UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
CIÊNCIA DA COMPUTAÇÃO

João Pedro Schmidt Cordeiro

Desenvolvimento de um Arcabouço Didático-Metodológico para o Ensino de Computação Distribuída

Florianópolis
2026

João Pedro Schmidt Cordeiro

Desenvolvimento de um Arcabouço Didático-Metodológico para o Ensino de Computação Distribuída

Trabalho de Conclusão de Curso submetido ao Curso de Graduação em Ciência da Computação do Centro Tecnológico da Universidade Federal de Santa Catarina como requisito para obtenção do título de Bacharel em Ciência da Computação.
Orientador: Prof. Odorico Machado Mendizabal, Dr.

Florianópolis
2026

Ficha catalográfica gerada por meio de sistema automatizado gerenciado pela BU/UFSC.
Dados inseridos pelo próprio autor.

Cordeiro, João Pedro Schmidt
Desenvolvimento de um arcabouço didático-metodológico para o ensino de computação distribuída / João Pedro Schmidt Cordeiro ; orientador, Odorico Machado Mendizabal, 2026.
107 p.

Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Santa Catarina, Centro Tecnológico, Graduação em Ciências da Computação, Florianópolis, 2026.

Inclui referências.

1. Ciências da Computação. 2. Educação. 3. Computação Distribuída. I. Mendizabal, Odorico Machado. II. Universidade Federal de Santa Catarina. Graduação em Ciências da Computação. III. Título.

João Pedro Schmidt Cordeiro

Desenvolvimento de um Arcabouço Didático-Metodológico para o Ensino de Computação Distribuída

Este Trabalho de Conclusão de Curso foi julgado adequado para obtenção do Título de Bacharel em Ciência da Computação e aprovado em sua forma final pelo curso de Graduação em Ciência da Computação.

Florianópolis, 15 Junho de 2026.

Profa. Lúcia Helena Martins Pacheco, Dra.
Coordenadora do Curso

Banca Examinadora:

Prof. Odorico Machado Mendizabal, Dr.
Orientador
Universidade Federal de Santa Catarina

Prof. Cristian Koliver, Dr.
Avaliador
Universidade Federal de Santa Catarina

Prof. Jean Carlo Rossa Hauck, Dr.
Avaliador
Universidade Federal de Santa Catarina

Ao meu eu do passado, apaixonado por aprender, ensinar e por computadores, e ao meu eu do futuro que anseia passar essa herança para a humanidade.

Aos meus pais.

AGRADECIMENTOS

Agradeço ao Prof. Odorico Machado Mendizabal, Dr., pela orientação cuidadosa e pelo espaço aberto para experimentação metodológica ao longo deste trabalho. Aos membros da banca, Prof. Cristian Koliver, Dr., e Prof. Jean Carlo Rossa Hauck, Dr., pelas contribuições à versão final deste texto. Aos estudantes da disciplina de Computação Distribuída do semestre 2026.1 que participaram da aplicação piloto das atividades aqui propostas. À Universidade Federal de Santa Catarina e ao Departamento de Informática e Estatística pela formação ofertada.

Agradeço a minha família por estar sempre muito presente e envolvida em minha vida, vocês são um pilar fundamental para o meu processo de evolução. À minha mãe Sandra Mara Lentz Schmidt, que sempre me incentiva a fazer tudo com excelência. Ao meu pai Pedro Cordeiro, que me estimula a pensar e fazer coisas diferentes, "sair da caixinha", inovar. À minha noiva Carolina Couto, que, com sua sensibilidade, é um estímulo permanente para que eu me torne um ser humano melhor. Ao meu vô Loso, que me ensinou as bases. À minha vó Bete, que me fez rir nos momentos difíceis. À minha vó Ruby, que mesmo um pouco mais distante sempre está perto trazendo o amor pela família. Ao meu vô Hamilton, ao qual com certeza aprendi muito pelo exemplo, mesmo não tendo a oportunidade de conhecê-lo fisicamente.

Ao meu querido amigo Victor Augusto Pereira Eyng, que constitui um exemplo de amizade em minha vida. Ao meu querido amigo Enzo Nicolás Spotorno Bieger, pela ajuda técnica e pela conduta exemplar como pesquisador, incentivando os que estão ao seu redor a sempre fazer um trabalho melhor. Ao amigo e colega de trabalho Thiago Crespo Felippi, que me ensina muito sobre a área de DevOps, estímulo inicial da busca por desenvolver o tema dessa pesquisa.

“Conseguir que as gerações futuras sejam mais felizes que a nossa, será o maior prêmio a que se possa aspirar. Não haverá valor comparável ao cumprimento dessa grande missão, que consiste em preparar para a humanidade futura um mundo melhor.”
Pecotche (1951)

RESUMO

Este trabalho de conclusão de curso investiga desafios do ensino de Computação Distribuída na graduação e propõe um arcabouço didático-metodológico que articula três dimensões complementares: metodologias de ensino, estratégias pedagógicas e tecnologias educacionais. A hipótese central é que a combinação intencional dessas três dimensões pode reduzir as barreiras de heterogeneidade de ambientes, carga cognitiva de configuração e dificuldade conceitual de tópicos como assincronia, sincronização, consistência e tolerância a falhas, promovendo aprendizagem mais efetiva em sistemas distribuídos.

O arcabouço é instrumentado por uma matriz conceitual que correlaciona tópicos, metodologias (Sala de Aula Invertida e Aprendizagem Baseada em Investigação), estratégias (Andaimagem em três níveis e Programação em Pares opcional) e tecnologias (contêineres Docker, máquinas virtuais Multipass/KVM e *scripts* de automação). Quatro atividades práticas materializam essa matriz: Cliente/Servidor Eco com *sockets* TCP, Espaço de Tuplas com Apache River, Sincronização de Relógios com NTP/chrony e Consenso com Raft acompanhado de *dashboard* ao vivo. Cada atividade é acompanhada de material de preparação para a fase pré-atividade, código reprodutível em repositório público e dois instrumentos de avaliação: questionário pós-atividade fundamentado em NASA-TLX, *Technology Acceptance Model* e itens de Aprendizagem Percebida, e relatório reflexivo entregue ao final de cada atividade.

A aplicação piloto ocorreu na disciplina de Computação Distribuída da UFSC no semestre 2026.1, em formato extracurricular, com sete respostas ao questionário e oito relatórios reflexivos coletados. O tamanho amostral restringe os resultados a uma leitura descritiva e qualitativa, sinalizando autoavaliação positiva de ganho conceitual e usabilidade na Atividade 1, e carga de trabalho baixa a moderada (Raw TLX médio de 23,6 em escala 0–100). A análise dos relatórios produziu, adicionalmente, um achado emergente que motivou refinamento iterativo do procedimento experimental da Atividade 2, ilustrando o ciclo de pesquisa baseada em design. As lições registradas alimentam frentes de continuidade que incluem escalonamento da aplicação em formato semestral pleno, ampliação das metodologias contempladas e disseminação do arcabouço para outras instituições.

