A learning companion system for distance education in computer science. In: **Anais do XII WEI - Workshop de Educação em Computação**. Salvador: XXIV Congresso da Sociedade Brasileira de Computação, 2004.

FISCHER, A. E.; GRODZINSKY, F. **The Anatomy of Programming Languages**. New Jersey: Prentice Hall, 1993.

FRASER, N. et al. Blockly: A visual programming editor. **URL: https://developers.google.com/blockly/guides/overview**, 2013.

GOOGLE. **Blockly Games**. 2016. Acesso em: 07/10/2016. Disponível em: <https://blockly-games.appspot.com/about?lang=en>.

_____. **Maze**. 2016. Acesso em: 07/10/2016. Disponível em: <https://blockly-games.appspot.com/maze>.

GRIES, D. What should we teach in an introductory programming course? **SIGCSE Bull.**, ACM, New York, NY, USA, v. 6, n. 1, p. 81–89, jan. 1974. ISSN 0097-8418. Disponível em: <http://doi.acm.org/10.1145/953057.810447>.

HALVERSON, C. **Inside the Cognitive Workplace: New Technology and Air Traffic Control**. Tese (Doutorado) — University of California San Diego, 1995.

HUTCHINS, E. Book. **Cognition in the wild**. [S.l.]: MIT Press, Cambridge, Mass. :, 1995. xviii, 381 p. : p. ISBN 0262082314 0262581469 0262581469 0262082314 0262082314 0262581469.

HUTCHINS, E.; KLAUSEN, T. **Distributed Cognition in an Airline Cockpit. In Communication and Cognition at Work**. [S.l.: s.n.], 1996.

IEEE. **Computer Science Curricula 2013**. [S.l.], 2013. Disponível em: <https://www.acm.org/education/CS2013-final-report.pdf>.

JENNINGS, N.; WOOLDRIDGE, M. **Agent Technology: Foundations, Applications, and Markets**. Springer Berlin Heidelberg, 2012. ISBN 9783662036785. Disponível em: <https://books.google.com.br/books?id=AOwACAAAQBAJ>.

JOHNSON, W. L.; SOLOWAY, E. Proust: Knowledge-based program understanding. In: IEEE. **IEEE, Transactions on Software Engineering**. [S.l.], 1985. p. 369–380.

KOLIVER, C.; DORNELES, R. V.; CASA, M. E. Das (muitas) dúvidas e (poucas) certezas do ensino de algoritmos. In: XII WORKSHOP DE EDUCAçãO EM COMPUTAçãO (WEI 2004). **Anais do XXIV Congresso da Sociedade Brasileira de Computação. p. 949-960.** Salvador, 2004.

KRILL, P. **It's official: ECMAScript 6 is approved**. 2015. Acesso em 26/04/2017. Disponível em: <http://www.infoworld.com/article/2937716/javascript/its-official-ecmascript-6-is-approved.html>.

LEAL, A. V. d. A.; FERREIRA, D. J. Aplicando padrões de seleção no ensino de programação de computadores para estudantes do primeiro ano do ensino médio integrado. In: **X Encontro Anual de Computação - EnAComp**. [S.l.: s.n.], 2013. p. 3.

LEMOS, M. A. **Uma Abordagem Baseada em Padrões Elementares para Aprendizado de Programação**. Tese (Doutorado em Engenharia Elétrica) — Universidade de São Paulo, 2004.

LEMOS, M. A.; BARROS, L. N.; LOPES, R. D. Uma biblioteca cognitiva para o aprendizado de programação. In: **CONGRESSO NACIONAL DA SOCIEDADE BRASILEIRA DECOMPUTAÇÃO**. [S.l.: s.n.], 2003.

LEMOS, M. A.; LOPES, R. D.; BARROS, L. N. Avaliação do ensino-aprendizagem de programação usando uma abordagem baseada em padrões elementares de programação. In: ABENGE/UFCG - UFPE. **XXXIII Congresso Brasileiro de Ensino de Engenharia**. Campina Grande, 2005.

LETOVSKY, S. Empirical studies of programmers. In: _____. [S.l.]: Ablex Publishing Corporation, 1986. cap. Cognitive Processes in Program Comprehension, p. 58 – 79.

MANSO, A.; OLIVEIRA, L.; MARQUES, C. G. **Portugol IDE–Uma ferramenta para o ensino de programação**. [S.l.]: PAEE, 2009.

MARRON, A.; WEISS, G.; WIENER, G. A decentralized approach for programming interactive applications with javascript and blockly. In: **Proceedings of the 2Nd Edition on Programming Systems, Languages and Applications Based on Actors, Agents, and Decentralized Control Abstractions**. New York, NY, USA: ACM, 2012. (AGERE! 2012), p. 59–70. ISBN 978-1-4503-1630-9. Disponível em: <http://doi.acm.org/10.1145/2414639.2414648>.

