Pontifícia Universidade Católica de São Paulo
PUC-SP

Bruno de Oliveira

Ambiente de aprendizagem de programação imperativa no ensino técnico: Uma proposta de ensino em espiral

Mestrado em Tecnologias da Inteligência e Design Digital

São Paulo
2021

Bruno de Oliveira

Ambiente de aprendizagem de programação imperativa no ensino técnico: Uma proposta de ensino em espiral

Mestrado em Tecnologias da Inteligência e Design Digital

Dissertação apresentada à banca Examinadora da Pontifícia Universidade Católica de São Paulo, como exigência parcial para obtenção do título de MESTRE em Tecnologias da Inteligência e Design Digital, na área de concentração Processos Cognitivos e Ambientes Digitais, na linha de pesquisa Modelagem de Sistemas de Software, sob a orientação do Prof. Dr. Ítalo Santiago Vega.

São Paulo
2021

BANCA EXAMINADORA

_____

_____

_____

_____

_____

Agradecimentos

Dedico este trabalho aos meus alunos do ensino técnico, os quais sempre se demonstraram sinceros expressando suas dificuldades e comentando seus pensamentos enquanto aprendiam programação. Foram muitas conversas dentro e fora da sala de aula, em grupos de estudo à noite e aos finais de semana, desde aqueles que tinham certa predisposição até os que tinham alguma dificuldade. A abertura e confiança recíproca permitiu-me entender suas ansiedades, angústias, frustrações e inseguranças, bem como sua perseverança, resiliência, dedicação, esforço e vontade de seguir em frente.

Agradeço a Deus pelo dom da vida e a minha família que me deu força necessária em toda trajetória da pesquisa, em que muitas vezes senti-me fatigado – física e mentalmente.

A meu orientador que abriu meus olhos para um mundo novo da pesquisa científica e me fez repensar a importância da academia para o crescimento social, além das diversas conversas inesquecíveis sobre programação.

Sou grato a todos amigos programadores e educadores que doaram seu tempo em ouvir minhas ideias e a tecer considerações riquíssimas que motivaram ainda mais a conclusão desse trabalho.

Resumo

O ensino de programação enfrenta, desde sua concepção, desafios em seu processo de aprendizagem, dentre os quais estão o alto índice de evasão, a falta de motivação dos alunos e a dificuldade na compreensão dos conceitos fundamentais. No ensino técnico acrescenta-se a falta de entendimento quanto aos objetivos do curso, a incompletude de conhecimentos primários fornecidos no ensino básico e a carência de experiências prévias na área de programação e construção de algoritmos. O objetivo desta pesquisa originou-se na investigação do processo de aquisição de uma linguagem de programação por novatos, estabelecendo um modelo teórico que contemple os fundamentos da programação categorizados por sua complexidade e propor um ensino em espiral onde seja possível estabelecer o estágio de aprendizagem em que o novato se encontra. No campo da computação, foram examinados trabalhos acadêmicos sobre as dificuldades e propostas de ensino em programação, e obras que remetem aos fundamentos da programação em seu aspecto conceitual teórico. No que diz respeito à educação, foram examinadas as teorias da aprendizagem significativa e do currículo em espiral, assim como as teorias gerais da construção do conhecimento. Como resultado, obteve-se a estrutura TFR (Tipo, Função, Rota), a qual descreve as principais atividades realizadas no momento da programação. A TFR explicita os fundamentos da programação imperativa e categoriza em cada um de seus pilares os graus de complexidade correspondentes a seus conceitos intrínsecos. Através de uma proposta de ensino em espiral aplicada a estrutura TFR, é possível identificar, no processo de aquisição de uma linguagem de programação, o estágio em que o novato se encontra, além de possibilitar discussões sobre o planejamento das aulas, a sequência de ensino e formas de apresentação e avaliação dos conceitos fundamentais da programação.

Palavras-chave: Ensino de Programação. Estrutura TFR. Aprendizagem Significativa. Ensino em Espiral. Fundamentos da Programação. Paradigma Imperativo. Novato. Ensino Técnico.

Abstract

Programming teaching has faced challenges in its learning process since its conception. Among these challenges, there are the high dropout rate, the students' lack of motivation and the difficulty in understanding the fundamental concepts. To the technical education it is possible to add the lack of understanding of the course objectives, the incompleteness of primary knowledge provided by the basic education and the lack of previous experiences in programming and building algorithms. This research aims at investigating the process of acquiring a programming language in novices, establishing a theoretical model that contemplates the fundamentals of programming categorized by its complexity and proposing a spiral teaching where it is possible to establish the learning stage that the novice finds themself. Fundamentals of programming in their theoretical conceptual aspect were examined in the field of computing as well as the academic papers on the difficulties and teaching proposals in programming. In the field of education, theories of meaningful learning and the spiral curriculum were examined, as well as general theories of knowledge construction. As a result, the TFR structure (Type, Function, Route) was obtained, which describes the main activities carried out at the time of programming. TFR explains the fundamentals of imperative programming and categorizes in each of its pillars the degrees of complexity corresponding to its intrinsic concepts. Through a spiral teaching proposal applied to the TFR structure, it is possible to identify, in the process of acquiring a programming language, the stage in which the novice is, in addition to enabling discussions about the planning of classes, the sequence of teaching, and ways of presenting and evaluating the fundamental concepts of programming.

Keywords: Programming Teaching. TFR structure. Meaningful Learning. Spiral Teaching. Fundamentals of Programming. Imperative Paradigm. Novice. Technical education.

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

CBO	Classificação Brasileira de Ocupação
CNCT	Cartilha Nacional de Cursos Técnicos
CS	Computer Science
INSF	Instituto Social Nossa Senhora de Fátima
LDBEN	Lei de Diretrizes e Bases da Educação Nacional
MEC	Ministério da Educação
TFR	Tipo, Função Rota
TI	Tecnologia da Informação

Sumário

PREÂMBULO

A educação é o meio mais eficaz para transformar uma sociedade, melhor dizendo, uma vida. As oportunidades de estudo muitas vezes não chegam a todos os pontos da cidade, ou quando chegam, padecem de qualidade por motivos variados, desde a infraestrutura precária até a própria descrença de poder sonhar com algo melhor.

Em 2005, aos 15 anos, fiz o curso Técnico em Informática em uma instituição sem fins lucrativos, onde ficaram claros os caminhos profissionais que gostaria de seguir: a programação de computadores. Cursei o ensino superior, concluí algumas especializações e abri uma microempresa na área de tecnologia, em paralelo, realizava aulas voluntárias de informática básica para jovens e adultos. O gérmen da educação instalara-se fortemente em mim.

Retornei como visitante à escola técnica que me abriu as portas na juventude e, de forma inesperada, fui convidado a ministrar as disciplinas de lógica e linguagem de programação. Com muita energia e disposição, agora como professor, poderia retribuir aos novos jovens, as possibilidades que também foram dadas a mim. No entanto, mesmo com 10 anos de experiência desenvolvendo *software* e certificações em diversas tecnologias, deparei-me com lacunas teóricas que me impossibilitavam visualizar com profundidade todos os fenômenos envolvidos no processo de aprendizagem de programação. De fato, desde sua concepção, o ensino de programação é conhecido por muitas dificuldades pedagógicas, evasões, falta de motivação e dificuldades de aprendizagem de seus conceitos fundamentais. Era necessário pois, preencher as lacunas e realizar novos estudos para que de alguma forma, pudesse contribuir efetivamente para a formação profissional desses jovens cuja esperança trazem desde tão tenra idade.

Agradeço desde já a paciência do leitor por acompanhar os pequenos passos que consegui conquistar, mas que trouxeram grandes resultados durante as aulas de programação no ensino técnico no qual ainda me encontro.

1 INTRODUÇÃO

O ensino de programação é uma preocupação de todos nós professores que lecionamos as disciplinas introdutórias como lógica de programação, algoritmos, estrutura de dados e outras que variam pelo nome. Observamos pela experiência dois aspectos sempre presentes nas turmas que se renovam semestralmente ou anualmente: dificuldade na compreensão e aplicação dos conceitos fundamentais da programação e desmotivação dos alunos em propor soluções para tarefas computacionais.

A linguagem de programação é uma particularidade das linguagens artificiais formais, com o objetivo de permitir que dispositivos computacionais executem tarefas em seu meio, resultando em um programa de computador. Toda linguagem de programação possui sintaxe e semântica semelhante às linguagens naturais, porém a analogia não deve ser levada muito adiante, já que é necessário que seja implementável em um computador e seja universal, no sentido de que cada problema deve ter uma solução que possa ser escrita pela linguagem. Ademais, uma linguagem de programação possui seu aspecto pragmático, no que diz respeito à forma que ela é usada na prática (WATT, 2004).

Programar, nesse sentido, diz respeito à capacidade do programador planejar as ações que devem ser feitas, incluindo desvios e repetições comuns à execução de tal tarefa, prevendo com alto grau de assertividade, precisão e detalhe, o processo que deve ser seguido para o sucesso da tarefa.

Para o ensino técnico de nível médio, há ainda a falta de visão dos alunos sobre a área de TI e o choque perceptível quando mudam de lado, isto é, de consumidores para criadores de tecnologia. Em sua maioria, pensam que o curso abrangerá o pacote *office* e como formatar e trocar as peças de um computador. Os que pensam em programação normalmente estão com grandes expectativas na criação de jogos computacionais com seus aparatos 2D e 3D, na criação de cenários e personagens. Poucos trazem a ideia de que podem criar seus próprios aplicativos de celular, sistemas *web*, interagir com tecnologias já existentes e criar sua própria empresa no futuro.

Muitos estudantes relatam estar cursando o ensino técnico por exigência dos pais, para não ficarem com suas tardes vazias ou por terem ouvido falar que é uma área que paga bem. Em uma visão pedagógica, identificamos muitos com déficits de

aprendizagem vindos do ensino fundamental. No geral, mesmo com a possibilidade de conseguirem um estágio ao final do curso, a mentalidade de que estão se capacitando profissionalmente para atuarem no mercado de trabalho não é bem compreendida.

Revela-se a necessidade de profissionais da programação de computadores para que sustentem a demanda dessa área que continua a crescer e a prover novas ferramentas e tecnologias. Um estudo realizado pela Associação Brasileira das Empresas de Tecnologia da Informação e Comunicação (BRASSCOM, 2019) diz que em 2024 o setor de tecnologia demandará 420 mil novos profissionais.

No entanto, o que se percebe é a falta de profissionais capacitados para exercerem as funções que o mercado de trabalho busca e muitas das vagas abertas não são preenchidas por falta de capacitação (BRASSCOM, 2020). Outro fator a considerar é a evasão dos alunos nos cursos de tecnologia voltados à programação, um levantamento realizado pelo Sindicato das Entidades Mantenedoras de Estabelecimentos de Ensino Superior no Estado de São Paulo (SEMESP, 2013) aponta que os cursos da área da tecnologia da informação possuem maior taxa de evasão, apenas um a cada três alunos recebe diploma.

Há na literatura científica, diversos trabalhos sobre os problemas que os alunos enfrentam nas disciplinas de programação e sobre os métodos e ferramentas que podem trazer maior benefício em seu ensino. Ainda não há uma conclusão definitiva para esses desafios, por mais que identifiquemos as dificuldades do aprendiz de programação, não existe ainda um senso comum, uma metodologia a ser seguida.

Outra discussão importante para o processo de aprendizagem é identificar em qual estágio de compreensão de determinado assunto o aprendiz encontra-se, assim como avaliar se os conhecimentos teóricos e práticos foram adquiridos. Esse é um ponto que merece atenção dos pesquisadores que se interessam pelo ensino da programação já que os resultados desses estudos estabelecem o terreno o qual, posteriormente, metodologias e ferramentas se apoiarão para uma discussão intermetodológica guiada pelos critérios comuns estabelecidos pelos estágios propostos de teorias para esse objetivo.

Nesse contexto, este trabalho estuda sobre o processo de aprendizagem da linguagem de programação no ensino técnico e tem como objetivo geral propor um modelo teórico que permita que o professor realize o planejamento da sequência do ensino de programação baseado em seus conceitos fundamentais. Nos objetivos

específicos estão o agrupamento e classificação dos fundamentos da programação baseado em um critério de complexidade que permita servir de índice no momento do planejamento.

As primeiras decisões tomadas foram escolher o nível de ensino e o paradigma de programação. Para o sistema de ensino foi escolhido o ensino técnico por ter como foco a formação profissional do aprendiz e pela classe discente geralmente não possuir nenhum conhecimento prévio na área, portanto carece de ser explorado. Para o paradigma de programação, o paradigma imperativo foi escolhido por possuir maior adesão entre a academia e, consequentemente, pelo mercado de trabalho. Identifica-se, inicialmente, a impossibilidade de ser criada uma solução para todos os paradigmas de programação devido à diferença de raciocínio que cada uma exige.

Por fim, apoiando-se nas teorias de aprendizagem significativa e no currículo em espiral, será proposto um modelo para o processo de aquisição da linguagem de programação imperativa em que os estágios que o aprendiz percorre, enquanto adquire os fundamentos que o capacitam como usuário de uma linguagem de programação, possam ser bem definidos. Esses estágios estão caracterizados nos graus de composição e abstração dos conceitos intrínsecos da programação.

O modelo teórico de organização e planejamento do ensino de programação será denominado Estrutura TFR (Tipo, Função e Rota), o qual indica as três principais habilidades necessárias no ato de programar, sendo elas, definir os tipos de dados para o problema em questão; manipular esses mesmos dados através de funções que permitirão encontrar a resposta do problema; e por fim, sequenciar essas funções, prevendo desvios ou repetições de forma que atenda aos requisitos da tarefa a ser solucionada. A estrutura TFR agrupa os conceitos definindo-os por seus graus de complexidade, partindo de conceitos mais simples para os mais complexos, em que a ideia de elementos primitivos, compostos e abstratos são evidenciados ao aprendiz durante todo seu processo de aprendizagem.

A partir da estrutura TFR, o modelo em espiral será utilizado, a cada volta o aprendiz é exposto aos elementos planejados previamente pelo professor dentro da estrutura TFR e são apresentados de forma mais ou menos aprofundadas, isso porque a cada nova volta da espiral, elementos novos são apresentados e combinados a elementos anteriores, possibilitando o gradual aprofundamento e assimilação de conceitos através da teoria e prática.

As sessões que dividem essa pesquisa são: (i) Introdução; (ii) Ensino de Programação, onde identifica-se os objetivos do ensino técnico e superior e as habilidades esperadas a partir da lei de diretrizes e bases da educação e, ainda nessa sessão, é trazida a ideia de aluno novato e os trabalhos que pesquisam sobre as dificuldades e propostas para o ensino de programação. Por último, são listadas as perguntas da pesquisa; (iii) Metodologia, por onde o processo de pesquisa e as hipóteses são apresentadas; (iv) Estrutura TFR, apresentando a resposta às perguntas da pesquisa e a contribuição principal desse trabalho; (v) Espiral TFR, a qual demonstra a aplicação da TFR no processo de ensino de programação; (vi) Guia TFR, onde será demonstrado, em formato de manual de instruções, como a espiral TFR pode ser aplicada pelo professor no planejamento da disciplina de programação; e por último, (vii) Conclusão.

2 ENSINO DE PROGRAMAÇÃO

2.1. Níveis de Ensino

A área de desenvolvimento de software é uma área que cresce a cada ano trazendo novas possibilidades tecnológicas que facilitam e automatizam tarefas da rotina em comum das pessoas. Novos desafios surgem e os profissionais devem estar capacitados para enfrentá-los. No campo da educação, os profissionais que lecionam as disciplinas de desenvolvimento de software se deparam com a dificuldade em transmitir os conhecimentos fundamentais que regem a chamada ciência da computação. O ensino de programação é realizado nas seguintes categorias: (i) Ensino Técnico de Nível Médio, (ii) Ensino Superior Tecnólogo ou Bacharel, (iii) Ensino Básico e (iv) Ensino Livre. Apenas (i) e (ii) são mantidos pelo MEC (Ministério da Educação) e passam por critérios mais rigorosos de aprovação já que se destinam a formação profissional, enquanto (iii) e (iv) podem possuir objetivos diversos como auxiliar no desenvolvimento lógico do aluno como complemento a outras áreas de conhecimento ou apresentar tecnologias específicas e atuais.

Essa pesquisa destina-se ao ensino de programação voltado à capacitação profissional de estudantes do ensino técnico, por essa razão é importante entender seus objetivos e as características principais desse público.

2.2. Contextualização do Nível Técnico

Os cursos de Educação Profissional Técnica de Nível Médio têm como finalidade “proporcionar ao estudante conhecimentos, saberes e competências profissionais necessários ao exercício profissional e da cidadania, com base nos fundamentos científico-tecnológicos, sócio-históricos e culturais” (MEC, 2012). A terceira edição do Catálogo Nacional de Cursos Técnicos (CNCT) disponibilizado pelo MEC (2014), apresenta 227 cursos agrupados em 13 eixos tecnológicos, quatro deles pertencem a área de programação de software e estão localizados no eixo *Informação e Comunicação*.

Tabela 1 – CNCT Cursos de Programação

Curso	Carga Horária	Ocupação CBO
Técnico em Desenvolvimento de Sistemas	1000 h	317105-Programador de internet. 317110-Programador de sistemas de informação
Técnico em Informática	1200 h	317110-Programador de sistemas de informação. 317210-Técnico de apoio ao usuário de informática. 317205-Operador de computador. 313220-Técnico em manutenção de equipamentos de informática.
Técnico em Informática para Internet	1000 h	317105-Programador de internet. 317120-Programador de multimídia. 317110-Programador de sistemas de informação
Técnico em Programação de Jogos Digitais	1000 h	317120-Programador de multimídia.

Fonte: Adaptado de MEC (2014).

O documento base publicado pela Secretaria de Educação Profissional e Tecnológica (MEC, 2007), informa que "os primeiros indícios do que hoje se pode caracterizar como as origens da educação profissional surgem a partir de 1809, com a criação do Colégio das Fábricas, pelo Príncipe Regente, futuro D. João VI" e continua dizendo que o Brasil tem, portanto, a sua origem "dentro de uma perspectiva assistencialista" mas que se renovaria no início do século XX, "modificando para a preparação de operários para o exercício profissional", avançando até chegar em sua concretização oficial na Lei de Diretrizes e Bases da Educação Nacional.