Palavras-chave: Computação Distribuída. Metodologias Ativas. Sala de Aula Invertida. Aprendizagem Baseada em Investigação. Reprodutibilidade.

ABSTRACT

This Undergraduate Thesis investigates the challenges of teaching Distributed Computing in undergraduate courses and proposes a didactic-methodological framework that articulates three complementary dimensions: teaching methodologies, pedagogical strategies, and educational technologies. The central hypothesis is that the intentional combination of these three dimensions can reduce the barriers of environment heterogeneity, configuration cognitive load, and conceptual difficulty of topics such as asynchrony, synchronization, consistency, and fault tolerance, promoting more effective learning in distributed systems.

The framework is instrumented by a conceptual matrix that correlates topics, methodologies (Flipped Classroom and Inquiry-Based Learning), strategies (three-level Scaffolding and optional Pair Programming), and technologies (Docker containers, Multipass/KVM virtual machines, and automation scripts). Four practical activities materialize this matrix: Echo Client/Server with TCP sockets, Tuple Space with Apache River, Clock Synchronization with NTP/chrony, and Consensus with Raft accompanied by a live dashboard. Each activity is accompanied by pre-activity preparation material, reproducible code in a public repository, and two assessment instruments: a post-activity questionnaire grounded in NASA-TLX, the Technology Acceptance Model, and Perceived Learning items, and a reflective report submitted at the end of each activity.

The pilot application took place in the Distributed Computing course at UFSC in semester 2026.1, in extracurricular format, with seven questionnaire responses and eight reflective reports collected. The sample size restricts the results to a descriptive and qualitative reading, signaling positive self-assessment of conceptual gain and usability in Activity 1, and low-to-moderate workload (mean Raw TLX of 23.6 on a 0–100 scale). The analysis of the reports additionally produced an emergent finding that motivated iterative refinement of Activity 2's experimental procedure, illustrating the design-based research cycle. The lessons recorded feed continuation fronts that include scaling the application to a full semester format, expanding the methodologies covered, and disseminating the framework to other institutions.

Keywords: Distributed Computing. Active Methodologies. Flipped Classroom. Inquiry-Based Learning. Reproducibility.

LISTA DE FIGURAS

Figura 1 – Relação entre metodologias, estratégias de ensino e tecnologias educacionais. 27
Figura 2 – Os quatro instantes (T_1, T_2, T_3, T_4) do algoritmo de Cristian e fórmulas de *offset* (θ) e RTT (δ). 57
Figura 3 – Visualizador ao vivo do cluster Raft: três nós em layout circular (node3 como líder em verde, node1 e node2 como *follower* em cinza), painel de logs replicados e painel de eventos ao vivo. 61
Figura 4 – Distribuição das respostas ao Bloco 1 (Aprendizagem Percebida) da Atividade 1 ($n = 6$), por item e valor Likert. 71
Figura 5 – Perfil NASA-TLX por subescala (medianas) para a Atividade 1 ($n = 6$, escala 0–100). Valores maiores indicam maior carga; Desempenho aparece invertido conforme a escala original. 74
Figura 6 – Adesão por atividade: número de questionários e relatórios submetidos ao longo do semestre 2026.1. 80

LISTA DE TABELAS

Tabela 1 – Correlações entre tópicos, disciplina, metodologias, estratégias, tecnologias e atividades propostas. 53
Tabela 2 – Cobertura dos instrumentos de avaliação (questionário pós-atividade e relatório reflexivo) por atividade aplicada. 69
Tabela 3 – Respostas ao Bloco 1 (Aprendizagem Percebida) da Atividade 1 ($n = 6$). . . 70
Tabela 4 – Respostas ao Bloco 2 (Usabilidade do Ambiente) da Atividade 1 ($n = 6$). . . 72
Tabela 5 – Respostas brutas ao Bloco 3 (NASA-TLX) e Raw TLX (0–100) por respondente da Atividade 1 ($n = 6$). 72
Tabela 6 – Síntese das respostas ao Bloco 4 (Questões Abertas) da Atividade 1. 74
Tabela 7 – Equívocos conceituais e lacunas identificados nos relatórios reflexivos das Atividades 1 e 2. 78

LISTA DE SÍMBOLOS

α	Coeficiente alfa de Cronbach (consistência interna)
n	Tamanho amostral
N	Número de nós em um *cluster* Raft
$T_1..T_4$	Quatro instantes do cálculo de *offset* em NTP (algoritmo de Cristian)
f	Tolerância máxima a falhas em Raft, $f = \lfloor (N-1)/2 \rfloor$

SUMÁRIO

1 INTRODUÇÃO 23
1.1 ABORDAGEM METODOLÓGICA 24
1.2 ORGANIZAÇÃO DO DOCUMENTO 24

2 FUNDAMENTAÇÃO TEÓRICA 27
2.1 METODOLOGIAS DE ENSINO 28
2.1.1 Aprendizagem baseada em problemas 28
2.1.2 Aprendizagem baseada em projetos 28
2.1.3 Aprendizagem baseada em equipe 29
2.1.4 Sala de aula invertida 30
2.1.5 Instrução pelos Pares (Peer Instruction) 31
2.1.6 Aprendizagem Baseada em Investigação (Inquiry-Based Learning) . . . 32
2.1.7 Outras metodologias 33
2.2 ESTRATÉGIAS DE ENSINO 34
2.2.1 Andaime (*Scaffolding*) 34
2.2.2 Coding Dojo 35
2.2.3 Programação em Pares (*Pair Programming*) 37
2.2.4 Outras Estratégias 38
2.3 TECNOLOGIAS UTILIZÁVEIS PARA O ENSINO DE COMPUTAÇÃO PARALELA E DISTRIBUÍDA 39
2.3.1 Contêineres 39
2.3.2 Máquinas Virtuais 40
2.3.3 *Scripts* 42
2.3.4 Outras Tecnologias 43

3 TRABALHOS RELACIONADOS 45
3.1 ENSINO DE PROGRAMAÇÃO PARALELA PARA INICIANTES 45
3.2 APRENDIZAGEM BASEADA EM PROJETOS COM *SCAFFOLDING* . . 46
3.3 INFRAESTRUTURA E FERRAMENTAS DE APOIO AO ENSINO 47
3.4 ELABORAÇÃO DE EXERCÍCIOS PRÁTICOS PARA INICIANTES . . . 48
3.5 SÍNTESE E POSICIONAMENTO DESTE TRABALHO 49

4 PROPOSTAS DE ABORDAGENS PARA O ENSINO 51
4.1 MATRIZ CONCEITUAL DE ABORDAGENS PEDAGÓGICAS 51
4.2 PROPOSTAS DE ATIVIDADES PRÁTICAS 53
4.2.1 Estrutura comum às quatro atividades 54
4.2.2 Atividade 1: Cliente/Servidor Eco com Sockets TCP 54
4.2.3 Atividade 2: Espaço de Tuplas com Apache River 55

4.2.4 Atividade 3: Sincronização de Relógios com NTP/chrony 56
4.2.5 Atividade 4: Consenso com Raft e Visualizador ao Vivo 59
4.3 AMBIENTE PADRONIZADO: ARQUITETURA E PRINCÍPIOS 62