MAYRHAUSER, A. von; VANS, A. M. **Program Understanding: A Survey**. [S.l.], 1994. Disponível em: <http://citeseer.nj.nec.com/vonmayrhauser94program.html>.

MCINTYRE, D. Comp.lang.visual - frequently-asked questions list. 1998. Disponível em: <ftp://rtfm.mit.edu/pub/usenet/news.answers/visual-lang/faq>.

MENDES, A. J. N.; GOMES, A. J. Suporte à aprendizagem da programação com o ambiente sicas. In: **V Congresso Iberoamericano de Informática Educativa - RIBIE**. Viña del Mar: [s.n.], 2000.

MYERS, B. A. Taxonomies of visual programming and program visualization. **jvlc**, v. 1, n. 1, p. 97–123, 1990.

NETSCAPE. **Netscape and Sun Announce JavaScript, the Open, Cross-Platform Object Scripting Language for Enterprise Networks and the Internet**. 1995. Acesso em 26/04/2017. Disponível em: <http://tech-insider.org/java/research/1995/1204.html>.

NOSCHANG, L. F. et al. Portugol studio: Uma ide para iniciantes em programação. In: INFORMáTICA, . Workshop sobre Educação em (Ed.). **Anais do Congresso Anual da Sociedade Brasileira de Computação**. Porto Alegre: SBC, 2014.

OLIVEIRA, M. K. d. **Vygotsky: aprendizado e desenvolvimento: um processo sócio-histórico**. São Paulo: Scipione, 1997.

OTWELL, T. Laravel documentation. **Laravel.[Online]**, 2016. Disponível em: <http://www.laravel.com>.

PACITTI, T.; AKTINSON, C. P.; TELES, A. A. d. S. **Programação e Métodos Computacionais**. 4. ed. Rio de Janeiro: [s.n.], 1983.

PAES, R. d. B. **Introdução a programação com a linguagem C**. São Paulo: [s.n.], 2016.

PAPERT, S. **Logo: computadores e educação.** São Paulo: Brasiliense, 1988.

PARAGUAÇU, F. **VYGOTSKY: un environnement d'apprentissage social pour la programmation fondé sur la collaboration entre agents d'aide à la conception par cas**. Tese (Tese de Doutorado) — Aix-Marseille 3, Grenoble, 1997. Th. : productique et informatique. Disponível em: <http://opac.inria.fr/record=b1064803>.

PARAGUAÇU, F. Fazendo a comunicação entre agentes artificiais colaborativos alfabetizadores e agentes humanos efetiva. In: MOURA, D. (Ed.). **Os múltiplos usos da língua**. Maceió: Edufal, 1999. p. 150–152.

PEARS, A. et al. A survey of literature on the teaching of introductory programming. In: **Working group reports on ITiCSE on Innovation and technology in computer science education**. New York, NY, USA: ACM, 2007. (ITiCSE-WGR '07), p. 204–223. Disponível em: <http://doi.acm.org/10.1145/1345443.1345441>.

PEñA-LóPEZ, I. **Personal Learning Environments and the revolution of Vygotsky's Zone of Proximal Development**. 2012. Acessado em: 02/03/2017. Disponível em: <http://ictlogy.net/20120831-personal-learning-environments-and-the-revolution-of-vygotskys-zone-of-proximal-develop>.

PHP.NET. **PHP documentation: History of PHP**. 2017. Acesso em: 23/04/2017. Disponível em: <http://php.net/manual/en/history.php.php>.

PIAGET, J. **The moral judgment of the child**. New York: The Free Press, 1965.

PICTORIUS, I. The home of visual object-oriented development environments. 1996. Disponível em: <http://www.pictorius.com/home.html>.

POLSON, M.; RICHARDSON, J. **Foundations of Intelligent Tutoring Systems**. L. Erlbaum Associates, 1988. (Interacting With Computers). ISBN 9780805800548. Disponível em: <http://books.google.com.br/books?id=gOlJgcpAuF4C>.

PORTER, R.; CALDER, P. Pattens in learning to program – an experiment? In: **Proceedings of the Sixth Australasian Conference on Computing Education**. [S.l.]: ACM Press, 2004. p. 241–246.

PPIG. **Psychology of Programming Interest Group**. 1987. Disponível em: <http://www.ppig.org/>.

PRASS, A. R. **Teorias de Aprendizagem**. [S.l.]: Universidade Federal do Rio Grande do Sul, 2007.

PREECE, J.; ROGERS, Y.; SHARP, H. **Design de Interacao**. Bookman, 2005. ISBN 9788536304946. Disponível em: <http://books.google.com.br/books?id=bl0H1cYIzAwC>.

Q-SUCCESS. **World Wide Web Technology Surveys**. 2017. Acesso em 22/04/2017. Disponível em: <https://w3techs.com>.