No artigo 36-A da lei 9394 de 20 de dezembro de 1996, referente à Educação Profissional Técnica de Nível Médio, encontra-se o descritivo:

> Art. 36-A. Sem prejuízo do disposto na Seção IV deste Capítulo, o ensino médio, atendida a formação geral do educando, poderá prepará-lo para o exercício de profissões técnicas. (BRASIL, 1996)

Na resolução Nº 6, de 20 de setembro de 2012, redigido pelo Conselho Nacional da Educação sobre essas diretrizes, destacam-se as seguintes considerações empregadas no capítulo II, Princípios Norteadores:

> I - Relação e articulação entre a formação desenvolvida no Ensino Médio e a preparação para o exercício das profissões técnicas, visando à formação integral do estudante. (MEC, 2012)

Portanto, podemos entender que o ensino técnico possui o objetivo final de capacitar profissionalmente o estudante com as habilidades necessárias para exercer funções operárias, ou seja, o caráter filosófico e científico do ensino superior não é sua maior característica.

Ainda nos Princípios Norteadores, item V é colocado a indissociabilidade entre educação com a prática social, e teoria com a prática, o que evidencia a preocupação em sua gênese de capacitar o aprendiz para fins práticos, úteis e atuais da sociedade através da entrada no mercado de trabalho. Isso, todavia, não separa o aprendiz dos conhecimentos teóricos, pelo contrário, é indissociável, isto é, devem estar sempre de acordo com os conhecimentos científicos da época. Em sua definição de tecnologia através do olhar sociólogo, MOURA, D., et all afirmam que:

> [...] tecnologia é uma extensão das capacidades humanas. Podemos definir a tecnologia, então, como mediação entre ciência (apreensão e desvelamento do real) e produção (intervenção no real) (MOURA, D., et all).

Com a ajuda dos artigos presentes na lei que estabelece as finalidades do ensino básico, técnico e superior e as resoluções que suportam essa lei, pode-se evidenciar critérios semelhantes com níveis de prioridade diferentes entre o ensino técnico e superior: No que diz respeito ao impacto social e cultural, o ensino superior prioriza a melhoria constante da ciência através da produção acadêmica e da difusão dos saberes científicos à educação básica, já no ensino técnico, está ligado diretamente à ação do trabalho profissional realizado, que por sua vez, buscará os saberes contemporâneos da ciência, de forma que possam ser aplicados à prática do trabalho (MOURA, D, et all).

2.3. O Novato de Dreyfus

O termo novato será atribuído aqui para fazer referência aos estudantes que cursam o ensino técnico. No modelo de aquisição de competências definido por Dreyfus, o estágio novato é definido por aquele que “Tem um entendimento incompleto, aborda as tarefas mecanicamente e precisa de supervisão para concluí-las.” (LESTER, 2005, tradução do autor). O estágio novato é o primeiro da escala e por sua vez, se depara com estudantes que possuem pouca ou nenhuma experiência sobre o objeto de estudo.

O ensino técnico de nível médio tem como público-alvo alunos que ainda não terminaram o ensino médio, ou seja, sua faixa etária está entre 15 e 18 anos, sendo optativo cursar os cursos oferecidos de forma integrada ou concomitante à educação básica (MEC, 2012).

O incentivo a ensinar lógica de programação no ensino fundamental ainda está prematuro, e quando ocorre, é por meio de atividades e ferramentas lúdicas, como o ensino de lógica por blocos utilizando Scratch (ROCHA, 2017). A ausência da rigidez de uma linguagem de programação possibilita que os conceitos lógicos possam ser expressados sem seus detalhes de implementação, por isso, o uso de linguagens de programação é evitado nesse nível de ensino.

Assim, compreende-se que aquele que cursa o ensino técnico, possuirá pouco ou nenhum conhecimento prévio, quanto aos fundamentos da programação expressos através de uma linguagem de programação.

Atribuir a ele o termo novato parece adequado já que traz a ideia de iniciante em que “o iniciante recebe regras para determinar as ações com base nesses recursos, assim como um computador seguindo um programa.” (DREYFUS, 2004, tradução do autor). Percebe-se que o novato não possui outra escolha a não ser seguir regras pré-determinadas pelo professor para cumprir tarefas e resolver problemas.

No que diz respeito ao ensino e aprendizagem de programação, Salleh, Shukur e Judi (2013, tradução do autor) afirmam que "os alunos precisam de instruções claras e precisas e precisam do apoio dos colegas", o que demonstra a necessidade de guiar passo a passo os primeiros caminhos do novato em programação, isso ocorre, como afirmam os autores, pela caracterização de consumidor em confronto à de produtor de tecnologias do novato, ou seja, as habilidades necessárias para produzir novos artefatos computacionais não estão presentes na bagagem do novato, que por sua vez, precisa ser guiado passo a passo para compreender a nova forma de pensar.

Ainda como forma de suprir a necessidade de regras precisas, a técnica de *scaffolding* (KUNKLE, 2010, tradução do autor), a qual "é uma forma de suporte que permite a um aluno resolver um problema, realizar uma tarefa ou atingir um objetivo que estaria além de suas capacidades sem assistência" é utilizada nas disciplinas introdutórias de programação com intuito de amenizar a dureza dos conceitos necessários a seu aprendizado. Nagano e Direne (2016) refere-se à técnica de *scaffolding* aplicada a programação onde:

> O aluno novato é induzido a resolver certo conjunto de problemas de uma determinada maneira e assim adquire a perícia ou a capacidade de agrupar comandos para formar soluções através da repetição do uso de um conjunto de comandos que foram apresentados através de um exemplo pronto. (NAGANO; DIRENE, 2016)

À medida que as regras são compreendidas e novas situações vão sendo resolvidas, o novato começa a avançar para alcançar o estágio "iniciante avançado" até conquistar o nível "expert" (DREYFUS, 2004).

Para tanto, o foco deve estar na construção de um modelo para que o novato em programação perceba os degraus necessários a conquistar em sua trajetória no aprendizado de programação, e que possibilite o professor estruturar o corpo conceitual das competências necessárias no que diz respeito à sequência de ensino.

2.4. Dificuldades de aprendizagem

As disciplinas introdutórias de programação possuem alto índice de reprovação e causam desmotivação aos estudantes que pretendem seguir a área de tecnologia (FERNANDES; FREITAS JUNIOR, 2014). O crescimento da área pede que novos profissionais estejam habilitados a preencher posições que exigem competências difíceis de serem compreendidas, por isso, continuam os esforços em pesquisar meios que facilitem a compreensão desses conhecimentos. Para apontar caminhos e soluções é necessário identificar suas atuais dificuldades de aprendizagem.

2.4.1. Conhecimento prévio

Annamalai e Salam (2017) apontam que, geralmente, novatos possuem diferentes níveis de conhecimentos quando entram em um curso de programação introdutório. Muitos desses conhecimentos, como raciocínio lógico, concentração, abstração e organização de ideias são requeridos no momento de programar (SANTOS et al, 2015). Assim, a primeira dificuldade que se apresenta é a de nivelamento entre os conhecimentos atuais da turma.

Como causa desse desnivelamento, estão as deficiências de formação observadas no ensino fundamental e médio (SANTOS et al, 2015) e (LIMA; LEAL, 2013). Além disso, os alunos pouco conhecem as exigências inerentes à área e das habilidades necessárias para desenvolver programas computacionais. Quanto às habilidades prévias capazes de serem trabalhadas no ensino básico, Lima e Leal (2013) sugerem que o "desenvolvimento gradual dessas habilidades poderia ajudar a diminuir as dificuldades dos alunos frente às unidades curriculares de programação de computadores e suas exigências". Annamalai e Salam (2017, tradução do autor)

afirmam que aprender a programar "é intensamente meticuloso para alunos iniciantes que não têm qualquer experiência em programação de computadores".

Ao cursar as disciplinas introdutórias de programação, o novato possui experiências e habilidades que o caracterizam como consumidor de tecnologia (SALLEH; SHUKUR; JUDI, 2013). Eles afirmam que as diversas estratégias em capacitar o novato estão em "aumentar o envolvimento dos alunos e desenvolver o pensamento criativo como uma das estratégias de preparação para os alunos se tornarem futuros produtores, não apenas consumidores de tecnologia".

A essa falta de habilidades criadoras no mundo computacional, Kunkle (2010) diz que:

> [...] muitas das dificuldades experimentadas por um estudante iniciante de ciência da computação derivam do fato de que ele não possui um modelo eficaz de computador. Em outras palavras, ele não tem nenhum conhecimento existente para integrar as informações sobre computadores e programação que ouve em aula ou lê em seu livro didático. Consequentemente, ele deve construir seu próprio modelo de computador. (KUNKLE, 2010, tradução do autor).

2.4.2. Complexidade dos conceitos

Annamalai e Salam (2017) refere-se ao aprendizado dos conceitos gerais da programação como o maior desafio que o novato enfrenta em seus primeiros passos. A esses mesmos conceitos, os autores afirmam que por serem muito abstratos e difíceis de aplicá-los no momento da construção do programa, acabam por causar frustação e desmotivação. Um estudo realizado por eles indicou que "os alunos estavam lutando para usar os conceitos de programação, estrutura e execução do programa que são muito abstratos e difíceis de visualizar". Annamalai e Salam (2017) aponta que ainda os que terminam as classes introdutórias "não sabem o que estão fazendo, pois não têm aprimoramento contínuo nos conceitos de programação". A necessidade da compreensão dos fundamentos gerais da programação está clara e é, por muitas vezes, o foco de pesquisas voltadas a seu ensino.

Outro fator decorrente da falta de um modelo de computador é a analogia errônea normalmente realizada por alunos novatos entre a linguagem de programação e outras linguagens. Kunkle (2010) descobriu, em seu estudo sobre o ensino da linguagem Pascal, que novatos tendem a confundir o significado de termos de programação com seu significado na linguagem natural. Kunkle em seu estudo, mostra um particular aluno dizer que a expressão *let c = c + 1* não fazia sentido já que

na matemática seria impossível que tal expressão fosse verdadeira. Por estar situada em um ambiente mecânico onde a máquina é quem executa os comandos descritos pela linguagem de programação, ideias de *intenção, interpretação, ambiguidade e imprecisão* não fazem parte da comunicação homem-máquina. Decorrente dessas confusões linguísticas, estão outros motivos pela falta de motivação dos novatos em programação (SALLEH; SHUKUR; JUDI, 2013).

Mohorovičić e Strčić (2011, tradução do autor) definem que os três principais objetivos pedagógicos no ensino de programação devem abranger "a sintaxe da linguagem, desenvolver habilidades de design de programa e pensamento criativo". Quanto ao processo de programar, Salleh, Shukur e Judi (2013, tradução do autor) definem como sendo a combinação de atividades como "planejar, modelar, testar e debugar". Fontes e Silva (2008) criticam os livros atuais disponíveis que atendem a essas habilidades dizendo que não são didáticos, pelo contrário são "guias completos sobre softwares específicos da linguagem de programação".

2.4.3. Interação Professor Aluno

Alterando o foco para o professor, Lima e Leal (2013) dizem que ele deve possuir domínio do conteúdo e das competências pedagógicas para "propiciar a satisfatória didatização dos conteúdos". Em meio à complexidade já referida, é importante que o professor não seja passivo quanto ao ensino, ademais, ele é responsável pela motivação (MOHOROVIČIĆ, STRČIĆ, 2011), garantindo o engajamento dos estudantes nas tarefas que atribui. Ainda a ele, é atribuída a competência de criar estratégias didáticas que permitam que o novato possua "controle total e capacidade de navegar pela lição" (ANNAMALAI; SALAM, 2017, tradução do autor), ou seja, que tenha autonomia em navegar pelas lições de forma que se sinta no controle de seu aprendizado.

Devido ao desnivelamento de estudantes, alguns alunos estão mais preparados que outros, assim, esses preferem seguir seu próprio ritmo e chamar o professor quando estiverem travados em um problema (MATTHÍASDÓTTIR, 2006). Assim o professor pode preparar materiais on-line seja por texto ou vídeo, e quando estiverem em sala de aula, oferecer maior atenção aos alunos com menor predisposição.

Araújo, Bittencourt e Santos (2018, tradução do autor) afirmam "Estudantes não estudam em casa [...] Este é um problema desafiador". O ensino de programação é complexo e leva tempo, ele exige muito do esforço do novato a desenvolver o pensamento algorítmico. Em uma entrevista realizada por Lima e Leal (2013) a estudantes das disciplinas iniciais de programação, alguns alunos comentam sobre os esforços em aprender programação: "Não é uma matéria que você vai ler e vai entender logo, logo. Não é igual a você ler um texto", outro aluno também comenta "Você tem que correr atrás, não pode ficar esperando só de quem está ensinando". A aprendizagem de programação exige que o estudante queira aprender, esteja motivado e ativo em seu ensino, encará-la como uma simples leitura passiva trará consequências negativas, portanto, é necessário que o estudo esteja presente também fora da sala de aula, porém dada a situação dos jovens que cursam o ensino técnico no Brasil, poucos possuem equipamento adequado para manter seus estudos em dia (ARAUJO; BITTENCOURT; SANTOS, 2018).

Quanto às características emocionais dos estudantes, Lima e Leal (2013) descobrem que essas podem ser divididas em dois polos:

> Aqueles que se aprofundam no conhecimento em busca de um embasamento que lhes possa valer no futuro – motivação intrínseca – e os que apenas se dedicam a cumprirem o “regulamento”, ou seja, fazer provas e trabalhos visando à conclusão da unidade curricular com um resultado minimamente satisfatório – motivação extrínseca. (LIMA; LEAL, 2013)

2.4.4. Metodologias

Programar consiste em três componentes principais: (i) programa, (ii) ferramenta de programação e (iii) linguagem de programação (SALLEH; SHUKUR; JUDI, 2013). Sendo que (i) é o resultado de (ii) e (iii), o ensino de programação foca nos dois últimos componentes. Os autores afirmam que as habilidades em utilizar uma ferramenta de programação são consideradas tão importantes e equivalentes às habilidades de sintaxe e lógica de uma linguagem de programação. Dada a diversidade de ferramentas, a familiarização com uma ferramenta de desenvolvimento acaba por ampliar ou diminuir, explicitar ou esconder, os conceitos gerais da programação.

A escolha da linguagem de programação e abordagem mais adequadas são colocadas como as duas fundamentais fonte de problemas no ensino de programação (MOHOROVIČIĆ; STRČIĆ, 2011). De outro lado, Kaplan (2010, tradução do autor)

diz que ensinamos programação apontando para ferramentas que resolvem problemas específicos, e que "não abordamos as competências gerais ou meta-habilidades exigidas no processo de programação". Essas *meta-skills* podem ser vistas pela afirmação de Mohorovičić e Strčić (2011):

> A programação de computadores requer o uso de habilidades cognitivas complexas, como raciocínio, solução de problemas e planejamento. Um programador forma representações abstratas de um processo, as expressa na forma de estruturas lógicas e, finalmente, as traduz em código correto usando a linguagem formal. (MOHOROVIČIĆ; STRČIĆ, 2011, tradução do autor)

Ramos, Neto e Vega (2009) definem linguagem formal como sendo "um conjunto finito ou infinito, de cadeias de comprimento finito, formadas pela concatenação de elementos de um alfabeto finito e não-vazio".

Lima e Leal (2013) comentam que as linguagens que representam o problema, em princípio, podem ser qualquer linguagem, já que o objeto não é ensinar programação, mas sim, como representar soluções de um problema em uma linguagem computacional:

> Assim, como meio de representação, o processo de aquisição da linguagem de computação deve ser a mais transparente e a menos problemática possível. Ela é um veículo para expressão de uma ideia e não o objeto de estudo. (LIMA; LEAL, 2013)

A dificuldade em conciliar teoria e prática é alvo das discussões sobre os métodos utilizados no ensino de programação (SANTOS et al, 2015). É necessário observar que prática sem teoria é como estar em um avião e em pleno voo, descobrir para que servem seus comandos e como se guia. Ao passo que teoria sem prática, é como conhecer todos os comandos e técnicas de voo sem nunca ter encostado em um avião. Matthíasdóttir (2006, tradução do autor) em sua entrevista com novatos, conclui que "a melhor forma de aprender programação era trabalhando sozinho nos cursos de programação, nas aulas práticas (na sala de informática) e sozinhos com o material didático".

Assim, a escolha da ferramenta, linguagem e metodologia de ensino são aspectos essenciais no ensino de programação. Ademais, as escolhas não podem transferir a responsabilidade do processo de aprendizagem do professor ao estudante (SANTOS et al, 2015). Dizem Lima e Leal (2013):

> Constitui-se como uma ideia errônea, o pensamento de que, nessa dinâmica, o discente aprenderia sozinho; que basta apresentar-lhe um problema, colocá-lo na frente de um computador para que o processo [...] se consolide. Muito pelo contrário. O aluno pode, por exemplo, incorrer numa situação em que ele não sabe um conceito – o que representaria uma estagnação do

> processo. Daí a importância do suporte a ser suprido pelo professor. (LIMA; LEAL, 2013)

Ainda outro fator criticado pelos autores é o modelo de funcionamento tradicional das unidades curriculares de programação, ou seja, a forma que são apresentados os conceitos gerais de programação e sua prática, geralmente através de exposição teórica seguida de lista de exercícios. Devido à complexidade dos conceitos envolvidos, o novato acaba por se desmotivar diante da forma pesada a qual o conhecimento é apresentado pelo professor. Eles confirmam essa afirmação dizendo:

> O tecnicismo predominante na Ciência da Computação, área de conhecimento na qual se enquadra a programação de computadores, parece exercer uma influência muito forte no comportamento rígido dos professores da área. Entendeu-se, que seria necessário o desenvolvimento de uma cultura mais leve (softer), [...] para se tentar promover o processo educacional. Ou seja: a rigidez imposta pelo formalismo matemático da computação precisaria ser adequada a uma forma menos abstrata, mais inteligível e atraente para o aluno. (LIMA; LEAL, 2013)

Para vencer as dificuldades desse modelo tradicional, Lima e Leal (2013) relatam a importância do relacionamento professor-aluno, que mantém a motivação do ambiente de aprendizagem e "que ajuda na superação de desafios e no acompanhamento de dificuldades particulares" de cada estudante.

Um último aspecto quanto à apresentação dos conceitos refere-se a sua sequência, Berssanette e Francisco (2018) afirmam que "a sequência em que tais informações são apresentadas nem sempre é a mais adaptada para facilitar a interação com o conhecimento do estudante". Aqui aparece um ponto culminante no ensino de programação, o qual é explorado posteriormente nessa pesquisa, em que cada professor, estabelece sua própria sequência de ensino, e impossível é comparar através de índice seguro as variações dessa sequência. Para estabelecer uma sequência de ensino é "essencial um planejamento que considere a estrutura da disciplina e dos conteúdos, a organização desse material ao longo do tempo e, em particular, manter foco no estudante e estimar seu conhecimento prévio" (BERSSANETTE, FRENCISCO, 2018).