5 RELATO DE EXPERIÊNCIA NO AMBIENTE DA UFSC 65
5.1 OBJETIVO E CARACTERIZAÇÃO DO ESTUDO 65
5.2 FORMA DE AVALIAÇÃO 65
5.3 CONTEXTO DE APLICAÇÃO 67
5.4 RESPOSTAS DO QUESTIONÁRIO 69
5.4.1 Perfil e cobertura das respostas 69
5.4.2 Leitura descritiva por bloco 70
5.4.3 Indicações iniciais e limitações 75
5.5 ANÁLISE DOS RELATÓRIOS REFLEXIVOS 76
5.5.1 Cobertura e procedimento de análise 76
5.5.2 Equívocos conceituais e acertos relevantes 77
5.6 OBSERVAÇÕES DA APLICAÇÃO 79
5.6.1 Adesão e engajamento 79
5.6.2 Material didático e viabilidade metodológica 80
5.7 LIÇÕES PARA REFINAMENTO DAS ATIVIDADES 81
5.8 SÍNTESE E AMEAÇAS À VALIDADE 82

6 CONSIDERAÇÕES FINAIS 85
6.1 SÍNTESE DAS CONTRIBUIÇÕES POR OBJETIVO ESPECÍFICO 85
6.1.1 Objetivo 1 – Conduzir revisão exploratória da literatura. 85
6.1.2 Objetivo 2 – Propor uma matriz metodologias × estratégias × tópicos × atividades. 85
6.1.3 Objetivo 3 – Prototipar ambiente padronizado e conjunto de atividades práticas. 86
6.1.4 Objetivo 4 – Desenvolver instrumentos de avaliação. 86
6.1.5 Objetivo 5 – Aplicar em turmas reais, analisar e identificar refinamentos. 86
6.2 LIMITAÇÕES E LIÇÕES APRENDIDAS 87
6.3 TRABALHOS FUTUROS 87

REFERÊNCIAS 91

APÊNDICES 95

APÊNDICE A – ARTIGO CIENTÍFICO NO FORMATO SBC 97

1 INTRODUÇÃO

O avanço da computação e a ubiquidade de sistemas distribuídos exigem profissionais com base conceitual sólida e fluência prática. A literatura educacional (Arroyo, 2013) sugere distribuir os conteúdos de Paralelismo e Distribuição ao longo da graduação, apoiados por práticas que promovam a compreensão e a aplicação dos mesmos. Contudo, persistem barreiras relevantes: heterogeneidade de ambientes (hardware e sistemas operacionais), complexidade de tópicos como assincronia, sincronização, consistência e tolerância a falhas, além da alta carga cognitiva para configuração e depuração. Ellis Michael et al. (Ellis et al., 2019) destacam a dificuldade de iniciantes em produzir código correto em sistemas distribuídos; embora técnicas como verificação de modelos (*model checking*) e depuração avançada sejam úteis, costumam ser onerosas em contextos educacionais.

Diante dessa problemática, este trabalho propõe a estruturação de um arcabouço didático-metodológico organizado em três frentes complementares: (i) **metodologias de ensino**, que definem os princípios pedagógicos orientadores da construção do conhecimento, colocando o estudante como protagonista do processo de aprendizagem; (ii) **estratégias de ensino**, que correspondem às ações e recursos específicos utilizados pelo professor para operacionalizar as metodologias; e (iii) **tecnologias educacionais**, que funcionam como ferramentas viabilizadoras que apoiam e potencializam o processo de ensino-aprendizagem, assegurando reprodutibilidade e praticidade nos laboratórios. A hipótese central é que a combinação intencional dessas três dimensões pode reduzir as barreiras identificadas e promover uma aprendizagem mais efetiva em Computação Distribuída, permitindo que os estudantes concentrem-se nos conceitos fundamentais enquanto desenvolvem competências técnicas aplicáveis. O arcabouço proposto foi aplicado na disciplina de Computação Distribuída da Universidade Federal de Santa Catarina, com coleta de dados empíricos para validação.

O **objetivo geral** deste trabalho é definir, instrumentar e validar empiricamente um arcabouço didático-metodológico que integre metodologias de ensino, estratégias pedagógicas e tecnologias educacionais para alcançar resultados práticos no ensino de Computação Distribuída, aproximando os estudantes da aplicação prática dos conceitos teóricos.

Os **objetivos específicos** deste trabalho são:

- Conduzir uma revisão exploratória da literatura sobre metodologias de ensino, estratégias pedagógicas e tecnologias educacionais aplicáveis ao ensino de Computação Distribuída, fundamentando as escolhas que orientam o arcabouço proposto.

- Propor uma matriz de metodologias $\times$ estratégias $\times$ tópicos $\times$ atividades, com critérios de seleção e progressão.

- Prototipar um ambiente padronizado utilizando tecnologias educacionais (*scripts*, contêineres, máquinas virtuais) e um conjunto de atividades práticas com material de apoio.

- Desenvolver instrumentos de avaliação aplicáveis às atividades propostas, abrangendo questionário pós-atividade e relatório reflexivo.

- Aplicar parte das atividades em turmas reais da disciplina de Computação Distribuída da UFSC, coletar e analisar os dados resultantes e identificar pontos de refinamento.

1.1 ABORDAGEM METODOLÓGICA

Quanto ao delineamento metodológico, este trabalho caracteriza-se como pesquisa de natureza **aplicada**, com objetivos **exploratório-descritivos** e abordagem **predominantemente qualitativa**, complementada por dados quantitativos tratados de forma descritiva (Gil, 2002). O percurso combina dois procedimentos principais: a pesquisa bibliográfica, que fundamenta a concepção do arcabouço, e a aplicação piloto em contexto real de ensino, relatada na forma de relato de experiência. O processo de concepção, aplicação e refinamento alinha-se aos princípios da pesquisa baseada em design (*design-based research*), que preconiza ciclos sucessivos de projeto, intervenção em contexto autêntico de sala de aula, análise e ajuste (The Design-Based Research Collective, 2003).

O desenvolvimento do trabalho organizou-se em cinco grandes etapas, diretamente relacionadas aos objetivos específicos enunciados acima e aos capítulos deste documento: (i) **revisão exploratória da literatura** sobre metodologias de ensino, estratégias pedagógicas, tecnologias educacionais e ensino de Computação Distribuída, que fundamenta as escolhas do arcabouço (Capítulo 2 e Capítulo 3); (ii) **concepção do arcabouço**, materializada na matriz conceitual que correlaciona metodologias, estratégias, tecnologias, tópicos e atividades (Capítulo 4); (iii) **prototipação** das quatro atividades práticas e do ambiente padronizado que viabiliza sua execução, mantidos em repositórios públicos (Capítulo 4); (iv) **instrumentação da avaliação**, com a construção do questionário pós-atividade (escala Likert, TAM e NASA-TLX) e do relatório reflexivo com gabarito de análise (Capítulo 5); e (v) **aplicação piloto** das atividades em uma turma real da disciplina de Computação Distribuída da UFSC, seguida da análise das evidências coletadas e da identificação de refinamentos (Capítulo 5 e Capítulo 6).