RAPKIEWICZ, C. E. et al. Estratégias pedagógicas no ensino de algoritmos e programação associadas ao uso de jogos educacionais. In: CICLO DE PALESTRAS NOVAS TECNOLOGIAS NA EDUCAçãO. **RENOTE : revista novas tecnologias na educação [recurso eletrônico]**. Porto Alegre, 2006. Disponível em: <http://hdl.handle.net/10183/22862>.

REIS, A. B. **Um ambiente para o agente pedagógico de aprendizagem colaborativa em harmonização ecológica - APACHE**. Dissertação (Mestrado) — Universidade Federal da Paraíba, Campina Grande, abril 2001.

RESNICK, M. et al. Scratch: Programming for all. **Commun. ACM**, ACM, New York, NY, USA, v. 52, n. 11, p. 60–67, nov. 2009. ISSN 0001-0782. Disponível em: <http://doi.acm.org/10.1145/1592761.1592779>.

RHEINFRANK, J.; EVENSON, S. Bringing design to software. In: WINOGRAD, T. (Ed.). New York, NY, USA: ACM, 1996. cap. Design languages, p. 63–85. ISBN 0-201-85491-0. Disponível em: <http://doi.acm.org/10.1145/229868.230034>.

RIBEIRO, S. L. d. C. Os múltiplos usos da língua. In: _____. [S.l.]: EDUFAL, 1999. cap. O papel da Informática na alfabetização colaborativa, p. 156–158.

ROBINS, A.; ROUNTREE, J.; ROUNTREE, N. Learning and teaching programming: A review and discussion. **Computer science education**, Taylor & Francis Group, v. 13, n. 2, p. 137–172, 2003.

ROCHA, P. S. et al. Ensino e aprendizagem de programação: Análise da aplicação de proposta metodológica baseada no sistema personalizado de ensino. **RENOTE**, v. 8, n. 3, 2010.

ROGERS, Y. A brief introduction to distributed cognition. **School of Cognitive and Computing Sciences**, 1997. Disponível em: <http://mcs.open.ac.uk/yr258/papers/dcog/dcog-brief-intro.pdf>.

ROOSEVELT, T. **Citizenship in a Republic**. 1910. Discurso proferido na Université Paris - Sorbonne, França. Acesso em 14 Out. 2016. Disponível em: <http://wallacecezar.wordpress.com/2010/11/24/o-homem-na-arena-cidadania-numa-repblica>.

RUSSELL, S. J.; NORVIG, P. **Artificial Intelligence: A Modern Approach**. 2. ed. [S.l.]: Pearson Education, 2003.

SANCHES, J. A. How to teach algorithmic reasoning. In: **Anais do IV Curso de Qualidade – Metodologia de ensino para cursos de graduação das áreas de Computação e Informática - Congresso da Sociedade Brasileira de Computação**. Florianópolis: [s.n.], 2002.

SANTOS, R. P.; COSTA, H. A. X. Análise de metodologias e ambientes de ensino para algoritmos, estruturas de dados e programação aos iniciantes em computação e informática. **INFOCOMP – JOURNAL OF COMPUTER SCIENCE, Lavras**, 2006.

SCHMITZ, D. **Conheça o Vue.js, um framework javascript para criação de componentes web reativos**. 2016. Disponível em: <https://tableless.com.br/conheca-o-vue-js-um-framework-javascript-para-criacao-de-componentes-web-reativos/>.

SEATTLER, P. **The Evolution of American Educational Technology**. Information Age Pub Incorporated, 2004. ISBN 9781593111397. Disponível em: <http://books.google.com.br/books?id=s1ThX561Z58C>.

SESU. **Diretrizes curriculares de cursos da área de Computação e Informática**. [S.l.], 1999. Secretaria de Educação Superior - SESu/MEC. Acesso em: agosto 2016. Disponível em: <http://www.mec.gov.br/sesu/ftp/curdiretriz/computacao/co_diretriz.rtf>.

SEVERANCE, C. Javascript: Desdesign a language in 10 days. **CSDL**, v. 45, n. 02, p. 7–8, 2012.

SHNEIDERMAN, B.; MAYER, R. Syntactic semantic interactions in programmer behavior: A model and experimental results. In: **International Journal of Computer and Information Sciences**. [S.l.: s.n.], 1979. v. 8, n. 3, p. 219–238.

SILVA, F. d. M. **Concepção e realização de um modelo computacional de jogos interativos no contexto da aprendizagem colaborativa**. Dissertação (Mestrado) — Universidade Federal de Alagoas, Maceió, 2008.

SILVA, M. T.; CAVALCANTE, M. C. T. C.; COSTA, E. B. Explorando correlações em programação: um estudo focado no processo seletivo e em disciplinas correlatas. In: SIMPóSIO BRASILEIRO DE INFORMáTICA NA EDUCAçãO. **Anais do XXIV Simpósio Brasileiro de Informática na Educação**. [S.l.], 2013.