Desde sua concepção, o tema sequência é discutido entre os pesquisadores, na tentativa de dividir os elementos de uma linguagem de programação em subconjuntos que podem ser trabalhados de forma independente e concentrados ao estudante os conceitos pertencentes a esse subconjunto, cada avanço é considerado um estágio percorrido e "em cada estágio, as novas construções e elementos de um

subconjunto atual são apresentados ao aluno e, em seguida, o novo conhecimento é dominado pela resolução de uma série de problemas de programação" (BRUSILOVSKY, 1994, tradução do autor). Essa abordagem foi chamada de abordagem em espiral e tem sido utilizada desde então, no entanto, carecem as pesquisas que aprofundam a discussão da sequência no ensino de programação respaldadas em teorias de aprendizagem como a aprendizagem significativa e o currículo em espiral.

2.5. Propostas de ensino

A programação de computadores requer o uso de habilidades cognitivas complexas, como raciocínio lógico, planejamento e solução de problemas (MOHOROVIČIĆ, STRČIĆ, 2011). Ventura (2019) pergunta "como tornar um conteúdo considerado difícil, como é o caso da Lógica de Programação, em interessante ou com sentido para os alunos?". As pesquisas que propõem a melhoria do ensino-aprendizagem de programação atuam para responder essa pergunta e garantir a qualidade na compreensão de seus conceitos intrínsecos.

Para tal tarefa, o professor deve planejar a sequência dos conteúdos considerando os conhecimentos atuais do aluno, de forma que não afete negativamente sua motivação, mas que mantenha sua vontade de aprender sempre ativa. Mohorovičić e Strčić (2011) comentam essas tarefas dizendo:

> A tarefa do professor é selecionar um método de ensino apropriado, ou uma combinação de métodos, para planejar tarefas para os alunos em conformidade e, depois de apresentar os conceitos da maneira que ele escolheu, mudar o foco nos alunos, motivando-os a se envolverem nas tarefas ele planejou. (MOHOROVIČIĆ; STRČIĆ, 2011, tradução do autor)

Dessa forma, o professor deve garantir primordialmente a compreensão dos conceitos intrínsecos ao objeto de estudo e a motivação dos estudantes proporcionando-lhes as ferramentas necessárias para construírem seu conhecimento de forma que se sintam no controle de seu aprendizado. Quanto ao ensino de programação, as pesquisas que propõem melhorias em seu ensino, apontam para ferramentas, técnicas, métodos e abordagens que contribuem nesse sentido.

2.5.1. Ferramentas e Linguagem

Programar consiste em três componentes principais: "o programa, ferramentas de programação e linguagem de programação" (SALLEH, SHUKUR, JUDI, 2013,

tradução do autor). Uma das primeiras escolhas que o professor das disciplinas introdutórias de programação realiza enquanto define suas estratégias pedagógicas é estabelecer qual linguagem e ferramenta de programação utilizará.

Araújo, Bittencourt e Santos (2018) apontam que "pesquisadores desenvolveram diversas ferramentas e linguagens para permitir a adoção desses contextos no ensino de programação.". Essas ferramentas e linguagens, geralmente, estão voltadas ao ensino de novatos, exemplos dessas ferramentas são: "Sratch, Logo, KTurtle, App Inventor, Lego Mindstorms, Jython Environment for Students (JES) e PPlay" (ARAUJO, BITTENCOURT, SANTOS, 2018). Devido à quantidade de ferramentas e linguagens disponíveis, o professor deve combinar o uso desses recursos e apresentar um ambiente propício à aprendizagem, complementam os autores.

A decisão de qual linguagem de programação é adequada a estudantes novatos é discutida por Araújo, Bittencourt e Santos (2018), no entanto, Santos (et al 2015) após apresentar sua proposta de aprendizagem através de um ambiente visual, afirma que "o objetivo principal da disciplina não é focar em uma ou outra linguagem, mas nos conceitos elementares da lógica de programação".

A escolha da ferramenta e linguagem de programação ditam as ideias que serão transmitidas ao estudante novato, já que cada linguagem ou ferramenta pode explicitar ou esconder preocupações que o novato enfrentará no processo de desenvolver um programa de computador. Ao passo que adentrando o mercado de trabalho, o profissional pode se deparar com tecnologias diferentes do que estudou, é essencial que os conceitos gerais contidos em qualquer linguagem de programação sejam explicitados pelo professor ao utilizar essa ou aquela tecnologia.

2.5.2. Apresentação do conteúdo

O processo de ensino-aprendizagem ocorre quando o professor apresenta o conteúdo ao estudante. Mohorovičić e Strčić (2011) apontam dois focos que diferenciam a apresentação do conteúdo:

> Enquanto alguns se concentram em como os conceitos de programação são apresentados aos alunos e como os materiais do curso são entregues [...], outros tentam encontrar maneiras alternativas em que os alunos podem trabalhar com esses conceitos. (MOHOROVIČIĆ; STRČIĆ, 2011)

Assim, percebe-se que a preparação do material e o meio (presencial, on-line, etc) o qual ele é entregue são fatores importantes no processo de aprendizagem. Annamalai e Salam (2017) apontam para o uso de recursos multimídia como tutoriais que servirão como ferramenta de motivação a alunos novatos. Os alunos podem rever o material gravado e desenvolver suas próprias estratégias de estudo, caminhando conforme sua necessidade, anotando as dúvidas e solicitando ajuda do professor quando um desafio não puder ser resolvido por ele próprio. É o que comentam Lima e Leal (2013) ao dizerem que:

> Os alunos colocaram o professor num lugar especial no processo de ensino-aprendizagem de programação. Além de ser o responsável por estruturar e criar as condições para a apresentação dos conteúdos programáticos, os alunos esperavam que a forma de conduzir o processo de aprendizagem extrapolasse a instrução direta, sendo necessária a incorporação de uma dimensão didática no seu fazer, objetivando a motivação e o envolvimento discente. (LIMA; LEAL, 2013)

Além da preparação e entrega do conteúdo, o professor deve garantir que este seja compreendido, Matthíasdóttir (2006) sugere uma apresentação híbrida de conteúdo presencial e on-line. Em seu trabalho, ele prepara pequenos testes e quizzes on-line semanais para que os estudantes acompanhem seu progresso durante o curso. Ele confirma que essa nova combinação de métodos e ferramentas focadas em "sessões de laboratório, palestras gravadas, testes online e ensino individual têm tido muito sucesso".

Voltando à diferença de foco apresentada por Mohorovičić e Strčić (2011), quanto à maneira que os estudantes podem trabalhar os conceitos apresentados, Araújo, Bittencourt e Santos (2018, tradução do autor) sugerem que atividades contextualizadas "são importantes para despertar atenção e relevância", isto é, garantir o engajamento e motivação dos estudantes continua presente nas preocupações do professor no processo de ensino-aprendizagem. O contrário seria trabalhar em atividades fora do contexto dos estudantes, ou seja, que não possuem significado às suas experiências e conhecimentos anteriores.

Fontes e Silva (2008) criticam essa descontextualização em tarefas iniciais a estudantes novatos, através da solução de "programas que resolvem problemas matemáticos a partir das informações digitadas pelo usuário, como o cálculo de somas e de médias aritméticas, cálculos de fatoriais e da sequência de Fibonacci". Aqui deve-se tomar cuidado em analisar esse argumento, já que as atividades iniciais aprendidas em uma linguagem de programação, inicialmente envolvem operações aritméticas,

deve-se, pois manter um equilíbrio entre cálculos complexos que exijam conhecimentos de domínio do estudante e tarefas que demonstrem os conceitos gerais da linguagem de programação. Araújo, Bittencourt e Santos (2018) apresentam uma proposta que busca "ensinar conceitos de programação, buscando facilitar o aprendizado e aumentar a motivação dos alunos em relação à programação", ou seja, para o novato, o foco não está no conhecimento de domínio que ele possui sobre o problema que está resolvendo, mas em apresentar os conceitos elementares da programação.

2.5.3. Abordagens

A abordagem diz mais respeito à estratégia pedagógica do que a especificidade do objeto de estudo. O construtivismo tem tido uma das teorias gerais do conhecimento utilizadas nas disciplinas introdutórias de programação. Kunkle (2010, tradução do autor) acredita que o construtivismo é uma "uma abordagem particularmente apropriada para o ensino de programação". Ele define o construtivismo sendo uma:

> Teoria da aprendizagem que afirma que os alunos constroem ativamente o conhecimento, em vez de absorvê-lo passivamente de livros e palestras. O novo conhecimento é criado combinando o conhecimento experiencial com o conhecimento existente ou refletindo sobre o conhecimento existente. (KUNKLE, 2010, tradução do autor)

A própria ação de programar favorece atividades construtivistas, segundo o autor. Partindo para as teorias de aprendizagem, ele diz que a aprendizagem significativa "ocorre da mesma forma que os construtivistas afirmam que novos conhecimentos são criados". Kunkle (2010) descreve o processo da aprendizagem significativa como:

> Especificamente, a aprendizagem significativa é um processo por meio do qual um aluno conecta novas informações com o conhecimento já existente na memória. O processo de conexão é chamado de "assimilação". O conhecimento existente ao qual novas informações são conectadas é chamado de "esquema". Aprender sobre computadores e programação de computadores requer assimilação ao esquema de novas informações técnicas. (KUNKLE, 2010, tradução do autor)

O termo assimilação vem da epistemologia genética de Piaget, e trabalha juntamente com a acomodação de novos conhecimentos nas estruturas cognitivas do estudante. Piaget define esse processo como:

> [...] uma integração a estruturas prévias, que podem permanecer invariáveis ou são mais ou menos modificadas por esta própria integração, mas sem

> descontinuidade com o estado precedente, isto é, sem serem destruídas, mas simplesmente acomodando-se à nova situação.” (PIAGET, 1973, p. 13)

A teoria da aprendizagem significativa tem sido alvo dos estudos sobre o ensino de programação a estudantes novatos (FONTES, SILVA, 2008), já que estes não possuem conhecimento prévio e necessitam que as construções mentais realizadas sejam significativas no sentido de ancorar as ideias relevantes dos conceitos novos estudados às estruturas de conhecimento atuais do novato (KUNKLE, 2010).

Algumas universidades adotaram o curso denominado CS0 (Computer Science 0) como um pré curso que antecede o estudante às primeiras disciplinas que ele irá cursar (CS1) (KAPLAN, 2010). Dessa forma, é possível que seja realizado um alinhamento dos conhecimentos base necessários ao decorrer do curso, assim como preparar o aluno para a ancoragem dos conhecimentos específicos de programação. Essa estratégia tem se tornado comum e eficaz entre as universidades, prevenindo a desmotivação e diminuindo o índice de evasão das disciplinas introdutórias (KAPLAN, 2010).

Uma última abordagem sendo utilizada no ensino de programação é a abordagem em espiral. Araújo, Bittencourt, Santos (2018) apontam o ensino em espiral como:

> Um elemento que afeta positivamente a aprendizagem dos alunos para não frustrá-los, introduzindo o conteúdo gradualmente e em níveis crescentes de complexidade. Isso torna possível aprender conceitos e cumprir os desafios e atividades propostas. (ARAUJO; BITTENCOURT; SANTOS, 2018)

Valente (2005), após descobrir o ciclo descrição-execução-reflexão-depuração que descreve a interação aprendiz-computador como processo de aprendizagem, também atribui o ciclo à dinâmica em espiral afirmando que com os avanços computacionais e uma melhor compreensão sobre a aprendizagem, tem-se observado que "a imagem da espiral capta melhor o processo de construção de conhecimento que resulta da interação aprendiz-computador". Como resposta de sua pesquisa, Araújo (2018) associa a ideia de blocos contextualizados a subconjuntos de uma linguagem de programação à dinâmica em espiral afirmando ser possível nessa abordagem "entender como esses conteúdos são repetidos em formato de espiral, dentro de blocos contextualizados, agregando maior profundidade aos conteúdos".

Mesmo com as últimas tentativas de melhorar o ensino de programação, fica evidente que não se chegou a uma conclusão comum de quais ferramentas, linguagens e abordagens devem ser adotas, bem como um método específico que

descreve a aprendizagem a novatos em programação. As pesquisas continuam contribuindo com o avanço da compreensão do processo de aquisição de uma linguagem de programação. Enquanto isso, "não temos ferramentas padronizadas para medir a aprendizagem dos alunos de conceitos fundamentais de programação" (KUNKLE, 2010, tradução do autor).

2.6. Perguntas da pesquisa

A fim de nortear os objetivos que circundam essa pesquisa, estando contextualizados nos itens anteriores quanto às pesquisas no ensino de programação e sua ação no ensino técnico, as perguntas abaixo que centralizam as preocupações principais a serem respondidas foram designadas.

- **Q1.** Qual estrutura conceitual descreve os principais elementos da programação imperativa de forma que facilite a compreensão a alunos novatos explicitando os fundamentos da programação?

- **Q2.** Através de qual modelo teórico pode ser entendido o processo de aprendizagem de uma linguagem de programação quanto a sua sequência e conteúdo tendo em vista a caracterização de novato do estudante?

3 METODOLOGIA

3.1. Tipo da pesquisa

Os procedimentos da pesquisa científica fazem parte de um processo metodológico sistemático. Esse processo possui critérios que guiam os caminhos que serão percorridos pelo pesquisador e possui cinco divisões fundamentais: finalidade, objetivos, abordagem, método e procedimentos. A seguir, serão descritas as categorias dessas divisões em que essa pesquisa se apoiará.

Será utilizada a finalidade básica estratégica, a qual consiste em aprofundar o conhecimento teórico de um assunto e contribuir com elementos em que os próximos trabalhos possam se apoiar em novos estudos.

Seu objetivo é descritivo, pois se baseia em assuntos teóricos tratados em livros, artigos científicos, monografias e teses. Assim, sua abordagem é qualitativa, pois se trata de analisar e avaliar lógica e racionalmente esses dados coletados por critérios definidos pelo próprio pesquisador com bases no tema da pesquisa.

Quanto ao método, utilizaremos o hipotético-dedutivo, onde através de hipóteses previamente criadas, são desenvolvidos os raciocínios lógicos que comprovam ou refutam essas mesmas hipóteses. Para isso, o procedimento bibliográfico documental oferecerá os dados vindos de trabalhos teóricos como comentado anteriormente e através de documentos que não receberam avaliação analítica como leis, relatórios e dados estatísticos.

3.2. Processo de pesquisa

Inicialmente foi realizado um levantamento das pesquisas sobre o ensino de programação com foco no ensino superior e técnico. Esse levantamento possibilitou a compreensão das principais dificuldades dos novatos e as principais propostas de soluções para sanar essas dificuldades. Para cada pesquisa, foi realizado seu fechamento com o resumo das principais ideias tratadas, citações e por último, catalogada sua referência bibliográfica em um arquivo geral de bibliografias.

Após o entendimento do cenário atual do ensino de programação, percebeu-se a necessidade de aprofundar os estudos voltados ao ensino técnico devida à baixa quantidade de trabalhos dedicados a esse e por ser a porta de entrada ao estudante

que se encontra por concluir seus estudos básicos. Com isso, foi possível levantar os objetivos principais dessa pesquisa que apontam para os desafios de ensinar programação em suas disciplinas introdutórias.

Houve uma tentativa sem êxito de tentar relacionar o ensino da linguagem natural com a linguagem de programação. Pensou-se que, por ambas serem linguagens, teriam implicações de aprendizagem correspondentes, no entanto, o levantamento de diversos estudos sobre a aquisição da linguagem natural, seja em sua concepção materna ou de segundo idioma, mostrou que as particularidades de cada tipo de linguagem necessitam de ferramentas e métodos próprios, isso se dá primordialmente pelo caráter artificial mecânico da linguagem de programação, a qual pode ser considerada como instrumento do programador para criar programas computacionais, ao invés de uma linguagem que proporciona comunicação entre duas partes.

O tempo utilizado na identificação da falta de correspondência entre linguagem natural e de programação, possibilitou que a direção da pesquisa fosse reajustada. Através do esforço em perceber suas diferenças, uma particular preocupação foi levantada em seu ensino: a de explicitar os fundamentos da programação ao novato no início de seu aprendizado. Pelo mesmo efeito, surgiu a necessidade de apontar a pesquisa na direção de uma proposta antes teórica, objetivando a organização curricular e sua dinâmica de ensino por sua sequência e contínuo relacionamento conceitual dos elementos que formam os fundamentos da programação. Devido a isso, obras que visam a compreensão teórica dos fundamentos da programação e propostas que se mostraram mais apropriadas ao ensino de novatos utilizando as ideias das teorias da aprendizagem significativa e currículo em espiral foram pesquisadas e fichadas como anteriormente.

Essa mudança de direção, mesmo que referente ao mesmo objeto de estudo, possibilitou a definição final do tema da pesquisa: *Ambiente de aprendizagem de programação imperativa no ensino técnico: Uma proposta de ensino em espiral*. E por conseguinte, foi possível descrever as perguntas norteadoras que dão clareza a quais resultados se quer alcançar: (i) Qual estrutura conceitual descreve os principais elementos da programação imperativa de forma que facilite a compreensão a alunos novatos explicitando os fundamentos da programação? (ii) Através de qual modelo teórico pode ser entendido o processo de aprendizagem de uma linguagem de

programação quanto a sua sequência e conteúdo tendo em vista a caracterização de novato do estudante?

Com o terreno pronto, o próximo passo foi analisar e avaliar as dificuldades e propostas levantadas anteriormente, e pelos objetivos designados, chegar a um resultado que respondesse às perguntas da pesquisa. Dessa forma, obteve-se como resultado da pesquisa a estrutura TFR, a qual possibilita classificar o estágio atual do novato baseado em critérios que explicitam os fundamentos da programação por sua complexidade. Para atender a dinâmica de ensino de forma que os conceitos elementares pudessem ser revistos continuamente, utilizou-se a abordagem em espiral aplicada à TFR, apresentando os conceitos dos mais simples para os mais complexos, pensando nos conhecimentos atuais do aprendiz e possibilitando a ampliação desses mesmos conceitos a cada nova volta da espiral.

Os limites da pesquisa estão em bem definir teoricamente o modelo conceitual da estrutura TFR e sua aplicação em espiral. Portanto, ultrapassa os limites da pesquisa identificar quais ferramentas, métodos e técnicas pedagógicas podem ser utilizados junto à espiral TFR. A variedade das abordagens lança propositalmente a ideia de que as estratégias utilizadas pelos docentes da área devem poder dialogar entre si quanto ao processo de aquisição da linguagem de programação e o estágio atual que o aprendiz se encontra. Dessa forma, poderemos ter melhores índices de comparação entre técnicas, já que a maioria, de alguma forma, diz ter melhor resultado comparada a outras.