1.2 ORGANIZAÇÃO DO DOCUMENTO

A estrutura deste trabalho é organizada da seguinte forma: o Capítulo 2 apresenta a fundamentação teórica que sustenta a proposta, percorrendo as metodologias de ensino ativas (PBL, PjBL, TBL, Sala de Aula Invertida, Instrução pelos Pares e IBL), as estratégias pedagógicas (*scaffolding*, *coding dojo* e programação em pares) e as tecnologias educacionais (contêineres, máquinas virtuais e *scripts*) que viabilizam ambientes reprodutíveis de laboratório. O Capítulo 3 discute os trabalhos relacionados, organizando a literatura em torno das dificuldades de iniciantes em Computação Distribuída, da aplicação de metodologias ativas no ensino de computação, das infraestruturas de laboratório utilizadas em disciplinas afins e dos exercícios

práticos já documentados, encerrando com uma síntese das lacunas que motivam esta proposta. O Capítulo 4 detalha o arcabouço didático-metodológico em si, apresentando a matriz conceitual que articula metodologias, estratégias, tópicos e atividades, a descrição de quatro atividades práticas (*sockets*, espaço de tuplas, sincronização de relógios via NTP e consenso com Raft), a arquitetura do ambiente padronizado e os instrumentos de avaliação propostos. O Capítulo 5 relata a aplicação em sala de aula de parte das atividades propostas na disciplina de Computação Distribuída da UFSC, descrevendo o contexto da aplicação, os instrumentos utilizados e a análise empírica dos dados coletados. Por fim, o Capítulo 6 sintetiza as contribuições do trabalho frente aos objetivos definidos, discute limitações encontradas e aponta direções para trabalhos futuros.

DECLARAÇÃO DE USO DE INTELIGÊNCIA ARTIFICIAL GENERATIVA

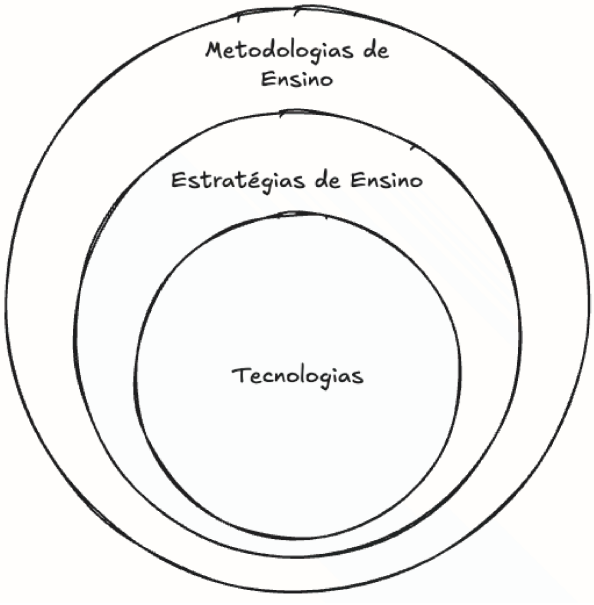
Em conformidade com a Portaria CNPq nº 2.664/2026 (Conselho Nacional de Desenvolvimento Científico e Tecnológico, 2026), que institui a Política de Integridade na Atividade Científica e estabelece diretrizes nacionais sobre o uso de Inteligência Artificial Generativa (IAG) em atividades de pesquisa, declara-se a seguir o emprego dessa ferramenta na elaboração do presente trabalho. Foi utilizada a ferramenta Claude (Anthropic, versões Sonnet 4.6 e Opus 4.7, acessada via interface Claude Code), aplicada em fases distintas do desenvolvimento do trabalho. Na fase de redação textual, a ferramenta apoiou a revisão linguística e ortográfica do português acadêmico, a sugestão de conectivos e a reorganização pontual de parágrafos, sendo todas as sugestões submetidas à verificação e reescrita pelo autor antes de incorporação ao texto final. Na fase de implementação dos artefatos de *software* associados às atividades práticas, a ferramenta foi empregada para depuração de código, geração de *boilerplate* em *Dockerfiles*, *scripts* de automação e estruturas iniciais de classes, com validação posterior por execução e teste manual. Na fase de análise e visualização de dados, parte dos gráficos apresentados neste trabalho foi gerada por meio de *scripts* Python elaborados com auxílio da ferramenta, sendo o código revisado pelo autor e os resultados validados por inspeção manual contra os dados originais antes da inclusão no texto. Não foi utilizada ferramenta de IAG para a formulação da hipótese central, a definição da matriz conceitual, a análise crítica da literatura, a interpretação dos dados empíricos coletados na aplicação piloto, nem para a produção de qualquer trecho submetido como original de autoria humana. Em conformidade com o disposto na referida portaria, o autor assume integral responsabilidade pelo conteúdo final apresentado neste trabalho, incluindo eventuais imprecisões, omissões ou ocorrências de plágio que tenham origem nas ferramentas utilizadas.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capitulo, são apresentadas algumas metodologias de ensino, estratégias de ensino e tecnologias educacionais utilizadas para o ensino de Computação Distribuída. Segundo Berbel (Berbel, 2011), as metodologias ativas de aprendizagem colocam o estudante como protagonista no processo de construção do conhecimento, estimulando a autonomia, a colaboração e a reflexão crítica. Já as estratégias de ensino correspondem a ações e recursos específicos utilizados pelo professor para operacionalizar a metodologia e favorecer a aprendizagem (Morán et al., 2015). As tecnologias educacionais, por sua vez, são ferramentas que apoiam e potencializam o processo de ensino-aprendizagem. Moran (Morán et al., 2015) ressalta que tecnologias educacionais funcionam como meios e não como fins, pois seu valor pedagógico decorre da intencionalidade com que são integradas ao projeto de ensino, e enfatiza a importância de o docente manter-se atualizado e incorporar progressivamente as novas tecnologias à prática de sala de aula. Neste capítulo, apresentam-se algumas tecnologias que têm sido utilizadas para auxiliar no ensino de Computação Distribuída, explorando como podem contribuir para uma aprendizagem mais efetiva e engajadora.

Tendo clarificado esses conceitos principais, podemos representar de forma visual a relação entre metodologias, estratégias de ensino e tecnologias educacionais, conforme ilustrado na Figura 1. Onde cada tecnologia pode ser associada a uma ou mais estratégias de ensino e uma estratégia de ensino pode ser associada a uma ou mais metodologias.

Figura 1 – Relação entre metodologias, estratégias de ensino e tecnologias educacionais.



Fonte: Elaborado pelo autor (2026).

2.1 METODOLOGIAS DE ENSINO

2.1.1 Aprendizagem baseada em problemas

A Aprendizagem Baseada em Problemas (*Problem-Based Learning* – PBL) é uma metodologia educacional que utiliza problemas como ponto de partida para o processo de aprendizagem (Wood, 2003). Segundo Wood, essa abordagem não se concentra somente na resolução de problemas, mas utiliza cenários relevantes para estimular a aquisição de conhecimento e compreensão pelos próprios alunos.