SKINNER, B. **Tecnologia do ensino**. Epu, 1975. ISBN 9788512650203. Disponível em: <http://books.google.com.br/books?id=X8BqPwAACAAJ>.

SKINNER, B.; VILLALOBOS, M. da P. **Sobre O Behaviorismo**. Cultrix, 1974. ISBN 9788531603600. Disponível em: <http://books.google.com.br/books?id=gej5uGOa_OkC>.

SKVORC, B. **Best PHP Frameworks for 2014**. 2013. Acesso em: 22/04/2017. Disponível em: <http://www.sitepoint.com/best-php-frameworks-2014>.

SOLOWAY, E. Learning to program = learning to construct mechanisms and explanations. **Commun. ACM**, ACM, New York, NY, USA, v. 29, n. 9, p. 850–858, set. 1986. ISSN 0001-0782. Disponível em: <http://doi.acm.org/10.1145/6592.6594>.

SOLOWAY, E.; ADELSON, B.; EHRLICH, K. Knowledge and processes in the comprehension of computer programs. In: **The Nature of Expertise**. [S.l.: s.n.], 1988.

SOLOWAY, E.; EHRLICH, K. Empirical studies of programming knowledge. **IEEE Trans. Software Engineering**, IEEE Computer Society Press, SE-10, n. 5, p. 595–609, September 1984.

SUPPES, P. Some theoretical models for mathematics learning. **Journal of Research and Development in Education**, v. 1, p. 5–22, 1967.

TEIXEIRA, J. d. F. **Mentes e Máquinas: uma introdução à ciência cognitiva**. Porto Alegre: Artes Médicas, 1998. ISBN 9788573073294. Disponível em: <http://books.google.com.br/books?id=FHnAXY5fHqoC>.

THE HILLSIDE GROUP. **ChiliPLoP: Southwestern Conference on Pattern Languages of Programs**. 1998. Acesso em 21 out 2016. Disponível em: <http://hillside.net/conferences/chili-plop>.

TORRES PATRíCIA; ALCANTARA, P. R. L.; IRALA, E. A. F. Grupos de consenso: Uma proposta de aprendizagem colaborativa para o processo de ensino-aprendizagem. **Revista Diálogo Educacional**, v. 4, 2004. Disponível em: <http://redalyc.uaemex.mx/src/inicio/ArtPdfRed.jsp?iCve=189117791011.>

UNIVERSITY OF NORTHERN IOWA. **The Elementary Patterns Home Page**. Iowa, 2005. Homepage de padrões elementares, integra a comunidade de pesquisadores, trabalhos realizados e em andamento na área. Acesso em: 14 out. 2016. Disponível em: <http://www.cs.uni.edu/~wallingf/patterns/elementary/>.

URBAN-LURAIN, M. Intelligent tutoring systems: An historic review in the context of the development of artificial intelligence and educational psychology. 1996. Disponível em: <http://www.cse.msu.edu/rgroups/cse101/ITS/its.htm>.

VEER, R. van der; VALSINER, J. **Vygotsky: uma síntese**. Unimarco, 1996. ISBN 9788515012756. Disponível em: <https://books.google.com.br/books?id=eTjmewWdpNAC>.

VIRGINIA POLYTECHNIC INSTITUTE AND STATE UNIVERSITY. **Intro to Computational Thinking: Algorithms - Intro Blockly**. 2016. Acesso em: 07/10/2016. Disponível em: <https://vt.instructure.com/courses/19969/pages/book-3-dot-2-algorithms-intro-blockly>.

VYGOTSKY, L. The problem of the cultural development of the child. In: **The Vygotsky Reader**. Oxford: Blackwell, 1994.

VYGOTSKY, L. S. **A formação social da mente**. São Paulo: Martins Fontes, 2007.

WALLINGFORD, E. Elementary patterns and their role in instruction. In: **OOPSLA'98 Educators Symposium Notes**. [S.l.: s.n.], 1998.

WEBER, G.; BRUSILOVSKY, M. S.; STENLE, F. Elm-pe: An intelligent learning environment for programming. 1996. Disponível em: <http://www.psychologie.uni-trier.de:8000/projects/ELM/ELM-PE/tour.html>.

WEXELBLAT, R. L. **History of Programming Languages**. New York: Academic Press, 1981. P. 6-15. ISBN 0-12-745040-8.

WINOGRAD, T. **Bringing Design to Software**. [S.l.]: ACM Press, 1996.

WOOD, D. J.; BRUNER, J. S.; ROSS, G. The role of tutoring in problem solving. **Jorunal of Child Psychiatry and Psychology**, v. 17, n. 2, p. 89–100, 1976.