3.3. Hipóteses

Através dos passos metodológicos citados e, por se tratar de uma pesquisa com objetivos descritivos e não exploratórios, ou seja, que ainda não foram testados em sala de aula, as hipóteses foram colocadas como pertencentes ao processo metodológico.

Tendo em vista as perguntas da pesquisa citadas anteriormente e das estratégias de pesquisa adotadas, três hipóteses são identificadas a serem discutidas como respostas a essas perguntas.

- **H1.** A teoria da aprendizagem significativa e de currículo em espiral são adequadas no ensino técnico de programação.

- **H2.** A organização em espiral do conteúdo possibilita que estudantes novatos desenvolvam melhor os conceitos estudados, revisitando e ancorando os mesmos a cada volta da espiral.

- **H3.** Estruturar os fundamentos da programação em: Tipo, Função e Rota, descreve as bases da programação imperativa e mantém a atenção do novato nos princípios elementares de composição e abstração.

4 ESTRUTURA TFR

Por mais sólidas que se encontrem as bases da programação, seu modelo de ensino diverge entre ferramentas, sequência, definição de conceitos e aprofundamento de seus elementos fundamentais. As diferenças em sua abordagem são passíveis de entendimento, já que cada nível de ensino preocupa-se em apresentar a programação conforme seus objetivos. Do maior para o menor, o ensino de programação está presente no (i) ensino superior, (ii) ensino técnico, (iii) ensino básico, (iv) cursos livres. Ainda pode-se apresentar seu caráter presencial ou à distância.

É compreensível que cada nível de ensino foque em questões diferentes já que possuem público com características e interesses diferentes. Mesmo a partir dessa divisão, haveria divergência quanto à sequência e aprofundamento de conceitos do mesmo nível de ensino (MEC, 2012), podendo essa, por sua vez, variar por instituição, curso, disciplina e até mesmo pelo próprio professor, isso porque cada um seguindo uma literatura que, obviamente varia em seus autores, e tendo o professor a autonomia dentro de sala de aula em propor o ambiente para que seja assimilado esse conhecimento, não é difícil concluir que cada estudante passe por processos de aquisição da linguagem diferentes.

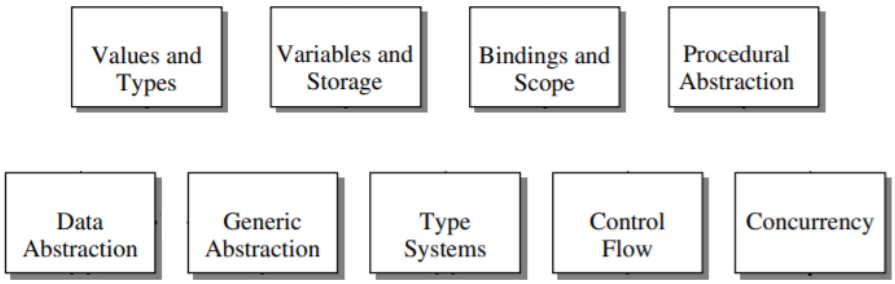
Tendo em vista que o ensino de programação deve ser melhorado, já que os artefatos tecnológicos crescem e mais profissionais qualificados são necessários à área, como pode ser discutido o processo de aquisição da linguagem de programação com tantas variações de abordagem? Quais critérios devem ser levantados para que se possa abrir as portas para o diálogo construtivo, não apenas acerca dos métodos e técnicas de ensino, mas principalmente sobre sua organização curricular, sua sequência de ensino e seus tópicos elementares que fazem parte dessa sequência? (SANTANA; ARAÚJO; BITTENCOURT, 2018).

Nos próximos itens será apresentada a estrutura TFR como resultado dessa pesquisa, servindo como modelo teórico que explicita os graus de complexidade dos elementos fundamentais da programação. Logo em seguida, será apresentada sua dinâmica em espiral, escolhida como proposta para o ensino de programação no nível técnico.

4.1. Fundamentos

Watt (2004) apresenta os fundamentos da programação nos primeiros capítulos de seu livro *Programming Language Design Concepts*, para logo em seguida aplicá-los nos principais paradigmas de programação. Os fundamentos da programação devem ser genéricos o suficiente para que possam pertencer a todos os paradigmas, porém quando aplicados a paradigmas específicos ganham especificidade e alguns conceitos se tornam chaves comparados a outros.

Figura 1 - Fundamentos da Programação



Fonte: Adaptado de Watt (2004, p. 16)

O termo imperativo vem do latim *imperare* o qual significa "comandar". Watt (2004, p. 265) define o paradigma imperativo como sendo "baseado em comandos, que atualizam variáveis mantidas em storages" e conclui dizendo que os principais conceitos desse paradigma são variáveis, comandos, procedimentos e abstração de dados.

Watt bem define os fundamentos da programação e seus paradigmas, no entanto, seu objetivo não era entender o processo de aprendizagem dos novatos em programação, mas clarear e formalizar os conceitos de programação. Para melhor entender esse processo, especificamente ao paradigma imperativo, é necessário criar um novo modelo, o qual chamaremos estrutura TFR (Tipo, Função e Rota). Seu objetivo será possibilitar a investigação de como ocorre a aquisição da linguagem de programação no paradigma imperativo e identificar os estágios que o novato percorre em seu aprendizado.

4.2. Tipo, Função e Rota

Os principais conceitos dos fundamentos da programação pela ótica do paradigma imperativo resumem-se em (i) sequenciar, (ii) selecionar e (iii) iterar sobre

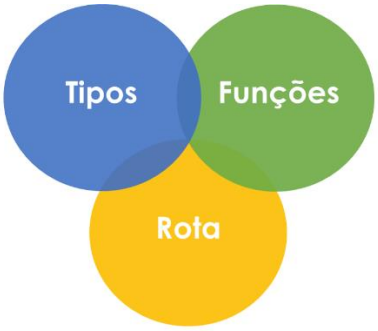
instruções da linguagem. Esse resumo é interessante quando o objetivo é demonstrar as características principais do paradigma, entretanto, de um ponto de vista que visa o processo de aprendizagem do aluno de modo a compreender cada estágio desse processo, torna-se um pouco geral, já que o item (i) esconde duas ideias que merecem ser explicitadas: atribuição de valores em variáveis pertencentes a um tipo e a aplicação de funções computacionais nesses mesmos valores (MOHOROVIČIĆ; STRČIĆ, 2011).

Dessa forma, pela necessidade didática de explicitar os tipos de dados e as funções que manipularão esses mesmos dados, chega-se à segunda divisão: (i) tipos, (ii) funções, (iii) seleção e (iv) iteração.

É preciso observar que seleção e iteração possuem uma relação muito estreita. Veja que uma iteração utiliza a seleção como base para satisfazer sua continuidade. Só isso seria suficiente para imaginar que seleção e iteração poderiam formar um único grupo, mas qual seria? Em uma definição mais objetiva, o que um estudante de programação deve entregar é um programa de computador. Um programa é um conjunto de algoritmos que, por sua vez, é um conjunto de instruções sequencializadas (KUNKLE, 2010). Pode-se pensar o algoritmo como os possíveis caminhos de instruções que computam e armazenam dados. Esses possíveis caminhos são determinados justamente pela seleção e iteração dessas instruções, formando o que será chamado de rotas.

A partir da ideia de rota, chega-se a uma visão que objetiva explicitar as atividades chave que compõem o aprendizado de programação, chega-se, então, à divisão final: (i) tipo (ii) funções e (iii) rota.

Figura 2 - Estrutura TFR



Fonte: O autor (2021)

Será chamada de estrutura TFR, ou simplesmente TFR, a divisão final que compõe a base da programação imperativa visando o processo de aprendizagem de uma linguagem de programação. Nos próximos itens, a preocupação será em estudar como utilizá-la para que o processo de aquisição da linguagem de programação possa ser compreendido, discutido e avaliado.

4.3. Composição

Definida a estrutura TFR que servirá de pilar para a discussão e compreensão do processo de aquisição de uma linguagem de programação, os esforços serão direcionados para dar corpo conceitual em sua estrutura para mais adiante serem analisados seus impactos na aprendizagem de programação.

A TFR explicita as principais habilidades do momento de codificação, mas como entender em qual estágio de aprendizado o aluno se encontra? Pode-se dizer que ele acabara de aprender sobre “tipo”? Que seu próximo passo é aprender “função” e seu aprendizado se encerra em “rota”? A resposta é não, ao invés de imaginar a TFR como o processo em si de aprendizado, deve-se olhá-la como os elementos que estão intrinsicamente relacionados a cada nova instrução codificada. Cada item da TFR representa uma classe de conceito que possui diversos itens, e em um ponto de vista mais prático, são esses itens que se relacionam (ABELSON, 1996).

A intensa relação dos itens da TFR e sua enorme variedade de elementos linguísticos subjacentes causam infinitas possibilidades no momento da codificação (FELLEISEN, 2001). Da mesma forma, não poder-se-ia exigir do aluno o domínio de todos os elementos ao mesmo tempo, isso se tornaria impraticável.

Surge a necessidade de estabelecer um caráter de ordem, porém ordenar todos os itens também se tornaria inalcançável, já que cada item da TFR pode conter dezenas de elementos subjacentes.

A partir dessas premissas, deve-se estabelecer um parâmetro que possa garantir a ordem desses elementos sem que seja necessário explicitar os mesmos, o que seria agir de forma muito específica e sem benefícios para a ordenação. Cada linguagem implementa sua própria sintaxe e semântica, e nomeia seus elementos a partir de seus próprios critérios (WATT, 2004). Estando a TFR direcionada ao paradigma imperativo, a qual contém uma grande variação de linguagens, não se pode agir, senão de forma mais geral.

Qual parâmetro é o mais adequado para classificar os elementos de um item da TFR possibilitando ordem no grau de aprendizado?

Considerando que estamos em um ambiente de aprendizagem e que deve-se iniciar em conceitos mais simples para mais os complexos (BREZOLIN, 2010), torna-se necessário eleger qual fundamento da programação causa maior ou menor complexidade. Essa complexidade, porém, do ponto de vista da programação, poderia tomar definições diferentes.

Em análise de algoritmos, complexidade é definida pela quantidade e tempo de instruções em um algoritmo, ou seja, quanto maior a quantidade de passos computacionais e o tempo para executá-los, maior é a complexidade do algoritmo (KELLER, 2001). Em arquitetura de software, complexidade é definida pela modularização, associação, composição, dependência, agregação, em outras palavras, pela forma de interação entre os componentes de um sistema (LEE, 2017).

Qual critério deve ser levantado para determinar a complexidade dos elementos que pertencem a um item da TFR, sabendo que ela se propõe a contribuir no ambiente pedagógico do ensino da programação? O aluno de programação, em seus primeiros passos, não pode julgar a complexidade de algoritmos, nem o impacto das escolhas da arquitetura de um sistema. Sua preocupação está concentrada em perceber os fundamentos da lógica, organizá-los de forma que raciocínios de sua linguagem natural possam ser convertidos em raciocínio de linguagem artificial somados às particularidades do paradigma que se encontra.

A partir dessas premissas, os conceitos de composição e abstração parecem ser os mais adequados para determinar a complexidade dos elementos pertencentes aos fundamentos da programação no momento da aprendizagem (ABELSON, 1996). Mas para os eleger como critérios de complexidade, é necessário investigar como serão interpretados e analisados nesse contexto.

4.3.1. Elementos Primitivos

Analisando o item "Tipo" da TFR, é comum que os primeiros elementos linguísticos, com os quais o aluno entra em contato são os designados por primitivos, ou seja, tipos que são indivisíveis (WATT, 2004), sua composição é a matéria prima com a qual os outros tipos se formam. Os tipos primitivos são essenciais para entender os objetivos da linguagem, seu rigor e sua forma de organização.

A linguagem Java, C# e C++ oferecem tipos diferentes para diferenciar valores que representam textos, booleanos, números inteiros e decimais. Possuem ainda, variações dentro desses conjuntos, como números inteiros e naturais, texto e caractere. A linguagem C oferece tipos para caractere, textos, números inteiros e decimais, mas não oferece tipo para booleanos. A linguagem JavaScript oferece tipo para booleanos, mas não diferencia números inteiros de decimais e texto de caractere.

Os tipos primitivos são a base indivisível que formarão outros tipos através do princípio de composição, aumentando o poder de abstração da linguagem, e são por conclusão, o primeiro grau de complexidade do item "Tipo" da TFR. O aluno deve, portanto, começar tomando ciência dos tipos primitivos, entendendo seus conjuntos de valores, seus limites e quais tipos de operações podem ser realizadas com eles.

Os tipos primitivos possuem operações que podem ser aplicadas entre seus valores, essas operações podem gerar um novo valor pertencente ou não ao tipo inicial. Essa afirmação traz a necessidade de abordar os elementos primitivos vistos pelo próximo item da TFR, as "Funções" (ABELSON, 1996).

Assim como os tipos primitivos, as funções primitivas são as funções indivisíveis da linguagem, muitas delas são chamadas de operadores e normalmente são divididos em operadores matemáticos, relacionais, lógicos, incrementação, decrementação, concatenação e indexação. Um operador é um literal que aponta para uma função procedimento da linguagem (WATT, 2004).

Atribui-se às funções primitivas o primeiro grau de complexidade do item "Função" da TFR, por serem a matéria prima para a construção de outras funções. Todas as funções primitivas podem ser aplicadas a tipos primitivos, formando a primeira inter-relação entre os elementos "Tipos" e "Funções" da TFR.

A essa inter-relação será dado o nome de instrução e a sua sequencialidade, algoritmo. Essa sequencialidade do algoritmo pode remontar diversos tipos de caminhos, ou seja, permitir que o programa tome rotas diferentes quanto às execuções das instruções (WATT, 2004).

O item "Rota" da TFR estabelece os possíveis caminhos, ou seja, as possíveis sequências de execução de instruções que um programa pode realizar baseado em estruturas de seleção e repetição que o compõem. Assim, podemos entender que o primeiro grau de complexidade desse item é exatamente a ausência dessas estruturas, onde o algoritmo passa por um caminho único.

A habilidade de planejar e prever esses caminhos é essencial para solucionar problemas cada vez mais complexos (SALLEH; SHUKUR; JUDI, 2013). Problemas que possuem um caminho único permitem ao aluno manter a concentração na relação "Tipos" e "Funções" da TFR e coordenar seus pensamentos para que a cada nova instrução, uma parte do problema se resolva.

Para o novato, não está claro quais relações entre "Tipos" e "Funções" devem ser feitas, ele ainda está descobrindo as possibilidades de combinação desses elementos (XIE, 2019) e adicionar a necessidade de caminhos diferentes, seja por seleção seja por repetição, traria dificuldades em seu aprendizado e é justamente essa a razão do primeiro grau de complexidade do item "Rota" da TFR ser a ausência dessas estruturas. Pelos conceitos de composição e abstração estabelecidos como fatores de complexidade, o caminho único é estabelecido, sendo o caminho indivisível, logo primitivo e as variedades de caminhos são permitidas pelas estruturas de seleção e repetição como responsáveis pela composição dos caminhos, logo, aumentando sua complexidade.

4.3.2. Elementos Compostos

Os graus de complexidade da tupla se baseiam nos conceitos de composição e abstração. O grau inicial indivisível para todas as linguagens de programação são seus elementos primitivos, portanto estão vinculados ao primeiro grau de complexidade.

Todo grau, a partir dos elementos primitivos, assume algum nível de composição. Por essa visão, tem-se apenas dois graus, o primeiro dos primitivos, e o segundo dos compostos. Essa divisão, no entanto, não exprime níveis intermediários de composição que ditam enormes mudanças de conceitos e de racionalizar os elementos da lógica de programação imperativa. Somente com essa divisão, seria difícil demonstrar um processo de aprendizagem que girasse em torno de apenas duas etapas (ABELSON, 1996).

Surge a necessidade de divisões intermediárias dos elementos compostos, a fim de explicitar o que será chamado de grau de complexidade da TFR. Os graus de complexidade também foram criados pelos princípios de composição e abstração dos elementos linguísticos de programação.

4.4. Graus de complexidade

Para identificar o estágio em que o aprendiz se encontra baseado na TFR, falta apresentar como são configurados seus graus de complexidade. Juntos, determinam o estágio que o aprendiz se encontra dentro dos conceitos elementares da linguagem de programação. Para cada item da TFR, serão apresentados seus graus de complexidade e descritas suas principais características. As divisões criadas foram resultado da análise dos trabalhos de Watt (2004) e Abelson (1996).

4.4.1. Item "Tipo" da TFR

01. Primitives: Tipos indivisíveis, normalmente representados por decimal, int, char, string, boolean. Exprimem as ideias base que podem ser computados números, textos e proposições.

02. Primitives Structures: Tipos definidos pelo agrupamento de tipos primitivos. Através da abstração de um conceito compõem sua representação estruturada, também são chamados de registros ou tipos cartesianos.

03. Compound Structures: Semelhante às estruturas primitivas, diferem-se por agruparem não apenas tipos primitivos, mas qualquer outro tipo composto.

04. Recursives: Tipos definidos em termos de si próprios. São compostos pelos componentes que remontam uma estrutura definida por um critério de sequência. As ideias de sequência e auto referenciamento em tipos recursivos fazem com que atinjam o último grau de complexidade, já que essas ideias se tornam confusas ao aprendiz em seu primeiro instante.

4.4.2. Item "Função" da TFR

01. Primitives: Formadas pelos operadores literais da linguagem. Esses operadores normalmente são categorizados em operadores matemáticos, relacionais, lógicos, incrementação, decrementação, concatenação e de indexação.

02. High level functions: Funções disponíveis que realizam operações comuns em tipos já existentes na linguagem. Quando em tipos primitivos, normalmente estão vinculadas aos tipos de texto e número, como forma de realizar cálculos matemáticos e manipular textos. Quando em tipos compostos, podem ser aplicadas em tipos recursivos como listas, filas, pilhas e dicionários como formas de

inserir, consultar, alterar e remover seus itens. E quando em estruturas, oferecem formas de manipular os itens compostos na estrutura.

03. Procedures: Funções não disponíveis pela linguagem, mas criadas pelo próprio desenvolvedor (*implementor*). Elas assumem um grau maior de complexidade, pois exigem que o desenvolvedor defina tanto sua assinatura quanto a implementação do corpo da função, exigindo habilidades lógicas de abstração para identificar seus argumentos de entrada e saída, além de habilidades algorítmicas para implementar seu código. Nas funções high level só a chamada da função era necessária (*application*), em procedimentos, o programador necessita modelar e implementar a função desde sua concepção.