O ciclo de aprendizagem na PBL inicia-se com a apresentação de um problema ou cenário que serve como "gatilho" para o aprendizado (Wood, 2003). Em seguida, os estudantes, trabalhando em grupo, discutem o problema apresentado, identificam o que já sabem sobre o tema e determinam quais informações adicionais precisam adquirir para compreendê-lo adequadamente. Após essa discussão inicial, cada estudante realiza um estudo autodirigido para pesquisar e aprender sobre os tópicos identificados como necessários. Com o conhecimento adquirido individualmente, os estudantes retornam ao grupo para compartilhar o que aprenderam, discutir e integrar as novas informações ao contexto do problema original. Finalmente, o grupo aplica o conhecimento construído coletivamente para propor soluções ou explicações para o problema inicial.

Essa metodologia promove o aprendizado ativo, o desenvolvimento de habilidades de pensamento crítico, colaboração em grupo e responsabilidade pelo próprio aprendizado (Wood, 2003). A PBL é fundamentada na teoria da aprendizagem de adultos e no construtivismo, enfatizando a construção do conhecimento pelos próprios estudantes.

2.1.2 Aprendizagem baseada em projetos

A Aprendizagem Baseada em Projetos (*Project-Based Learning* – PjBL) é uma forma de instrução centrada no estudante, fundamentada em três princípios construtivistas: a aprendizagem é específica ao contexto, os aprendizes estão ativamente envolvidos no processo de aprendizagem, e eles alcançam seus objetivos por meio de interações sociais e do compartilhamento de conhecimento e compreensão (Kokotsaki; Menzies; Wiggins, 2016). Trata-se de um tipo particular de aprendizagem baseada em investigação, onde o contexto de aprendizagem é fornecido por meio de questões autênticas e problemas dentro de práticas do mundo real, que conduzem a experiências significativas de aprendizagem.

A formalização dessa metodologia remonta a 1918, quando William Heard Kilpatrick, influenciado por seu mentor John Dewey, apresentou o "Método de Projetos" em seu artigo seminal "The Project Method: The Use of the Purposeful Act in the Educative Process" (Kilpatrick, 1918). Kilpatrick propôs que a educação deveria ser centrada no aluno, permitindo que este direcionasse seu próprio aprendizado de acordo com seus interesses e experiências. O método é caracterizado por um processo cíclico composto por quatro etapas fundamentais: pro-

posição do propósito (*purposing*), planejamento (*planning*), execução (*executing*) e avaliação (*judging*). Segundo Kilpatrick, todas essas fases devem ser conduzidas pelo próprio aluno, com o professor atuando como facilitador e orientador do processo.

Na conceituação contemporânea, (Kokotsaki; Menzies; Wiggins, 2016) identificam cinco características essenciais que definem a Aprendizagem Baseada em Projetos: (1) *Centralidade*: o projeto é central ao currículo, não é periférico ou suplementar; (2) *Questão Norteadora*: o projeto é organizado em torno de uma questão ou problema que conduz os estudantes a encontrar conceitos e princípios centrais da disciplina; (3) *Investigações Construtivas*: os estudantes se envolvem em uma busca construtiva que envolve questionamento, construção de conhecimento e resolução de problemas; (4) *Autonomia*: os projetos são, em certa medida, dirigidos pelos estudantes, não inteiramente prescritos pelo professor; e (5) *Realismo*: os projetos são autênticos, não simulados, envolvendo desafios da vida real e soluções com potencial de implementação.

Um aspecto distintivo da PjBL em relação a outras metodologias ativas é a construção de um produto final concreto, um "artefato concreto" (Kokotsaki; Menzies; Wiggins, 2016), que representa a compreensão, o conhecimento e as atitudes dos estudantes em relação à questão investigada. Esse produto pode assumir diversas formas, como vídeos, fotografias, esquetes, relatórios, modelos, apresentações e deve ser apresentado e compartilhado, frequentemente com audiências além da sala de aula. A construção desse produto final diferencia a PjBL da aprendizagem baseada em problemas, onde o foco está primordialmente no processo de aprendizagem.

A abordagem enfatiza a importância de conectar a educação à vida real, promovendo a autonomia, o pensamento crítico, a colaboração, a comunicação e a reflexão dos alunos (Kokotsaki; Menzies; Wiggins, 2016). Ao engajarem-se em projetos que refletem problemas autênticos e práticas do mundo real, os estudantes desenvolvem não somente habilidades práticas e conhecimento conceitual profundo, mas também competências socioemocionais essenciais para a cidadania democrática e para a vida no século XXI. O professor, nesse contexto, desempenha papel fundamental como facilitador, fornecendo suporte adequado (*scaffolding*), orientação e feedback ao longo do processo, equilibrando instrução direta com métodos de investigação em profundidade.

2.1.3 Aprendizagem baseada em equipe

A Aprendizagem Baseada em Equipe (*Team-Based Learning* – TBL) é uma metodologia educacional estruturada que combina aprendizagem individual e em grupo para maximizar o engajamento dos estudantes e a qualidade do aprendizado (Parmelee et al., 2012).

A TBL é fundamentada em quatro princípios essenciais que garantem sua eficácia: grupos permanentes e heterogêneos, responsabilidade individual e grupal, feedback frequente e imediato, e tarefas de equipe que promovem tanto o aprendizado quanto o desenvolvimento de habilidades de trabalho em equipe (Parmelee et al., 2012). Esses princípios trabalham em conjunto para criar um ambiente de aprendizagem colaborativa que motiva os estudantes a se

engajarem ativamente no processo educacional.

O processo de implementação da TBL segue uma estrutura bem definida que inicia com a preparação individual dos estudantes mediante leituras prévias e materiais de estudo. Em seguida, os estudantes realizam um teste individual (*Individual Readiness Assurance Test* – iRAT) que avalia sua compreensão do conteúdo estudado. Imediatamente após, o mesmo teste é aplicado em equipe (*Team Readiness Assurance Test* – tRAT), permitindo que os membros discutam e cheguem a um consenso sobre as respostas. Durante o tRAT, os estudantes têm a oportunidade de apelar de questões que consideram mal formuladas ou ambíguas, promovendo o pensamento crítico e a argumentação fundamentada.

Após a fase de avaliação, os estudantes participam de atividades de aplicação (*Application Activities*) que consistem em problemas complexos e realistas que requerem a aplicação do conhecimento adquirido. Essas atividades são projetadas para serem resolvidas em equipe, promovendo a discussão, o debate e a tomada de decisão colaborativa. Os problemas apresentados são intencionalmente complexos e abertos, sem uma única resposta correta, estimulando o pensamento crítico e a criatividade dos estudantes.

A implementação bem-sucedida da TBL requer planejamento cuidadoso e atenção aos detalhes. Os grupos devem ser formados de forma permanente e intencionalmente heterogênea, considerando fatores como experiência prévia, habilidades e personalidade dos estudantes. As atividades de aplicação devem ser desafiadoras o suficiente para promover discussão significativa, mas não tão complexas que desencorajem a participação. O professor deve atuar como facilitador, guiando as discussões e fornecendo feedback construtivo sem dominar o processo de aprendizagem.