04. Compound procedures: São funções que em sua implementação utilizam outras funções procedimento para realizarem partes de sua tarefa. Sua lógica é composta por instruções próprias e pela coordenação de outras funções procedimento.

05. Packages: São definidos pelo agrupamento de funções procedimento simples ou compostas com o objetivo de aumentar a coesão e o reaproveitamento de código. Nesse grau aparecem os princípios de abstração que o aluno tomará conhecimento em disciplinas que prezam as boas práticas da programação, como é o caso da arquitetura de software.

06. Recursives: São funções procedimentos compostas com a especificidade de utilizarem a si mesmas obrigatoriamente como parte de sua coordenação, ou seja, assim como nos tipos recursivos, são funções definidas em termos de si próprias. Frequentemente utilizam outras funções para compor sua lógica.

07. High Order: A abstração que envolve sua construção não se dá apenas sobre tipos, mas também sobre funções. Pode-se dizer que funções high order devem cumprir pelo menos dois critérios: (i) Um de seus argumentos deve ser uma função ou (ii) deve retornar uma função como resposta de sua chamada. Callbacks e Closures são exemplos de funções high order.

4.4.3. Item "Rotas" da TFR

01. Simple: A rota primitiva ou simples, possui apenas um caminho desde o início até o fim do algoritmo. Essa rota representa a ausência de composição de caminho justamente por ter apenas um único caminho.

02. Conditional: Representada por comandos de seleção que, a partir de uma expressão booleana, escolhe qual caminho percorrerá. Os comandos mais comuns são if/else e switch/case. Quanto mais seleções em um algoritmo, maior serão as possibilidades de caminhos. Sua denotação simples significa que logo após escolher uma instrução à outra, voltam ao caminho original. São desvios temporários que retornam ao caminho base.

03. Compound Conditional: Permite que uma seleção seja composta por outra seleção e assim sucessivamente. As possibilidades de caminhos em seleções compostas aumentam de forma exponencial. A diferença para a seleção simples é que após escolherem uma instrução não voltam ao caminho original, mas se deparam com novas escolhas, gerando novos caminhos.

04. Looping: Representados por comandos de iteração, possuem um subcomando chamado body, que é o conjunto das instruções que serão repetidas pelo comando de iteração. Os comandos são categorizados em (i) iterações definidas, onde é possível saber antecipadamente a quantidade de iterações; (ii) iterações indefinidas, onde não é possível saber antecipadamente a quantidade de iterações. A complexidade desse grau está em planejar as instruções que serão repetidas para cumprir uma tarefa mesmo sem saber de fato quantas vezes esse caminho será repetido.

05. Compound Looping: Permite que uma iteração seja composta por outra iteração e assim sucessivamente. Em outras palavras, o subcomando body possui outro comando de iteração. A quantidade de caminhos em iterações compostas é o grau mais alto que pode-se chegar devida à impraticabilidade de remontar mentalmente as instruções que estão sendo executadas pelo computador, além de possibilitar a combinação com rotas condicionais no subcomando body.

A figura abaixo mostra o resumo dos graus de complexidade da TFR, onde o item "Tipo" da TFR possui 4 graus de complexidade, o item "Funções" possui 7 e o item "Rotas" conta com 5 graus de complexidade. O aprendiz de programação, após dominar todos os graus de complexidade e compreender os conceitos semânticos e sintáticos da linguagem em questão, pode considerar ter deixado sua característica de novato e continuar sua busca em ser expert da programação.

Tabela 2 - Graus de Complexidade da TFR

	Tipos		Funções		Rota
1	Primitives	1	Primitives	1	Simple
2	Primitives Structures	2	High level Functions	2	Conditional
3	Compound Structures	3	Procedures	3	Compound Conditional
4	Recursives	4	Compound procedures	4	Looping
		5	Packages	5	Compound Looping
		6	Recursive		
		7	High order		

Fonte: O autor (2021)

Conforme visto, os graus se apoiam nos princípios de composição e abstração, e representam categorias de complexidade que possuem dezenas de representações linguísticas. Por sua vez, não é necessário que um grau seja coberto integralmente de seus elementos linguísticos para ser possível o avanço a um novo grau de complexidade (BRUSILOVSKY, 1994). Esse movimento é antes relativo à estratégia pedagógica que o professor planeja do que uma lei a ser seguida, isso torna o modelo TFR um guia para o diálogo entre as estratégias adotadas pelos docentes de programação.

Com o modelo TFR melhor entendido e seus graus bem descritos, é preciso identificar como ocorre o movimento de avanço nos graus de complexidade e de aprofundamento de seus conceitos, e como são conectados aos conhecimentos prévios do aprendiz, as maneiras de planejar a sequência de ensino, e por fim, critérios para avaliar se o aprendiz está pronto para avançar ao próximo estágio. Essas preocupações serão vistas a seguir, e resultarão em uma proposta de ensino de programação no nível técnico baseado em um modelo espiral da TFR.

5 ESPIRAL TFR

Três requisitos para compreender o processo de aquisição da linguagem de programação dentro do paradigma imperativo foram bem definidos nos itens anteriores: (i) estabelecer os fundamentos da programação imperativa, (ii) identificar como esses fundamentos se relacionam e como agrupá-los de forma que representem as habilidades explícitas no ato de programar, e (iii) avaliar o grau de complexidade de cada grupo baseado em sua composição e abstração.

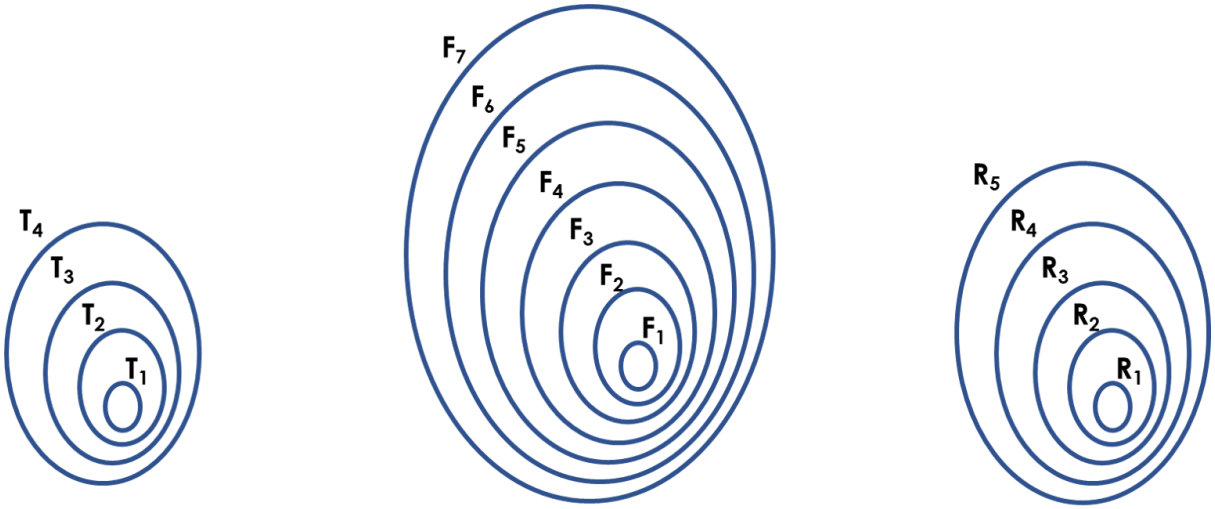
Como já dito, o contínuo relacionamento entre os itens da TFR é a base para a criação de um algoritmo. Logo, o aprendiz não poderia concentrar seu aprendizado em apenas um de seus itens, conquistando o conhecimento de todos os graus de complexidade para, em seguida, partir para o próximo item. Isso pela razão de não poder criar um algoritmo sem que esse inter-relacionamento ocorra concomitantemente. Assim, o aprendiz deve avançar simultaneamente em todos os itens da TFR para que seu primeiro programa seja escrito. É a partir desse momento que o processo de aprendizagem se inicia (VALENTE, 2005).

Quais elementos da linguagem devem ser expostos ao aluno em seus primeiros passos? Existe uma sequência única para ensinar programação? Como medir o grau de aquisição da linguagem de um aluno? Essas perguntas seriam extremamente difíceis de responder se não tivéssemos os requisitos comentados acima bem definidos.

5.1. Análise do modelo TFR

Pode-se resumir as ideias de graus de complexidade da TFR na visão de conjuntos. A figura abaixo demonstra os graus de complexidade na visão de conjuntos:

Figura 3 - Visão de conjuntos da TFR



Fonte: O autor (2021)

Pode-se observar a hierarquia dos graus de complexidade pela organização dos conjuntos, onde um conjunto com grau de complexidade maior contém um conjunto com grau de complexidade menor. É possível definir essa afirmação através da notação abaixo, onde *E* representa um item da TFR e *n* seu grau de complexidade:

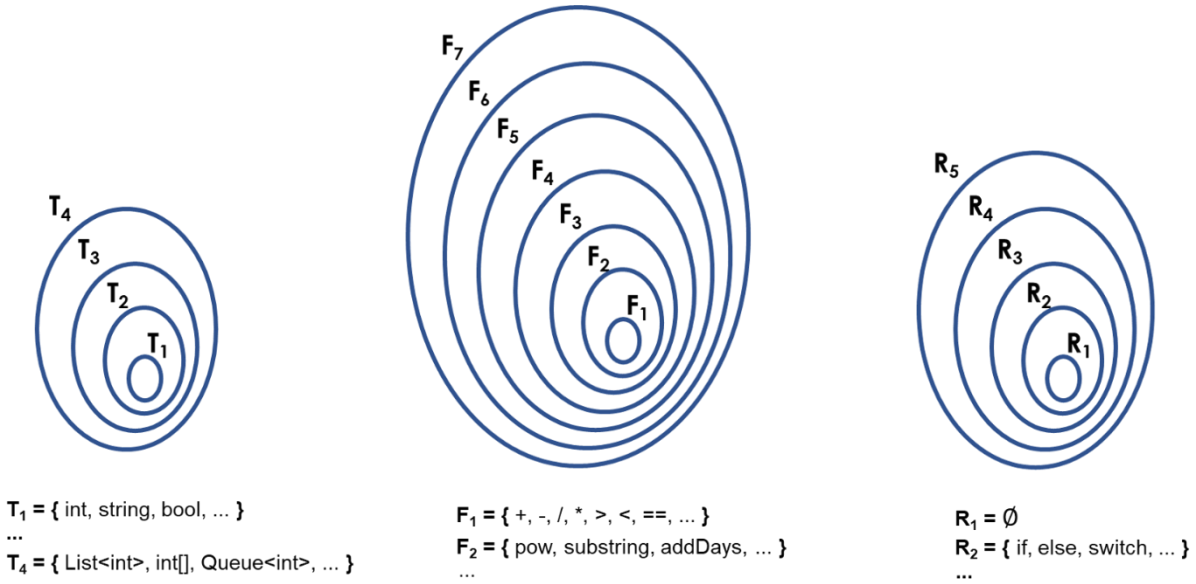
$$\mathrm{E_n} \supseteq \mathrm{E_{n-1}}$$

Com exceção dos graus primitivos:

$$\mathrm{E_0} \supseteq \emptyset$$

Dessa forma, fica claro que o aprendizado deve começar dos menores graus de complexidade e ir avançando até chegar ao último grau de todos os itens da TFR, ou seja, cada avanço implica em um aumento de complexidade (AUSUBEL, 1968). Ainda deve ser considerada a possibilidade de haver inúmeros elementos da linguagem de programação em um único grau.

Figura 4 - Elementos dos conjuntos da TFR



Fonte: O autor (2021)

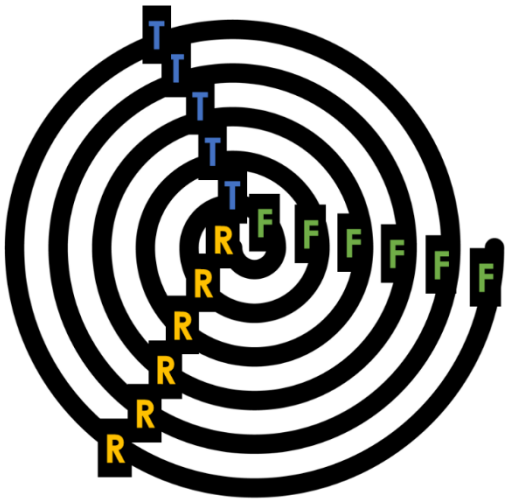
O aluno deve dominar todos os elementos de um grau para passar ao próximo? Todos os itens da tupla devem avançar um grau de complexidade ao mesmo tempo? Como avaliar se um aluno pode avançar para o próximo grau? Essas perguntas podem ser respondidas se estiver aplicada à estrutura TFR o modelo em espiral, o qual permite o avanço gradual de conceitos.

5.2. Espiral TFR

Foi apontado o modelo em espiral como mais adequado para o processo de aprendizagem de programação. O modelo em espiral traz a ideia de que os mesmos conceitos são vistos durante todo o processo de aprendizagem, com a diferença que a cada volta na espiral, aprofunda-se o conhecimento sobre esses conceitos e aumenta-se o grau de complexidade (BRUNER, 1966).

A espiral TFR consiste em colocar os elementos "Tipo", "Função" e "Rota" na espiral de forma que a cada nova volta, seja possível aprofundar os conceitos de cada item, seja em complexidade ou em itens linguísticos (ARAUJO, 2018). A imagem abaixo demonstra os itens da estrutura TFR em espiral:

Figura 5 - Espiral TFR



Fonte: O autor (2021)

A espiral TFR é o modelo que guia em qual estágio o aprendiz se encontra em seu processo de aquisição de uma linguagem de programação.

Estando os itens da TFR separados na espiral, podemos aumentar a complexidade de forma independente, ou seja, não é necessário que todos os itens aumentem sua complexidade ao mesmo tempo, basta que um item aumente sua complexidade para que uma nova volta na espiral possa ser realizada. É possível perceber através dessa dinâmica que as metodologias e estratégias de ensino de programação divergem muito em seu processo (BRUNER, 1966).

Nesse sentido, o fator determinante no processo de ensino de linguagem de programação é a forma que o professor planeja as voltas da espiral TFR (AUSUBEL, 1968). Fatores de ensino como métodos, técnicas e ferramentas influenciam na aprendizagem, entretanto, esse trabalho visa compreender o processo que o aluno percorre para adquirir conhecimento do corpo conceitual dos fundamentos da programação e possibilitar uma forma de medir em qual estágio desse processo o aluno se encontra, além de abrir portas para o diálogo entre metodologias e técnicas no ensino de programação, já que os fundamentos foram agrupados e classificados conforme princípios presentes em toda linguagem de programação, composição e abstração (WATT, 2004).

Para discutir as técnicas de planejamento da espiral TFR, precisa-se antes aprofundar alguns aspectos sobre sua composição. Compreender melhor como os itens de uma volta na espiral podem ser adicionados, ou melhor, incrementados, dará

a base para planejar a espiral volta a volta, conforme as especificidades da disciplina, da instituição e das estratégias pedagógicas do professor.

5.3. Composição da espiral TFR

Cada volta da espiral é composta pela união dos itens linguísticos que expressam os fundamentos da programação agrupados pelos itens da TFR. Pode-se expressar a composição de uma volta pela fórmula abaixo:

$$E = T \cup F \cup R$$

Onde *E* representa a volta da espiral, *T, F e R* representam os itens linguísticos da volta, estando cada item vinculado a um grau de complexidade.

No entanto, essa fórmula não apresenta a ideia essencial de uma volta na espiral: Identificar em qual grau de complexidade cada item se encontra. Dessa maneira, é necessário melhorar a definição para a fórmula abaixo:

$$E = T_x \cup F_y \cup R_z$$

Sendo *x, y e z* os graus atribuídos a cada item da TFR.

Através dessa definição, deduz-se, no mínimo, a primeira e a última volta da espiral, em outras palavras, o início e o fim do processo de aprendizagem de uma linguagem de programação.

Reiteramos que de forma similar às linguagens naturais, os idiomas, a linguagem de programação também possuem aspectos sociais e pode ser vista em contínua evolução (VIGOTSKI, 1978). A comunidade que apoia aquela ou essa linguagem a impulsiona para que novas tecnologias, plataformas e frameworks contribuam com sua longevidade. Essa mesma comunidade desenvolve artefatos que podem posteriormente serem adicionados oficialmente ao core da linguagem. Por essas e outras razões, uma linguagem ganha com o tempo novos operadores, expressões e possibilidades de aumentar sua abstração e performance.

Por sua vez, a primeira e última volta da espiral podem ser expressadas pelas fórmulas:

$$E_{min} = C_1 \cup T_1 \cup A_1$$
$$E_{max} = C_4 \cup T_7 \cup A_5$$

É fácil observar que a primeira volta deve conter em sua totalidade o primeiro grau de complexidade e que a última volta deve abranger o maior grau de complexidade para todos os itens.

Essa visão nos permite ver a evolução no que diz respeito ao grau de complexidade acrescentado a cada item da TFR por volta da espiral, porém, limita a visão em identificar quais itens linguísticos foram escolhidos em cada grau. Ou seja, tem-se uma visão que não contempla as particularidades da linguagem de programação escolhida. Em geral, não é necessário que essa especificidade seja apresentada no planejamento da espiral, contudo, para detalhar a composição de uma volta para contemplar os itens linguísticos, será preciso de uma nova fórmula para calcular o grau de complexidade da volta da espiral.

5.4. Cálculo de complexidade de uma volta na espiral TFR

Para calcular a complexidade de uma volta da espiral, existem duas variáveis a serem aplicadas nos itens da tupla. A primeira é o grau de complexidade já definido anteriormente; a segunda é a quantidade de elementos linguísticos escolhidos na linguagem em questão, vinculados aos graus de complexidade.

A quantidade de itens impacta a complexidade da volta na espiral por um motivo simples. Ao ensinar "Tipos Recursivos", algumas linguagens oferecem dezenas desses tipos, os mais comuns são vetor, matriz, lista, fila, pilha e dicionário, nesse contexto, o professor pode escolher ensinar vetor, lista, fila e pilha em uma volta da espiral e introduzir matriz e dicionário em uma próxima volta. Da mesma forma, outro professor pode escolher apresentar todos os tipos em uma mesma volta. Não seria justo dizer que o aprendiz está exposto ao mesmo grau de complexidade da volta da espiral nas duas abordagens, até porque cada tipo possui suas próprias funções e sua lógica de como sequenciar seus componentes internos.