2.1.4 Sala de aula invertida

A Sala de Aula Invertida (*Inverted Classroom* ou *Flipped Classroom*) é uma metodologia educacional que inverte a lógica tradicional do ensino, de modo que eventos que tradicionalmente ocorriam dentro da sala de aula passam a ocorrer fora dela e vice-versa (Lage; Platt; Treglia, 2000). Segundo Lage; Platt; Treglia, essa abordagem utiliza tecnologias de aprendizagem, particularmente recursos multimídia, para oferecer novas oportunidades aos estudantes, permitindo que tenham acesso a aulas e outros materiais expositivos fora do ambiente de sala de aula, enquanto o tempo presencial é dedicado a atividades práticas, discussões e resolução de problemas em grupo.

O processo de implementação da sala de aula invertida oferece aos estudantes um conjunto de opções para aprender o conteúdo. Antes da aula presencial, os estudantes têm acesso a diversos formatos de recursos de aprendizagem, como videoaulas, apresentações multimídia com áudio, slides para download ou materiais impressos, permitindo que cada estudante escolha o método que melhor se adequa ao seu estilo de aprendizagem (Lage; Platt; Treglia, 2000). Além da leitura do material textual obrigatório, os estudantes são encorajados a estudar o conteúdo através dos recursos disponibilizados. Durante o encontro presencial, a aula inicia-se com

o professor perguntando se há dúvidas sobre o material estudado. As questões dos estudantes geralmente levam a uma breve explicação ou mini-aula de aproximadamente 10 minutos. Caso não haja dúvidas, o professor não realiza uma aula puramente expositiva, interpretando a ausência de questionamentos como sinal de compreensão adequada do material.

Após o esclarecimento de dúvidas, o restante do tempo de aula é dedicado a atividades práticas que promovem a aplicação ativa dos conceitos. Os estudantes participam de experimentos práticos que demonstram os princípios teóricos em ação, trabalham em exercícios práticos para uma primeira aplicação do conteúdo, e resolvem problemas mais complexos em pequenos grupos, apresentando posteriormente seus resultados para toda a classe (Lage; Platt; Treglia, 2000). Essa estrutura garante que os estudantes cheguem preparados para discussão e aplicação prática do conteúdo, promovendo maior responsabilidade e protagonismo no processo de aprendizagem.

A metodologia da sala de aula invertida fundamenta-se no reconhecimento de que os estudantes possuem estilos de aprendizagem distintos (Lage; Platt; Treglia, 2000). Ao variar os métodos de ensino e permitir que os estudantes escolham como acessar o conteúdo, essa abordagem pode atender a perfis de aprendizagem dependentes (que requerem direcionamento por meio de materiais estruturados), colaborativos (que se beneficiam do trabalho em equipe durante as atividades presenciais) e independentes (que preferem estudar por conta própria usando os recursos disponibilizados). Além disso, Lage; Platt; Treglia destacam que essa metodologia cria um ambiente de aprendizagem mais inclusivo, beneficiando particularmente estudantes que podem ter menos oportunidades de participação em formatos de aula puramente expositivos, sem sacrificar a cobertura do conteúdo programático.

2.1.5 Instrução pelos Pares (Peer Instruction)

A Instrução pelos Pares (*Peer Instruction* – PI) é uma metodologia educacional que modifica o formato tradicional de aula para incluir questões conceituais projetadas para engajar os estudantes e revelar dificuldades com o material (Crouch; Mazur, 2001). Segundo Crouch; Mazur, essa abordagem foi desenvolvida em resposta à constatação de que muitos estudantes aprendem muito pouco com aulas tradicionais, mesmo quando conseguem resolver algoritmos de problemas com sucesso. A PI fundamenta-se no reconhecimento de que os estudantes desenvolvem habilidades de raciocínio complexo de forma mais efetiva quando estão ativamente engajados com o material que estão estudando.

O processo de implementação da Instrução pelos Pares segue uma estrutura bem definida que divide a aula em uma série de apresentações curtas, cada uma focada em um ponto central e seguida por uma questão conceitual relacionada, denominada *ConcepTest* (Crouch; Mazur, 2001). Os estudantes recebem um ou dois minutos para formular respostas individuais e reportá-las ao instrutor. Em seguida, os estudantes discutem suas respostas com outros ao redor, sendo encorajados pelo instrutor a tentar convencer reciprocamente da correção de suas próprias respostas, explicando o raciocínio subjacente. Durante a discussão, que tipicamente

dura de dois a quatro minutos, o instrutor circula pela sala ouvindo as conversas. Finalmente, o instrutor encerra a discussão, consulta novamente os estudantes sobre suas respostas (que podem ter mudado baseadas na discussão), explica a resposta correta e prossegue para o próximo tópico. Para liberar tempo de aula para os ConcepTests e preparar melhor os estudantes para aplicar o material durante a aula, os estudantes são fortemente encorajados a completar a leitura sobre os tópicos a serem cobertos antes da aula.

Os resultados de dez anos de ensino com a Instrução pelos Pares demonstram melhorias significativas na compreensão dos estudantes (Crouch; Mazur, 2001). O ganho normalizado no *Force Concept Inventory* (FCI), um teste padronizado de compreensão conceitual de assuntos da Física, dobrou de 0,25 (instrução tradicional) para 0,49 (primeiro ano de PI), e continuou a aumentar com refinamentos subsequentes da implementação, atingindo 0,74 em 1997. Além disso, quase metade das respostas corretas dadas aos ConcepTests foram alcançadas após a discussão entre pares, e a vasta maioria dos estudantes que revisam suas respostas durante a discussão muda de uma resposta incorreta para a correta. Os estudantes se beneficiam mais da discussão quando a porcentagem inicial de respostas corretas está entre 35% e 70%, sendo o maior benefício observado quando essa porcentagem está em torno de 50%.

A metodologia da Instrução pelos Pares promove o engajamento ativo de todos os estudantes na classe, diferentemente da prática comum de fazer perguntas informais durante uma aula, que tipicamente engaja somente alguns estudantes altamente motivados (Crouch; Mazur, 2001). Essa abordagem desenvolve não somente a compreensão conceitual, mas também habilidades de explicação e comunicação, já que os estudantes devem articular seu raciocínio para convencer seus pares. A PI pode ser combinada com outras estratégias, como atribuições de leitura pré-aula e atividades cooperativas nas seções de discussão, e tem se mostrado adaptável a uma ampla gama de contextos e estilos de instrutor. A flexibilidade da metodologia permite que os instrutores adaptem o número de questões por aula, o tempo dedicado a cada questão e a quantidade de exposição tradicional conforme melhor se adequar ao contexto específico e aos objetivos do curso.

2.1.6 Aprendizagem Baseada em Investigação (Inquiry-Based Learning)

A Aprendizagem Baseada em Investigação (*Inquiry-Based Learning* – IBL) é uma estratégia educacional na qual os estudantes seguem métodos e práticas análogas às de cientistas para construir conhecimento (Pedaste et al., 2015). A abordagem define-se como um processo de descoberta de novas relações causais, no qual o aprendiz formula hipóteses e as testa por meio de experimentação e/ou observação, enfatizando a participação ativa e a responsabilidade do estudante na descoberta de conhecimento que é novo para si.