Assim, tem-se a fórmula que considera as duas variáveis para cálculo da complexidade da volta da espiral por item da TFR.

$$\mathrm{C} = \sum_{g=1}^{n} (g + e_g . 0{,}01)$$

É necessário observar que a fórmula acima deve ser aplicada para cada item da TFR, ou seja, após calcular a complexidade para cada item da TFR com a fórmula acima, basta somar as complexidades para se obter a complexidade da volta da espiral.

$$\mathrm{C_E} = \mathrm{C_t} + \mathrm{C_f} + \mathrm{C_r}$$

A fórmula acima contém: *C* representando a complexidade de um item da TFR em uma volta da espiral. Cada item da TFR possui graus diferentes e cada grau conta com sua própria quantidade de itens linguísticos. Assim, para calcular a complexidade basta que somemos a complexidade para cada grau e quantidade de itens linguísticos correspondentes.

O grau possui peso maior que a quantidade de itens, por isso, a quantidade de itens, podendo ser dezenas, é multiplicada por 0,01 para que não ultrapasse o peso atribuído ao grau. Com isso, evitamos que graus menores com muitos itens linguísticos ultrapassem graus maiores com poucos.

5.5. Planejamento da Espiral

Voltando às perguntas que guiam esta pesquisa: Qual estrutura conceitual descreve os principais elementos da programação imperativa de forma que facilite a compreensão a alunos novatos explicitando os fundamentos da programação? Através de qual modelo teórico pode ser entendido o processo de aprendizagem de uma linguagem de programação quanto à sua sequência e conteúdo?

Essas perguntas esbarram em sutilezas metodológicas de ferramentas e técnicas de apresentação de conteúdo, na didática, no ambiente de aprendizado, no

público discente, no objetivo da instituição e do curso onde a linguagem é lecionada (ALLEH; SHUKUR; JUDI, 2013).

O aprofundamento de um conteúdo depende desses e outros fatores. Quanto mais aprofunda-se um conteúdo, mais pode-se explicar seus fenômenos, suas leis, suas exceções, ou seja, pode-se dizer que o domina mais ou menos.

O processo de aquisição da linguagem de programação visto como uma espiral baseada na estrutura TFR onde a cada volta, adicionam-se graus de complexidade correspondentes à composição e quantidade de itens linguísticos, permite medir o aprendizado do estudante de programação dizendo em qual volta da espiral ele se encontra e a qual complexidade ela está atribuída.

O modelo em espiral, portanto é genérico o suficiente para abranger linguagens e métodos de ensino diferentes e possibilita que o diálogo pedagógico entre eles estejam uniformizado pelas medidas resultantes do cálculo de complexidade.

O diálogo surge no momento que cada sistema de ensino planeja sua espiral de forma diferente a outro. Para planejar uma espiral, deve-se usar o esquema a seguir:

$$\mathrm{E_{x,y,z}} = \mathrm{T_x} \cup \mathrm{F_y} \cup \mathrm{R_z}$$

Destaca-se aqui o acréscimo da fórmula às variáveis *x, y e z* a entidade *E*, assim o planejamento da volta da espiral pode ser resumido sem explicitar o corpo que define a fórmula. Por exemplo, pode-se planejar as voltas da espiral conforme abaixo:

$$\mathrm{E_{1,1,1}} \rightarrow \mathrm{E_{1,2,1}} \rightarrow \mathrm{E_{1,2,1}} \rightarrow \cdots \rightarrow \mathrm{E_{4,7,5}}$$

Esse esquema de planejamento permite a visualização de forma macro, a estratégia adotada para o ensino de programação (FELLEISEN, 2001). Esse planejamento deve sempre começar dos graus menores até chegar aos maiores (AUSUBEL, 1968). Nesse exemplo, as três primeiras voltas aumentaram apenas a complexidade do item "Funções" da TFR, interpretando esse planejamento, vê-se que o estudante se familiarizará com os tipos primitivos e rotas simples, sendo exposto a uma diversidade de funções para desenvolver as habilidades de raciocinar sobre a

combinação dessas funções e iniciar os raciocínios abstratos que envolvem implementar suas próprias funções. Em contraste, há o seguinte planejamento:

$$\mathrm{E}_{1,1,1} \rightarrow \mathrm{E}_{1,2,2} \rightarrow \mathrm{E}_{1,3,3} \rightarrow \mathrm{E}_{2,3,4} \rightarrow \cdots \rightarrow \mathrm{E}_{4,7,5}$$

Nesse exemplo, as ideias de rotas condicionais são apresentadas mais cedo logo na segunda volta da espiral. Também podemos ver tipos e funções compostos sendo apresentados junto às rotas de repetição na quarta volta da espiral.

Qual planejamento é melhor? Qual planejamento é mais conservador ou arriscado? Existe um consenso único no ensino de programação? Essas perguntas só poderão ser respondidas em uma visão de planejamento micro e no que será chamado de avaliação da espiral.

No planejamento micro serão informados quais itens serão estudados em cada volta para os graus informados:

Tabela 3 - Planejamento detalhado espiral

$\boldsymbol{E_{1,1,1}}$	$\boldsymbol{T_1} = \{int, string, decimal\}$ $\boldsymbol{F_1} = \{+, -, *, /\}$ $\boldsymbol{R_1} = \{\emptyset\}$
$\boldsymbol{E_{1,2,2}}$	$\boldsymbol{T_1} = \{int, string, decimal, bool\}$ $\boldsymbol{F_1} = \{+, -, *, /, >, \geq, <, \leq, ==, ! =\}$ $\boldsymbol{R_1} = \{\emptyset\}$
	$\boldsymbol{T_2} = \{\emptyset\}$ $\boldsymbol{F_2} = \{Math.Pow, Math.Sqrt, string.Substring, string.IndexOf\}$ $\boldsymbol{R_2} = \{if, else\ if, else\}$

Fonte: O autor (2021)

Essa visão explica os elementos linguísticos de cada volta da espiral e permite que seja discutido com mais clareza os porquês de cada decisão.

No entanto, ainda é necessário um processo que conclua o término da volta da espiral, de forma que possa garantir a assimilação do conhecimento pelo estudante e confirme que ele é capaz de criar algoritmos que combinem os elementos apresentados nesse e nos graus anteriores (VALENTE, 2005).

Essa verificação avaliativa pode ser feita através de diversas técnicas: exercícios teóricos, práticos, projetos, trabalhos, apresentações, discussões, debates, etc. A característica essencial do processo avaliativo da espiral é que haja ancoragem entre as voltas anteriores. A ancoragem é a essência da espiral, ela garante que: (i) os conteúdos anteriores fiquem sempre ativos na memória do estudante, (ii) as lacunas de compreensão deixadas em voltas anteriores sejam preenchidas, (iii) aumente a familiarização dos conceitos para que se tornem hábitos inconscientes, (iv) melhore o raciocínio lógico de combinação dos elementos para solução de problemas.

5.6. Avaliação da espiral

Como apresentado, o processo em espiral mostra-se um bom modelo para medir o estágio em que o aprendiz se encontra para que adquira os conhecimentos fundamentais da programação expressos em uma linguagem. Com o planejamento finalizado, o aprendiz coloca-se na posição inicial da espiral e inicia seu percurso rumo à aquisição da linguagem. Em qual momento o aprendiz está pronto para a próxima volta da espiral? Quais garantias o professor tem de que as habilidades mínimas necessárias foram assimiladas pelo estudante? Como estabelecer que os conceitos antigos não foram adormecidos entre os novos apresentados? Essas indagações levam ao último processo que deve ser considerado na espiral, a avaliação (PIMENTEL, 2003).

Após planejar a sequência da espiral e aplicar o conteúdo com os métodos escolhidos, o professor deve verificar se o conteúdo foi assimilado para que então, o aluno avance para a próxima volta da espiral. A avaliação é a etapa final do planejamento da espiral, ela deve assegurar as seguintes premissas: (i) o aluno assimilou os conceitos adicionados à espiral, (ii) o aluno não adormeceu os conceitos anteriores à espiral, (iii) o aluno é capaz de relacionar os conceitos anteriores aos novos, proporcionando-lhes ampliação de significado (KRUG, 2018).

Conforme mencionado, são denominadas habilidades mínimas necessárias, porque dificilmente o aluno conseguirá dominar um conceito enquanto não puder combiná-lo com outros, gerando as estruturas lógicas que possibilitam a criação de instruções exclusivamente novas, do contrário, o programador ficaria preso às instruções que lhe foram expostas, e é exatamente essa a característica daquele que

domina a linguagem, isto é, combinar os elementos linguísticos de forma inovadora e válida para expressar as ações que deseja que a máquina execute (ABELSON, 1994).

Portanto, o processo de avaliação deve criar cenários desafiadores para que o aluno possa combinar antigos e novos conceitos. Esses cenários podem ser em formato teórico, prático, individual, em grupo, prova, trabalho ou qualquer outro instrumento que fizer parte do método adotado pelo professor (LEMOS; LOPES; BARROS, 2005).

Metodologias de avaliação não fazem parte do escopo dessa pesquisa, mas são de suma importância para aprofundar o entendimento do tema escolhido. Os métodos e técnicas devem ser estudados em próximas pesquisas, tendo como base a dinâmica da espiral como guia da aprendizagem do aluno.

6 GUIA TFR: CASO DE USO COM A LINGUAGEM CSHARP

Nesse capítulo será abordado como o professor pode utilizar a estrutura TFR para planejar o processo de aquisição da linguagem de programação. A linguagem escolhida é a CSharp (C#) criada pela *Microsoft* em julho de 2000, atualmente encontra-se na versão 8.0 lançada em setembro de 2019. Primariamente pertencente ao grupo das linguagens imperativas, a partir da versão 3.0 e mais tarde na versão 7.0 ganhou recursos que possibilitam expressões pertencentes às linguagens funcionais. As abordagens incluídas nesse guia não estão vinculadas à linguagem C#, mas podem ser aplicadas a qualquer linguagem que possua características do paradigma imperativo como Java, Python, C, Ruby e outras.

Esse guia será contextualizado para o ensino de nível técnico na disciplina Fundamentos da Lógica. A organização escolhida chama-se Instituto Social Nossa Senhora de Fátima (INSF), na qual o autor leciona atualmente. A instituição oferece 7 cursos técnicos e de qualificação profissional. Nela, o curso Técnico em Informática está associado a CNCT com carga horária de 1.200 horas e a disciplina Fundamentos da Lógica possui carga horária total de 80 horas, sendo dois módulos de 40 horas. A descrição das habilidades e competências da disciplina encontram-se no Anexo 1.

Por tratar-se de um guia, outras formas de aplicação a TFR são encorajadas e devem ser pesquisadas em trabalhos futuros, apresentamos esse guia para que possa ser utilizado de forma experimental por professores da área com o objetivo de coletar novas observações e contribuir com o estudo da TFR.

Esse guia não tem como objetivo apontar técnicas de apresentação e avaliação de conteúdo, estudadas em outros trabalhos sobre o aprendizado de programação. A TFR busca entender o processo de aprendizagem de programação pela ótica de seus fundamentos, ou seja, o caminho que o novato percorre enquanto aprende os fundamentos essenciais, base para compreensão da lógica presente em todas as linguagens imperativas.

O guia TFR será dividido em ações *pré-espiral*, ou seja, ações que devem ser realizadas antes de iniciar o processo de aprendizagem, e *in-espiral*, no qual encontram-se as ações que devem ser realizadas durante o processo de aprendizagem.

6.1. Pré-Espiral

As ações pré-espiral correspondem ao momento antes do processo de aprendizagem e estão relacionadas às escolhas, decisões e planos que o professor estabelece antes do início dos dias letivos. A pré-espiral deve ser entendida como momento decisivo e inadiável para a elaboração do planejamento do processo em que os elementos essenciais da linguagem são levantados, a sequência de conteúdo é formada e os conceitos bases são estabelecidos.

As quatro ações que definem a pré-espiral são: (1) Identificar os objetivos da disciplina, (2) escolher o ambiente de programação, (3) vincular os elementos linguísticos à TFR, e (4) planejar a espiral TFR.

6.1.1. Identificar os objetivos da disciplina

Cada instituição oferece cursos específicos que podem aprofundar mais ou menos certas áreas da ciência da computação, por exemplo o curso de Eletrônica pode conter em sua grade a disciplina de programação assim como o curso Ciência da Computação, mas é simples observar que o foco e aprofundamento desejado entre os cursos divergem devido aos objetivos gerais do curso.

Torna-se necessário que o professor de programação recupere no plano de curso os objetivos gerais quanto às competências e habilidades que o aluno deve desenvolver na disciplina. Abaixo apresenta-se o resumo das competências e habilidades para a disciplina Fundamentos da Lógica oferecida no curso Técnico em Informática do Instituto Social Nossa Senhora de Fátima.

Tabela 4 - Plano de Ensino - Fundamentos da Lógica

	Objetivos
Competências	- Desenvolver algoritmos através de divisão modular e refinamento sucessivos. - Distinguir e avaliar linguagens e ambiente de programação, aplicando no desenvolvimento de software. - Interpretar pseudocódigos, algoritmos e outras especificações para codificar programas. - Avaliar resultados de testes dos programas desenvolvidos. - Integrar módulos desenvolvidos separadamente. - Compreender o paradigma de orientação por objeto e sua aplicação em programação.
Habilidades	- Selecionar e utilizar estrutura de dados na resolução de problemas computacionais. - Utilizar editores de textos, planilhas, gerenciadores de bases de dados, compiladores e ambientes de desenvolvimento na elaboração de programas. - Utilizar modelos, pseudocódigos e ferramentas na representação da solução de problemas. - Elaborar e executar casos e procedimentos de testes de programas. - Redigir instruções de uso dos programas implementados. - Aplicar as técnicas de programação (orientada a objeto, estruturadas e outras).

Fonte: O autor (2021)

Desenvolver, interpretar e avaliar algoritmos são habilidades chave da disciplina de lógica de programação, bem como implementá-los utilizando pseudocódigos ou linguagens de programação. Selecionar e distinguir modelos e estruturas de dados indicam raciocínios complexos voltados a elementos específicos da programação, é necessário compreender bem o que são, sua formação e como utilizá-las para poder selecionar, distinguir e criar novas estruturas de dados. A partir desses pontos chave, avança-se para a escolha da linguagem.

6.1.2. Escolher o ambiente de programação

O tema de qual linguagem de programação deve ser usada para uso dos novatos em programação é discutido em trabalhos como o de Araújo, Bittencourt e Santos (2018), além de ser um tema levantado nas universidades e escolas técnicas que causam divergência entre os professores, muitas vezes pela familiaridade maior ou menor entre eles.

A linguagem de programação não pode estar divergente com a proposta da disciplina, por exemplo não se pode priorizar a linguagem C em uma disciplina que trata sobre orientação a objetos, da mesma forma deve-se priorizá-la em uma disciplina que trata sobre automação com uso de Arduino. Esse pensamento mais

exclui possibilidades do que dita a linguagem perfeita para a disciplina, todavia não cabe a esse guia voltar à discussão, mas apontar a necessidade da escolha da linguagem de programação para que possa ser realizado o planejamento da espiral posteriormente.

6.1.3. Vincular os elementos linguísticos à TFR

Antes de planejar a espiral TFR, o professor necessita realizar um esforço para vincular os elementos da linguagem de programação escolhida à estrutura TFR. O vínculo consiste em atribuir termos chave ou comandos da linguagem de programação aos diversos níveis da estrutura TFR. A conclusão dessa tarefa resultará em um mapa linguístico dos fundamentos da linguagem segregados por complexidade. Além do vínculo, o professor estabelece quais conceitos essenciais pertencem a cada item da TFR, assim esses conceitos ficam explícitos e serão trabalhados e definidos junto aos novatos conforme o andamento da espiral.

A figura abaixo mostra o vínculo linguístico da linguagem CSharp ao item Tipo da TFR.

Tabela 5 - Vínculo CSharp x TFR Tipo

Tipo	
1. Primitives	short, int, long, ushort, uint, ulong, float, decimal, double, bool, DateTime, TimeSpan, string, char, byte
2. Primitives Structures	struct, class
3. Compound Structures	struct, class
4. Recursives	Array, List, Queue, Stack, Dictionary, Hashtable, HashSet
Conceitos	Type System, Type, Value, Instance, Literal, Variable, Limits, Pointer, Binding, Scope, Environment

Fonte: O autor (2021)

Para o item "Tipo" da TFR pode-se iniciar mapeando os tipos primitivos da linguagem, base para a criação de todos outros tipos. As linguagens podem variar em alguns aspectos sobre seus tipos primitivos, algumas linguagens por exemplo, não apresentam tipos para valores sem sinal (*unsigned*), por isso é importante que os tipos

primitivos sejam identificados pelo professor para que o aluno tenha as primeiras impressões e consiga entender o processo de composição e abstração que ocorre quando avançamos em complexidade.

Para os tipos compostos, decidiu-se mapear apenas as palavras chave para a criação das estruturas primitivas e compostas, no entanto, poder-se-ia mapear tipos compostos disponibilizados pela linguagem como *Point, Size, Rectangle*, e outros. Para os tipos recursivos, são mapeados alguns dos principais tipos disponibilizados na linguagem, que pode oferecer dezenas, cabe ao professor apontar quais serão indispensáveis na disciplina e alertar os alunos sobre essa diversidade. Além de tipos recursivos oferecidos pela linguagem, o professor poderia propor a implementação de um novo tipo recursivo utilizando os conceitos desenvolvidos na TFR.

Como ficará explícito posteriormente no tema planejamento macro, alguns graus de complexidade não possuem vínculo direto a elementos linguísticos, estão mais relacionados aos conceitos fundamentais do item da TFR. Esses conceitos também devem ser mapeados pelo professor e geralmente serão muito semelhantes entre as linguagens, por exemplo os conceitos de *Variáveis, Escopo e Ambiente* estão presentes em todas as linguagens, mesmo que seu significado possa variar em alguns aspectos dependendo do sistema de tipos utilizado.