Com base em uma revisão sistemática da literatura, Pedaste et al. propuseram um modelo que consolida o ciclo em cinco fases gerais: Orientação, Conceituação, Investigação, Conclusão e Discussão. Este modelo oferece uma estrutura abrangente para a implementação da IBL.

O ciclo inicia-se com a fase de **Orientação**, que visa estimular o interesse e a curiosidade sobre um problema, resultando em sua formulação clara. Segue-se a **Conceituação**, onde o problema é embasado teoricamente através da formulação de questões de pesquisa (subfase de Questionamento) ou da geração de hipóteses testáveis (subfase de Geração de Hipóteses). A escolha do caminho depende do conhecimento prévio do estudante: questões mais abertas levam a uma abordagem exploratória (indutiva), enquanto ideias mais específicas, baseadas em teoria, conduzem a uma abordagem experimental (dedutiva) (Pedaste et al., 2015).

A fase de **Investigação** é onde a curiosidade se transforma em ação. Nela, os estudantes planejam e conduzem atividades para coletar e analisar dados, seja através da Exploração (busca de relações entre variáveis sem uma hipótese definida) ou da Experimentação (teste sistemático de uma hipótese). Ambas as abordagens culminam na Interpretação de Dados, que consiste na síntese do novo conhecimento. Na fase de **Conclusão**, os resultados da investigação são utilizados para responder às questões de pesquisa ou validar as hipóteses formuladas inicialmente. Por fim, a fase de **Discussão**, que engloba a Comunicação (apresentação de resultados a outros) e a Reflexão (análise sobre o processo de aprendizagem), pode ocorrer em qualquer etapa do ciclo, funcionando como um processo metacognitivo contínuo (Pedaste et al., 2015).

Embora o ciclo apresente uma sequência lógica, sua aplicação é flexível, permitindo diferentes percursos e iterações. Por exemplo, os resultados da fase de Investigação podem levar o estudante de volta à Conceituação para refinar suas hipóteses. Contudo, o sucesso da IBL não reside somente na estrutura do ciclo, mas depende criticamente da orientação (*guidance*) oferecida ao estudante (Lazonder; Harmsen, 2016). Uma meta-análise conduzida por Lazonder; Harmsen demonstrou que a IBL guiada supera métodos de instrução mais expositivos. A orientação adequada, que pode variar desde restrições no problema até explicações explícitas, é fundamental para apoiar o desenvolvimento das habilidades de raciocínio científico e garantir a eficácia da aprendizagem.

2.1.7 Outras metodologias

Outras metodologias ativas têm sido aplicadas no ensino de Computação e merecem registro, ainda que não tenham sido aprofundadas neste capítulo. O POGIL (*Process Oriented Guided Inquiry Learning*) constitui uma implementação altamente estruturada da aprendizagem por investigação e cooperativa, caracterizada pela definição de papéis específicos para os estudantes e por um ciclo de atividades bem delimitado. A Aprendizagem Baseada em Casos (*Case-Based Learning*) utiliza narrativas e situações reais ou realísticas como ponto de partida para análise crítica e tomada de decisão, enquanto a Aprendizagem para o Domínio (*Mastery Learning*) fundamenta-se na premissa de que todos os estudantes podem alcançar domínio sobre o conteúdo quando recebem tempo e suporte adequados. O recorte adotado privilegiou as seis metodologias detalhadas anteriormente por sua maior recorrência na literatura específica sobre ensino de Computação Distribuída e Paralela revisada para este trabalho (Conte et al., 2020; Manoharan; Ye, 2023).

2.2 ESTRATÉGIAS DE ENSINO

2.2.1 Andaime (*Scaffolding*)

O Andaime (*Scaffolding*) é uma estratégia educacional na qual um tutor, instrutor ou par mais experiente oferece suporte temporário ao aprendiz durante a realização de uma tarefa que estaria além de sua capacidade se executada de forma autônoma (Wood; Bruner; Ross, 1976). Segundo Wood; Bruner; Ross, esse processo consiste essencialmente no tutor "controlar" aqueles elementos da tarefa que estão inicialmente além da capacidade do aprendiz, permitindo-lhe concentrar-se e completar apenas os elementos que estão dentro de seu alcance de competência. A metáfora do andaime, emprestada do campo da construção civil onde andaimes são estruturas temporárias erguidas para auxiliar na construção ou modificação de outra estrutura, refere-se ao suporte provisório fornecido para a conclusão de uma tarefa que os aprendizes de outra forma não seriam capazes de completar (van de Pol; Volman; Beishuizen, 2010).

A literatura contemporânea identifica três características-chave que definem o andaime (van de Pol; Volman; Beishuizen, 2010): (1) *Contingência*: o suporte do tutor deve ser adaptado ao nível atual de desempenho do estudante, sendo ajustado de forma responsiva às suas respostas; (2) *Esvanecimento* (*fading*): a retirada gradual do suporte à medida que o aprendiz desenvolve competência, com a taxa de esvanecimento dependendo do nível de desenvolvimento e competência do estudante; e (3) *Transferência de responsabilidade*: através do esvanecimento contingente, a responsabilidade pela execução da tarefa é gradualmente transferida ao aprendiz, que assume controle crescente sobre sua própria aprendizagem.

Wood; Bruner; Ross identificaram seis funções específicas que caracterizam o processo de andaime (Wood; Bruner; Ross, 1976). A primeira função, *Recrutamento*, consiste em despertar o interesse do aprendiz e garantir sua adesão aos requisitos da tarefa. A segunda, *Redução dos graus de liberdade*, envolve simplificar a tarefa reduzindo o número de atos constituintes necessários para alcançar a solução, permitindo que o aprendiz reconheça se atingiu ou não os requisitos da tarefa. A terceira função, *Manutenção da direção*, refere-se a manter o aprendiz em busca de um objetivo particular, evitando que ele se desvie ou regrida para outros fins. A quarta, *Marcação de características críticas*, consiste em acentuar certas características relevantes da tarefa, fornecendo informação sobre a discrepância entre o que o estudante produziu e o que seria reconhecido como uma produção correta. A quinta função, *Controle da frustração*, visa tornar a resolução de problemas menos estressante com a presença do tutor do que seria sem ela. Por fim, a sexta função, *Demonstração*, envolve a modelagem de soluções de forma idealizada, frequentemente completando ou explicitando uma solução parcialmente executada pelo próprio aprendiz.