Tabela 6 - Vínculo CSharp x TFR Função

	Função
1. Primitives	+, -, *, /, +=, -=, *=, /=, ++, --, >, >=, <, <=, ==, !=, !, &&, ||, ?:
2. High level Functions	substring, indexOf, replace, contains, toUpper, toLower, trim, pow, sqrt, round, abs, ceiling, floor, truncate, addDays, addMonths, addYears, where, first, last, select, any, all, take, skip
3. Procedures	public, private, protected, params, in, out, ref, return
4. Compound Procedures	-
5. Packages	struct, class
6. Recursives	-
7. High Order Functions	delegate, Action, Func
Conceitos	Evaluation, Parameter, Argument, Function/Procedure, Arity, Notation, Procedence, Scope, Environment

Fonte: O autor (2021)

Assim como no item "Tipo", em "Função" inicia-se pelas funções primitivas, geralmente operadores matemáticos, relacionais e lógicos. Os operadores também variam entre as linguagens e podem oferecer maiores facilidades ao desenvolvedor, por exemplo, algumas linguagens oferecem o operador (**) para realizar exponenciações. Em seguida é necessário realizar o vínculo de funções bases disponibilizadas pela linguagem para realizar operações com texto, número e data. A linguagem pode oferecer dezenas de funções *high level*, por isso, novamente é necessário que o professor delimite e oriente os alunos conforme as diretrizes do plano de curso.

Para os itens 3, 4, 5, 6 e 7 são mais importantes os conceitos descritos na tabela como *Parâmetro, Argumento, Escopo e Ambiente* do que propriamente palavras chave para sua criação. Nesses itens, a ideia de abstração é fundamental, dessa forma, a apresentação desses conceitos na *in-espiral* deve ser realizada de forma clara, explorando a maior quantidade de situações computacionais.

Tabela 7 - Vínculo CSharp x TFR Rota

Rota	
1. Simple	-
2. Conditional	if, else if, else, switch, case, default
3. Compound Conditional	-
4. Looping	for, foreach, while, do while
5. Compound Looping	-
Conceitos	Chained, Nested, Counter, Increment, Definite/Indefinite/ Infinite Loop

Fonte: O autor (2021)

Não existem elementos primitivos em "Rota", já que o primeiro nível refere-se à ausência de rotas. No grau "Condicional" são mapeados os comandos de seleção da linguagem, e em "Looping" os comandos de iteração. Ambos dificilmente variam entre as linguagens, senão por sua sintaxe. Os conceitos também devem ser especificados como os de *Encadeamento, Aninhamento, Contador e Looping Definido, Indefinido e Infinito.* Os itens 3 e 5 são formas de compor rotas mais complexas com maiores possibilidades de trajetos computacionais e por isso, não possuem elementos linguísticos a serem vinculados.

6.1.4. Planejar a espiral TFR

Com o mapa linguístico definido contendo os principais itens da linguagem que servirão para a compreensão dos fundamentos da programação, o professor pode iniciar o planejamento da espiral TFR. O planejamento é o momento que o processo da aquisição da linguagem de programação é definido e por isso, haverá divergências quanto a sua sequência, o que será produtivo se houver um trabalho sério de observação, análise e comparação entre os diversos planejamentos propostos, considerando o nível de ensino, idade, familiaridade e outros critérios que podem fazer parte do estudo da espiral TFR.

A espiral pode ser planejada de forma micro e macro conforme apresentado a seguir, importa destacar que ambas são visões da mesma espiral, por isso devem conter a mesma quantidade de voltas e incrementação nos graus de complexidade. O planejamento da espiral deve ter um início e fim, já que ela está contida em um ambiente escolar que possui módulos e carga horária bem definida. Esses fatores impactam na espiral já que podem fazer com que alguns itens sejam mais ou menos trabalhados ou que exija um salto maior em complexidade a cada volta se a carga horária for menor. O planejamento abaixo considera uma disciplina que será dada no primeiro semestre de 2021.

Tabela 8 - Espiral TFR: Planejamento Macro

Volta	Espiral	Início	Fim	Horas
1	$E_{1,1,1}$	01/02/21	07/02/21	5h
2	$E_{1,2,1}$	08/02/21	14/02/21	5h
3	$E_{1,3,1}$	15/02/21	21/02/21	5h
4	$E_{2,3,1}$	22/02/21	28/02/21	5h
5	$E_{2,3,2}$	01/03/21	07/03/21	5h
6	$E_{2,3,3}$	08/03/21	14/03/21	5h
7	$E_{2,4,3}$	15/03/21	21/03/21	5h
8	$E_{2,5,3}$	22/03/21	28/03/21	5h
9	$E_{3,5,3}$	29/03/21	04/04/21	5h
10	$E_{4,5,3}$	05/04/21	11/04/21	5h
11	$E_{4,5,4}$	12/04/21	18/04/21	5h
12	$E_{4,5,5}$	19/04/21	25/04/21	5h
13	$E_{4,6,5}$	26/04/21	02/05/21	5h
14	$E_{4,7,5}$	03/05/21	09/05/21	5h

Fonte: O autor (2021)

O planejamento macro é uma visão simplificada onde observamos a quantidade de voltas, os saltos de complexidade e o tempo de cada volta. Dependendo dos objetivos da disciplina, a espiral não precisa alcançar o grau máximo da TFR. Os itens 6 e 7 do item "Função" da TFR normalmente são mais difíceis de serem compreendidos por alunos novatos e podem não entrar na espiral se o professor não avaliar que seja produtivo.

A figura acima demonstra uma espiral completa, ou seja, que abrange todos os graus de complexidade, e que avança um grau de complexidade por item a cada volta. Cada volta na espiral tem duração de uma semana em que o novato será exposto ao conteúdo e passará por um processo de avaliação para avançar na espiral. Se necessário, o planejamento pode ser modificado para se adequar à turma, questões de conteúdo, avaliação e replanejamento serão vistos na próxima sessão em *in-espiral*.

Pode-se dizer que essa é uma espiral conservadora já que avança lentamente nos graus de complexidade. Outras abordagens poderiam ter menos voltas, avançando em mais de um item da TFR de uma vez ou até mesmo considerando mais de um grau de complexidade em um mesmo item da TFR. Por exemplo, o professor poderia apresentar a rota condicional simples e composta na mesma volta da espiral se assim desejasse.

O planejamento macro possibilita uma visão desvinculada dos elementos linguísticos da linguagem, de outro lado, o planejamento micro considera o conteúdo linguístico que será apresentado a cada volta.

Tabela 9 - Espiral TFR - Planejamento Micro

Volta	Espiral	Elementos Linguísticos
1	$E_{1,1,1}$	T_1 = { int, short, long, decimal, DateTime, string, char, bool } F_1 = { +, -, *, /, >, >=, <, <=, ==, != } R_1 = { }
2	$E_{1,2,1}$	T_1 = { ..., float, double, TimeSpan } F_1 = { ..., +=, -=, *=, /=, ++, -- } F_2 = { substring, indexOf, replace, pow, sqrt, round } R_1 = { }
3	$E_{1,3,1}$	T_1 = { ..., ushort, uint, ulong } F_1 = { ..., !, &&, || } F_2 = { ..., truncate, abs, addDays, addMonths, addYears } F_3 = { class } R_1 = { }
4	$E_{2,3,1}$	T_1 = { ... } T_2 = { class } F_1 = { ... } F_2 = { ..., Day, Month, Year, TotalDays, TotalHours, DayOfWeek } F_3 = { class } R_1 = { }

Fonte: O autor (2021)

Na figura acima é apresentado o planejamento micro da mesma espiral, porém mostramos apenas até a volta quatro para não dificultar o entendimento de sua organização. O planejamento completo pode ser encontrado no Anexo II.

Na volta um, inicia-se informando quais elementos da linguagem CSharp serão apresentados ao novato para o grau 1 em todos os itens da TFR ($E_{1,1,1}$). Utiliza-se a letra do item da TFR com o grau de complexidade da volta subscrito e foi atribuído um conjunto de elementos que serão apresentados. Para graus que não possuem elementos, será utilizado a notação de conjunto vazio.

É importante observar que no planejamento micro pode-se detectar divergência nos elementos linguísticos apresentados. Por exemplo, um professor poderia escolher apresentar apenas tipos numéricos na primeira volta enquanto outro, tipos numéricos e de texto. A diferença pode ser pequena, mas com um estudo detalhado, poder-se-ia apontar a importância das primeiras voltas da espiral para o aprendizado do novato,

de fato, os primeiros contatos podem motivar ou traumatizar alguns alunos, trazendo marcas difíceis de serem removidas devido à complexidade do aprendizado de programação, isso se comprova pela quantidade de alunos que desistem do curso no primeiro módulo letivo.

Na segunda volta, a complexidade do item "Função" foi aumentada em 1, enquanto os outros itens permaneceram iguais ($E_{1,2,1}$). Porém, outros itens foram adicionados ao grau 1 do item "Tipo", pode-se observar essa ação pelos caracteres (...) os quais significam "os itens anteriores". Optou-se por não listar todos os itens pertencentes ao grau para explicitar o que foi incrementando na volta corrente ou se os elementos foram mantidos. Para cada grau do item "Função" acrescentou-se um novo conjunto (F_1 e F_2), dessa forma, os elementos não são descaracterizados de seu grau de complexidade.

Avançando para a volta 3, ele corre adicionando um grau de complexidade em "Função" ($E_{1,3,1}$) e adicionando novos elementos em T_1 e F_1 e F_2. A volta 4 adiciona complexidade em "Tipo" ($E_{2,3,1}$) e não possui novos itens em T_1. Esse processo ocorre até a última volta da espiral.

O estudo do planejamento macro e micro da espiral pode avançar conforme novas pesquisas experimentais aconteçam. O vínculo dos conceitos no planejamento micro pode ser um aspecto a ser trabalhado futuramente.

6.2. In-Espiral

As ações *in-espiral* correspondem ao momento que ocorre durante o processo de aprendizagem e estão relacionadas à apresentação e avaliação do conteúdo, bem como técnicas didáticas e motivacionais, no entanto, esse guia não pretende apontar essas técnicas, mas deter-se em um processo essencial da *in-espiral*, o replanejamento da espiral TFR, a qual necessita regularmente - durante o processo de aprendizagem - ser reajustada conforme as percepções do professor em relação àquela turma.

As duas ações que definem a *in-espiral* são: (1) Relembrar, apresentar e avaliar conteúdo, e (2) replanejar espiral.

6.2.1. Relembrar, apresentar e avaliar conteúdo

Como dito anteriormente, esse guia não apontará técnicas ou ferramentas para apresentação e avaliação de conteúdo, mas antes quer colocá-los em um lugar

privilegiado na espiral TFR. O professor deve escolhê-los conforme sua própria estratégia didática. No entanto, a espiral TFR está fundada nas ideias da aprendizagem significativa conforme mencionado nos capítulos anteriores, dessa forma, "relembrar" ocupa um papel especial em cada volta da espiral, onde o novato é estimulado a lembrar os principais conceitos já percorridos nas voltas anteriores e vinculá-los aos novos conceitos, ampliando-os e aprofundando-os. Relembrar é a primeira ação ao iniciar uma nova volta na espiral.

A ancoragem dos conhecimentos anteriores com os atuais, portanto, tem grande importância para a espiral TFR. A apresentação dos novos conteúdos devem ampliar os conhecimentos anteriores, ou seja, um conceito pode ter sido bem compreendido pelo novato e pode estar sendo bem aplicado em exercícios práticos, mas esse mesmo conceito pode trazer dificuldades quando combinado com outro, o que significa que o novato dominará um certo conceito quando puder combiná-lo com outros diversos em situações variadas. Essa habilidade determinará a completude de um conceito, já que se aplicado isoladamente, tornar-se-ia difícil compreender se o aluno está apto a avançar na espiral.

Avaliar o novato expondo-o a situações variadas que exijam combinar e variar os conceitos possibilita identificar se ele está apto a avançar na espiral. O método de avaliação também deve ser feito a cada volta e não precisa necessariamente ser uma prova ou trabalho, às vezes exercícios práticos ou dinâmicas podem expor o novato a problemas que exigirão raciocínio mais rigoroso para chegar em uma solução. Avaliar é a última ação antes de encerrar uma volta na espiral.

6.2.2. Replanejar espiral TFR

Cada turma pode avançar mais ou menos rápido na espiral, a avaliação e percepção do professor quanto ao ritmo de aprendizado da turma possibilitará que ele replaneje as próximas voltas da espiral de forma a adaptá-la à turma. Às vezes replanejar pode apenas significar estender o tempo da volta da espiral ao invés de modificar o conteúdo e sua sequência.

Com experiência na espiral TFR, o professor distinguirá os itens mais e menos complexos, podendo acelerar ou frear o processo de aquisição da linguagem de programação. Dessa forma, o professor tendo o mapa linguístico vinculado aos fundamentos da programação e suas complexidades e propondo cada volta através

da observação e análise, contribui para o amadurecimento da espiral TFR e consequentemente com o processo de aquisição da linguagem de programação. Os índices e parâmetros organizados na estrutura TFR possibilitam o diálogo intermetodológico entre os professores e produzirão trabalhos que buscam entender como um novato adquire uma linguagem de programação.

7 CONCLUSÃO

Este trabalho buscou entender o processo de aquisição de uma linguagem de programação dentro do paradigma imperativo com a finalidade de formar profissionais mais capacitados a atuarem nas oportunidades de trabalho oferecidas pelo mercado. Portanto, é um trabalho que envolve saberes da educação e computação, especificamente em linguagem de programação.

Através desse prisma, foi necessário levantar as diretrizes que regem as leis educacionais ditadas pelo MEC para entender os objetivos em que os níveis de ensino técnico e superior colocam seus esforços. Foram apontados estudos sobre a aprendizagem de programação com o fim de examinar os mecanismos que facilitam e dificultam o aprendizado do novato na área. Em seguida, buscou-se entender, nas teorias mais gerais de aprendizado, como se dá a dinâmica aprender/ensinar e como os conceitos são adquiridos pelo aprendiz.

Os objetivos gerais e específicos, os quais focam na elaboração de um modelo teórico que organize e classifique os fundamentos da programação e permita o professor realizar o planejamento do processo de aquisição da linguagem de programação, foram atendidos com a criação da estrutura TFR e sua abordagem em espiral.

A TFR explicita os conceitos chaves que um programador deve aplicar no momento da codificação, ou seja, ele deve estar apto a trabalhar com Tipos, Funções e Rotas. Ela classifica os fundamentos da programação conforme o grau de complexidade baseando-se nas ideias de composição e abstração.

O processo de aprendizagem diz respeito à forma que o professor planeja o ensino desses conceitos. Para isso, foi proposto como modelo de planejamento a aplicação em espiral da TFR, onde é possível revisar e aprofundar os conceitos a cada nova volta da espiral, permitindo partir do simples para os mais complexos.

O planejamento da espiral permite o diálogo intermetodológico e possibilita novos estudos sobre a aquisição da linguagem de programação, permitindo que outros aspectos sejam discutidos como didática, ferramentas e técnicas.

Assim, o estudo da espiral TFR contribui com a compreensão do processo de aquisição de uma linguagem de programação, mas ainda não resolve todos os aspectos da aquisição, necessitando trabalhos posteriores que utilizem o guia da espiral como base para estudos sobre as dicotomias ensinar/aprender, teoria/prática,

memorização/compreensão, conceito/aplicação, linguagem/lógica e outras que envolvem a aprendizagem de programação.

8 CONSIDERAÇÕES FINAIS

Conheci poucas pessoas que fizeram uma pesquisa acadêmica de nível superior, encontrava-me no grupo de pessoas que pouco sabiam sobre o que é um mestrado ou doutorado, ainda acrescento alguns preconceitos sobre a importância da academia muitas vezes ignorada pelo mercado de trabalho. Entrei para o curso de mestrado após 10 anos de experiência na área de TI, mas entrei como aprendiz, ou melhor, como novato, coloquei-me à disposição para beber do conhecimento incomparável de meus professores, já que não consegui tal admiração em minha trajetória na graduação.

O mestrado me proporcionou nova visão do mundo acadêmico. Mesmo em meus primeiros passos, sinto que o espírito da pesquisa tocou-me os ideais. Ainda há muito trabalho a fazer, muito a pesquisar sobre o ensino técnico de programação e desejo contribuir com futuros trabalhos, buscando aumentar a qualidade do ensino e a abrir novas portas de esperança, principalmente aos jovens talentosos menos afortunados que aguardam uma oportunidade para alterarem o rumo de sua jornada profissional.

Iniciei minha pesquisa com um tema que buscava criar um método para ser autodidata em programação, logo me vi impossibilitado em continuar com a ideia de criar um método, já que me faltavam experiências e aprofundamento teórico em certas áreas da computação. Coloquei-me disposto a preencher as lacunas teóricas e diminuí minha ambição para criar uma técnica ao invés de método. Porém, observei que levou anos para que meu orientador, que detinha maior propriedade acadêmica que eu, criasse uma técnica hoje em ascensão contínua chamada OC2RD2. Nessa época, estava cursando a disciplina Fundamentos das Ciências Cognitivas e estonteado com as ideias da construção do conhecimento, linguagem e semiótica, alterei meu tema para o ambiente de aprendizagem de programação. Busquei correlacionar os estudos sobre linguagem natural e linguagem de programação, essa etapa da pesquisa durou tempo considerável e me fez perceber certas analogias sobre ambas categorias de linguagem e a adquirir profundidade nas teorias gerais da aprendizagem, todavia, novamente, alertado por meu orientador e comprovado pela experiência própria, percebi que pouco poderia utilizar da linguagem natural que contribuísse efetivamente com os desafios do ensino de programação.

Aproveitei o tempo de isolamento causado pela pandemia do vírus COVID-19 para colocar-me em reflexão sobre o tema final, após muitas tardes caminhando pelos corredores de meu apartamento consegui chegar ao tema final, dessa vez aprovado por meu orientador e finalmente com um sentimento de que o caminho estava traçado. Os próximos meses levaram à conclusão do trabalho. Senti-me fazendo parte do processo de pesquisa, já que me permiti errar e passar pelos tortuosos momentos da busca pelo tema, do levantamento bibliográfico do estado da arte, do fechamento categorizado das citações, da análise crítica das obras voltando os olhares ao tema definido, do levantamento dos objetivos e hipóteses, da escrita dos primeiros parágrafos, da fundamentação teórica e de tantos outros momentos que hoje me fizeram lançar novos olhares sobre o que é ser um pesquisador.

REFERÊNCIAS

ABELSON, H.; SUSSMAN, G. J.; SUSSMAN, J. **Structure and Interpretation of Computer Programs**. 2.ed. Cambridge: MIT Press, 1996.

ANNAMALAI, Subashini; SALAM, S. N. A. Facilitating programming comprehension for novice learners with multimedia approach: A preliminary investigation. **AIP Conference Proceedings**, out. 2017. Disponível em: <https://aip.scitation.org/doi/10.1063/1.5005362>. Acesso em: 20 ago. 2020.

ARAUJO, L. G. J. Uma Abordagem em Espiral para Disciplinas Iniciais de Programação na Educação Profissional em Informática. 2018. 266f. **Dissertação (Mestrado em Computação Aplicada)** - Universidade Estadual de Feira de Santana, Feira de Santana, 2018. Disponível em: <http://tede2.uefs.br:8080/handle/tede/696>. Acesso em: 22 ago. 2020.

ARAUJO, L. G. J.; BITTENCOURT, R. A.; SANTOS, D. M. A Contextualized Spiral Approach for Teaching Programming in IT Vocational Secondary Education. **JCThink**, [s.l], v. 2, n. 1, 2018. Disponível em: <https://siaiap32.univali.br/seer/index.php/IJCThink/article/view/12421/0>. Acesso em: 2 fev. 2020.

AUSUBEL, D. P. 1963. **The psychology of meaningful verbal learning**.

AUSUBEL, D. P., Novak, J. D., & Hanesian, H. **Educational psychology: A cognitive view**. v. 6, New York: Holt, Rinehart and Winston, 1968.

BERSSANETTE, J. H. Ensino de programação de computadores: uma proposta de abordagem prática baseada em Ausubel. 2016. 145 f. **Dissertação (Mestrado em Ensino de Ciência e Tecnologia)** - Universidade Tecnológica Federal do Paraná, Ponta Grossa, 2016. Disponível em: <http://repositorio.utfpr.edu.br/jspui/handle/1/2487>. Acesso em: 22 ago. 2020.

TI precisa de 420 mil novos profissionais até 2024. **BRASSCOM**. 02 mai. 2019. Disponível em: <https://brasscom.org.br/ti-precisa-de-420-mil-novos-profissionais-ate-2024>. Acesso em 10 nov. 2020.

Estudo aponta falta de mão de obra qualificada em tecnologia. **BRASSCOM**. 28 fev. 2020. Disponível em: <https://brasscom.org.br/estudo-aponta-falta-de-mao-de-obra-qualificada-em-tecnologia>. Acesso em 10 nov. 2020.

BRASIL. **Lei de Diretrizes e Bases da Educação Nacional**, LDB. 9394/1996.

BREZOLIN, L. M. T. F. Uma proposta para aplicação de mapas conceituais ao processo de ensino-aprendizagem e computação. 2010. 138 f. **Dissertação (Mestrado em Tecnologias da inteligência e design digital)** - Pontifícia Universidade Católica de São Paulo, São Paulo, 2010. Disponível em: <https://tede2.pucsp.br/handle/handle/18256>. Acesso em: 22 ago. 2020.

BRUNER, J. S. 1966. **Toward a theory of instruction**.

BRUSILOVSKY, P. et al. Teaching Programming to Novices: A Review of Approaches and Tools. **Proceedings of ED-MEDIA 94--World Conference on Educational Multimedia and Hypermedia**, Vancouver, 1994. Disponível em: <https://eric.ed.gov/?id=ED388228>. Acesso em: 2020 ago. 2020.

FELLEISEN, M. **How to Design Programs: An Introduction to Computing and Programming**. Cambridge: MIT Press, 2001.

FERNANDES, V. S.; FREITAS JUNIOR, V. Evasão e reprovação: uma análise das metodologias de ensino para a disciplina de lógica e programação. **IX MICTI**, Santa Catarina, ago. 2014. Disponível em: <http://eventos.ifc.edu.br/micti/wp-content/uploads/sites/5/2014/08/Metodologias-para-o-ensino-de-programa%C3%A7%C3%A3o.pdf>. Acesso em: 20 ago. 2020.

FONTES, C. R.; SILVA, F. W. O. DA. O ensino da disciplina linguagem de programação em escolas técnicas. **Ciências & Cognição**, v. 13, n. 2, p. 84 98, 2008. Disponível em: <http://www.cienciasecognicao.org/revista/index.php/cec/article/view/221>. Acesso em: 22 ago. 2020.

KAPLAN, R. M. Teaching Novice Programmers Programming Wisdom. **In Proceedings of the 22nd Annual Psychology of Programming Interest Group**, Kutztown, 2010. Disponível em: <https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.222.6209&rep=rep1&type=pdf>. Acesso em: 20 ago. 2020.

KELLER, R. M. Computer Science: Abstraction to Implementation. **Harvey Mudd College**. set. 2001. Disponível em: <https://www.cs.hmc.edu/~keller/cs60book/%20%20%20All.pdf>. Acesso em 20 ago. 2020.

KUNKLE, W. M. The Impact of Different Teaching Approaches and Languages on Student Learning of Introductory Programming Concepts. 2010. 175 f. **Dissertação (Doutorado em Filosofia)** - Drexel University, Drexel, 2010. Disponível em: <https://core.ac.uk/download/pdf/190335704.pdf>. Acesso em: 22 ago. 2020.

KRUG, D. L. Método e ferramental para mapeamento da evolução de programadores durante o desenvolvimento de programas. 2018. 126f. **Dissertação (Mestrado em Computação Aplicada)** - Universidade Tecnológica Federal do Paraná, Curitiba, 2018. Disponível em: <http://repositorio.utfpr.edu.br/jspui/handle/1/3470>. Acesso em: 22 ago. 2020.

LEE, K. D. **Foundations of Programming Languages**. 2.ed. Nova York: Springer, 2017.

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: COBENGE XXXIII Congresso brasileiro de educação de engenharia**, Campina Grande, set. 2005. Disponível em:

<http://www.abenge.org.br/cobenge/arquivos/14/artigos/SP-5-05258920879-1117953533096.pdf>. Acesso em: 20 ago. 2020.

LIMA, M.; LEAL, M. C. Motivação discente no ensino-aprendizagem de programação de computadores. **Educação & Tecnologia**, Belo Horizonte, v. 17, n. 1, p. 94-110, jun. 2013. Disponível em: <https://seer.dppg.cefetmg.br/index.php/revista-et/article/view/447>. Acesso em: 22 ago. 2020.

LIMA, V. V. Espiral construtivista: uma metodologia ativa de ensino-aprendizagem. **Interface (Botucatu)**, Botucatu, v. 21, n. 61, p. 421-434, jun. 2017. Disponível em: <http://www.scielo.br/scielo.php?script=sci_arttext&pid=S1414-32832017000200421&lng=en&nrm=iso>. Acesso em: 27 ago. 2020.

MATTHÍASDÓTTIR, Á. How to teach programming languages to novice students? Lecturing or not? **International Conference on Computer Systems and Technologies**, [s.l.]. jan. 2006. Disponível em: <https://www.researchgate.net/publication/251812193_How_to_teach_programming_languages_to_novice_students_Lecturing_or_not>. Acesso em: 20 ago. 2020.

MEC. **Catálogo Nacional de cursos técnicos**. jan. 2014. Disponível em: <http://portal.mec.gov.br/secretaria-de-regulacao-e-supervisao-da-educacao-superior-seres/30000-uncategorised/52031-catalogo-nacional-de-cursos-técnicos>. Acesso em: 12 set. 2019.

_____. **Diretrizes Curriculares Nacionais para a Educação Profissional Técnica de Nível Médio**. set. 2012. Disponível em: <http://portal.mec.gov.br/cursos-da-ept/cursos-da-educacao-profissional-tecnica-de-nivel-medio>. Acesso em: 24 jul. 2019.

_____. **Educação profissional técnica de nível médio integrada ao ensino médio**. dez. 2007. Disponível em: <http://portal.mec.gov.br/setec/arquivos/pdf/documento_base.pdf>. Acesso em: 12 mar. 2020.

NAGANO, L.; DIRENE, A. I. Ensino de Lógica de Programação baseado na indução-dedução através de exemplos. **Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação - SBIE)**, [s.l.], p. 1352, nov. 2016. Disponível em: <https://br-ie.org/pub/index.php/sbie/article/view/6827>. Acesso em: 22 ago. 2020.

PIMENTEL, E. P. et al. Avaliação Contínua da Aprendizagem, das Competências e Habilidades em Programação de Computadores. **Anais do Workshop de Informática na Escola**, [s.l.], p. 533-544, jan. 2003. Disponível em: <https://br-ie.org/pub/index.php/wie/article/view/819>. Acesso em: 22 ago. 2020.

RAMOS, M. V. M.; NETO, J. J.; VEGA, I. S. **Linguagens Formais: Teoria, Modelagem e Implementação**. Porto Alegre: Bookman, 2009.

ROCHA, H. R. Programação com o scratch para aprender ciências uma proposta para a formação de professores da educação básica. 2017. 72 f. **Dissertação (Mestrado em Ensino Tecnológico)** - Instituto Federal Amazonas, Manaus, 2017. Disponível em: <http://repositorio.ifam.edu.br/jspui/handle/4321/284>. Acesso em: 22 ago. 2020.

SALLEH, S. M.; SHUKUR, Z.; JUDI, H. M. Analysis of Research in Programming Teaching Tools: An Initial Review. **Procedia - Social and Behavioral Sciences**, v. 103, p. 127-135, nov. 2013. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1877042813037622>. Acesso em: 20 ago. 2020.

SANTANA, B.; ARAÚJO, L. G.; BITTENCOURT, R. Considerando a Motivação dos Estudantes em Experiências de Ensino-Aprendizagem de Computação. **Anais dos Workshops do Congresso Brasileiro de Informática na Educação**, [s.l.], p. 500, out. 2018. Disponível em: <https://br-ie.org/pub/index.php/wcbie/article/view/8275>. Acesso em: 23 ago. 2020.

SANTOS, A.; GORGÔNIO, A.; LUCENA, A.; GORGÔNIO, F. A Importância do Fator Motivacional no Processo Ensino-Aprendizagem de Algoritmos e Lógica de Programação para Alunos Repetentes. **In: WORKSHOP SOBRE EDUCAÇÃO EM COMPUTAÇÃO (WEI),** 2015, Recife. Anais do XXIII Workshop sobre Educação em Computação. Porto Alegre: Sociedade Brasileira de Computação, jul. 2015. p. 168-177. Disponível em: <https://sol.sbc.org.br/index.php/wei/article/view/10233>. Acesso em: 22 ago. 2020.

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**, [s.l.], v. 5, n. 1, p. 41-50, mar. 2006. Disponível em: <http://infocomp.dcc.ufla.br/index.php/infocomp/article/view/121>. Acesso em: 20 ago. 2020.

Índice de evasão de alunos é maior na área de tecnologia da informação. **SEMESP**, 7 out. 2013. Disponível em: <https://www.semesp.org.br/imprensa/indice-de-evasao-de-alunos-e-maior-na-area-de-tecnologia-da-informacao-2>. Acesso em: 10 jan. 2020.

SILVA, B. S; TRENTIN, M. A. S. Dificuldades no ensino-aprendizagem de programação de computadores: contribuições para a sua compreensão e resolução. **V Simpósio nacional de ensino de ciência e tecnologia**, nov. 2016. Disponível em: <http://www.sinect.com.br/2016/down.php?id=3349&q=1>. Acesso em: 20 ago. 2020.

SILVA, I. F. A.; SILVA, I. M. M.; SANTOS, M. S. **Análise de problemas e soluções aplicadas ao ensino de disciplinas introdutórias de programação**. Disponível em: <http://www.eventosufrpe.com.br/jepex2009/cd/resumos/R1479-1.pdf>. Acesso em: 20 ago. 2020.

SILVA, T. R. et al. Ensino-aprendizagem de programação: uma revisão sistemática da literatura. **Revista Brasileira de Informática na Educação**, [s.l.], v. 23, n. 01, p. 182, abr. 2015. Disponível em: <https://www.br-ie.org/pub/index.php/rbie/article/view/2838>. Acesso em: 26 ago. 2020.

VALENTE, J. A. A espiral da espiral de aprendizagem: o processo de compreensão do papel das tecnologias de informação e comunicação na educação. 2005. 238 p. **Tese (livre-docência)** - Universidade Estadual de Campinas, Instituto de Artes, Campinas, SP. Disponível em: <http://www.repositorio.unicamp.br/handle/REPOSIP/284458>. Acesso em: 20 ago. 2018.

VENTURA, L. et al. Perspectivas de ensino e aprendizagem: lógica da programação, piaget e tecnologias digitais. **In: PEREIRA, E. A pesquisa em psicologia em foco**. Belo Horizonte: Atena, 2019. cap. 8. p. 87-98.

VIGOTSKI, L. S. **A formação social da mente: O desenvolvimento dos processos psicológicos superiores**. Tradução de José Cipolla Neto. ed. 7. São Paulo: Martins Fontes, 2007 (1978).

_____. **Pensamento e linguagem**. Tradução de Jefferson Luiz Camargo. ed. 4. São Paulo: Martins Fontes, 2008 (1954).

VIGOTSKI, L. S; LURIA, A. R; LEONTIEV, A. N. **Linguagem, desenvolvimento e aprendizagem**. Tradução de Maria da Pena Villalobos. ed. 16. São Paulo: Ícone, 2010.

XIE, B. et al. A theory of instruction for introductory programming skills. **Computer Science Education**, v. 29, n. 2-3, p. 205-253, 2019. Disponivel em: <https://doi.org/10.1080/08993408.2019.1565235>. Acesso em: 01 set. 2020.

WATT, D. A. **Programming language design concepts**. Hoboken: John Wiley, 2004.

ANEXO I – COMPETÊNCIAS E BASES TECNOLÓGICAS: FUND. DA LÓGICA

ESCOLA PROFISSIONAL NOSSA SENHORA DE FÁTIMA
Av. Cel. Octaviano de Freitas Costa, 463
04773-000 Veleiros – São Paulo – SP
Fone: 5687-8876

4. Fundamentos da Lógica

Competências

- Desenvolver algoritmos através de divisão modular e refinamento sucessivos.
- Distinguir e avaliar linguagens e ambiente de programação, aplicando no desenvolvimento de software.
- Interpretar pseudocódigos, algoritmos e outras especificações para codificar programas.
- Avaliar resultados de testes dos programas desenvolvidos.
- Integrar módulos desenvolvidos separadamente.
- Compreender o paradigma de orientação por objeto e sua aplicação em programação.

Habilidades

- Selecionar e utilizar estrutura de dados na resolução de problemas computacionais.
- Utilizar editores de textos, planilhas, gerenciadores de bases de dados, compiladores e ambientes de desenvolvimento na elaboração de programas.
- Utilizar modelos, pseudocódigos e ferramentas na representação da solução de problemas.
- Elaborar e executar casos e procedimentos de testes de programas.
- Redigir instruções de uso dos programas implementados.
- Aplicar as técnicas de programação (orientada a objeto, estruturadas e outras).

Bases Tecnológicas

- Lógica computacional.
- Algoritmos e pseudocódigos.
- Técnicas de programação (estruturada, orientada a objetos e outras.
- Linguagem de programação.
- Estruturas de dados.
- Ambientes de desenvolvimento de programas.
- Ferramentas CASE
- Prototipação de sistemas.

ANEXO II – PLANEJAMENTO MICRO ESPIRAL TFR

Volta	Espiral	Elementos Linguísticos
1	$E_{1,1,1}$	T_1 = { int, short, long, decimal, DateTime, string, char, bool } F_1 = { +, -, *, /, >, >=, <, <=, ==, != } R_1 = { }
2	$E_{1,2,1}$	T_1 = { ..., float, double, TimeSpan } F_1 = { ..., +=, -=, *=, /=, ++, -- } F_2 = { substring, indexOf, replace, pow, sqrt, round } R_1 = { }
3	$E_{1,3,1}$	T_1 = { ..., ushort, uint, ulong } F_1 = { ..., !, &&, || } F_2 = { ..., truncate, abs, addDays, addMonths, addYears } F_3 = { class } R_1 = { }
4	$E_{2,3,1}$	T_1 = { ... } T_2 = { class } F_1 = { ... } F_2 = { ..., Day, Month, Year, TotalDays, TotalHours, DayOfWeek } F_3 = { class } R_1 = { }
5	$E_{2,3,2}$	T_1 = { ..., ?: } T_2 = { ... } F_1 = { ... } F_2 = { ... } F_3 = { ... } R_1 = { } R_2 = { if, else, else if }

6	$E_{2,3,3}$	T_1 = { ... } T_2 = { ... } F_1 = { ... } F_2 = { ... } F_3 = { ... } R_1 = { } R_2 = { ... } R_3 = { }
7	$E_{2,4,3}$	T_1 = { ... } T_2 = { ... } F_1 = { ... } F_2 = { ... } F_3 = { ... } F_4 = { } R_1 = { } R_2 = { ... } R_3 = { }
8	$E_{2,5,3}$	T_1 = { ... } T_2 = { ... } F_1 = { ... } F_2 = { ... } F_3 = { ... } F_4 = { } F_5 = { } R_1 = { } R_2 = { ... } R_3 = { }
9	$E_{3,5,3}$	T_1 = { ... } T_2 = { ... } T_3 = { } F_1 = { ... } F_2 = { ... } F_3 = { ... } F_4 = { } F_5 = { } R_1 = { } R_2 = { ... } R_3 = { }

10	**$E_{4,5,3}$**	**T_1** = { ... } **T_2** = { ... } **T_3** = { } **T_4** = { Array, List } **F_1** = { ... } **F_2** = { ... } **F_3** = { ... } **F_4** = { } **F_5** = { } **R_1** = { } **R_2** = { ... } **R_3** = { }
11	**$E_{4,5,4}$**	**T_1** = { ... } **T_2** = { ... } **T_3** = { } **T_4** = { ..., Queue, Stack } **F_1** = { ... } **F_2** = { ... } **F_3** = { ... } **F_4** = { } **F_5** = { } **R_1** = { } **R_2** = { ... } **R_3** = { } **R_4** = { for, foreach }
12	**$E_{4,5,5}$**	**T_1** = { ... } **T_2** = { ... } **T_3** = { } **T_4** = { ..., Dictionary, Hashtable } **F_1** = { ... } **F_2** = { ... } **F_3** = { ... } **F_4** = { } **F_5** = { } **R_1** = { } **R_2** = { ... } **R_3** = { } **R_4** = { ..., while, do while } **R_5** = { }

13	**$E_{4,6,5}$**	**T_1** = { ... } **T_2** = { ... } **T_3** = { } **T_4** = { ..., HashSet } **F_1** = { ... } **F_2** = { ... } **F_3** = { ... } **F_4** = { } **F_5** = { } **F_6** = { } **R_1** = { } **R_2** = { ... } **R_3** = { } **R_4** = { ... } **R_5** = { }
14	**$E_{4,7,5}$**	**T_1** = { ... } **T_2** = { ... } **T_3** = { } **T_4** = { ... } **F_1** = { ... } **F_2** = { ... } **F_3** = { ... } **F_4** = { } **F_5** = { } **F_6** = { } **F_7** = { delegate, Func, Action } **R_1** = { } **R_2** = { ... } **R_3** = { } **R_4** = { ... } **R_5** = { }