A eficácia do andaime como estratégia educacional tem sido demonstrada em diversos estudos, indicando que ele apoia efetivamente as atividades metacognitivas e cognitivas dos estudantes, bem como seu engajamento afetivo (van de Pol; Volman; Beishuizen, 2010). No contexto do ensino de Computação Distribuída, essa estratégia mostra-se particularmente rele-

vante dado que conceitos como sincronização, consistência, tolerância a falhas e comunicação assíncrona impõem elevada carga cognitiva aos iniciantes (Wood; Bruner; Ross, 1976). A aplicação do andaime permite estruturar progressivamente o suporte oferecido, desde a configuração inicial de ambientes através de *scripts*, contêineres e automação, até a transferência gradual da responsabilidade pela implementação completa de sistemas distribuídos. Dessa forma, os estudantes podem concentrar-se nos conceitos fundamentais enquanto desenvolvem as competências técnicas para operar de forma independente.

2.2.2 Coding Dojo

O Coding Dojo é um encontro periódico (geralmente semanal) organizado em torno de um desafio de programação onde pessoas são encorajadas a participar e compartilhar suas habilidades de codificação com a audiência enquanto resolvem o problema (Sato; Corbucci; Bravo, 2008). Segundo Sato; Corbucci; Bravo, embora a sessão seja organizada em torno de um desafio de programação, o objetivo principal de um Coding Dojo é aprender com outros e melhorar habilidades de design e codificação através da prática deliberada. Isso cria um ambiente de aprendizagem onde práticas técnicas ágeis, como as propostas pela Programação Extrema (*Extreme Programming* - XP), podem ser compartilhadas.

A origem do Coding Dojo remonta à proposta de Dave Thomas sobre o conceito de *Code Kata*, um exercício onde programadores poderiam escrever código descartável para praticar sua arte fora de um ambiente de trabalho (Sato; Corbucci; Bravo, 2008). Posteriormente, Laurent Bossavit propôs a ideia de um Coding Dojo: uma sessão onde um grupo de programadores se reuniria para resolver o Code Kata juntos. Os principais princípios do Coding Dojo são criar um *Ambiente Seguro* (*Safe Environment*) que seja colaborativo, inclusivo e não competitivo, onde as pessoas possam estar *Continuamente Aprendendo* (*Continuously Learning*). Alguns princípios da XP alinham-se bem com isso: *Falha*, pois é aceitável falhar ao aprender algo novo; *Redundância*, pois sempre é possível obter novos insights ao abordar o mesmo problema com diferentes estratégias; e *Passos de Bebê* (*Baby Steps*), em que cada passo em direção à solução deve ser pequeno o suficiente para que todos possam compreender e replicar posteriormente.

Existem dois formatos principais de reunião do Coding Dojo (Sato; Corbucci; Bravo, 2008). O primeiro formato, denominado *Kata Preparado* (*Prepared Kata*), ocorre quando alguém já resolveu o Kata proposto antes da reunião (sozinho ou em grupo) e a solução é apresentada à audiência durante a sessão. Em vez de mostrar o código e testes finais, os apresentadores começam do zero, explicando cada passo e permitindo que os outros participantes façam perguntas ou sugestões. O objetivo da sessão é que todos sejam capazes de reproduzir os passos e resolver o mesmo problema após a reunião. O segundo formato, denominado *Randori*, envolve todos os participantes trabalhando juntos no problema, seguindo TDD e Programação em Pares (*Pair Programming*) em rodadas com tempo limitado,geralmente entre 5 e 7 minutos. Ao final de cada rodada, o piloto junta-se à audiência, o co-piloto torna-se piloto, e um novo co-piloto

junta-se ao par a partir da audiência. Uma regra adicional é que discussões e sugestões devem ser dadas apenas quando o par chega em uma barra verde, com todos os testes atuais passando. A razão é que, enquanto em uma barra vermelha, o par deve focar e trabalhar junto para fazer os testes passarem. A audiência pode sempre sugerir refatorações e otimizações em uma barra verde.

O processo típico de uma sessão de Coding Dojo segue uma estrutura bem definida (Sato; Corbucci; Bravo, 2008). Inicia-se com a *Escolha do Problema* (5 a 10 minutos), onde os participantes recebem opções de problemas a serem resolvidos e votam em qual será abordado. Em seguida, ocorre a *Discussão do Problema* (10 a 20 minutos), onde o grupo discute as diferentes abordagens para resolvê-lo, geralmente chegando a uma abordagem acordada e uma lista de itens TO-DO para guiar os pares durante a implementação. A *Sessão de Codificação* (1 a 2 horas) segue um dos dois formatos, Kata Preparado ou Randori, com os participantes praticando Programação em Pares e Desenvolvimento Orientado por Testes (*Test-Driven Development* - TDD) como regra geral. Finalmente, a sessão encerra com uma *Retrospectiva* (10 a 20 minutos), onde os participantes param de codificar (mesmo que o problema não tenha sido completamente resolvido) para refletir sobre a experiência e compartilhar as lições aprendidas com o grupo.

Esses formatos permitem a criação de um ambiente onde os participantes podem discutir e praticar uma ampla gama de tópicos, tais como: TDD, Desenvolvimento Orientado por Comportamento (*Behaviour-Driven Development*), práticas ágeis, Refatoração, Programação em Pares, Design Orientado a Objetos, Algoritmos, diferentes linguagens de programação, paradigmas e frameworks (Sato; Corbucci; Bravo, 2008). Um princípio fundamental estabelecido pela experiência do Coding Dojo de São Paulo é que "é aceitável não terminar". Completar o problema nunca deve ser o objetivo de uma reunião. Mais do que isso, foi acordado que não escrever a solução inteira é aceitável, desde que os participantes possam aprender algo da sessão de codificação. Com esse princípio em mente, os participantes dedicam tempo para escrever código limpo e compreensível, e o grupo frequentemente não termina implementando a solução completa para os problemas. Diferentemente de um desafio de programação ou concurso, ir rápido em uma sessão de Coding Dojo não é benéfico.

A eficácia do Coding Dojo como estratégia educacional fundamenta-se no conceito de prática deliberada, estudado por Ericsson et al. em diferentes domínios como música, xadrez e esportes (Sato; Corbucci; Bravo, 2008). A pesquisa demonstra que a prática deliberada ao longo de um longo período de tempo (geralmente mais de 10 anos) está no cerne da obtenção de expertise. Embora leve tempo para se tornar um especialista, o papel da prática deliberada continua importante ao longo do processo de aprendizagem.

No contexto do ensino de Computação Distribuída, o Coding Dojo mostra-se particularmente relevante para criar um ambiente seguro onde os estudantes podem praticar conceitos complexos como sincronização, comunicação entre processos, consistência e tolerância a falhas sem a pressão de um ambiente de trabalho real. A estrutura colaborativa e não competitiva permite que estudantes de diferentes níveis de habilidade compartilhem conhecimento e desenvol-

vam competências técnicas através da prática guiada, enquanto a ênfase em TDD e refatoração ajuda a construir sistemas distribuídos mais robustos e testáveis.

2.2.3 Programação em Pares (*Pair Programming*)

A Programação em Pares (*Pair Programming*) é uma estratégia de desenvolvimento de software na qual dois programadores trabalham lado a lado em um único computador, colaborando na produção de um mesmo artefato (projeto, algoritmo ou código) (Williams et al., 2000). Segundo Williams et al., na programação em pares, os dois programadores funcionam como um organismo unificado e inteligente trabalhando com uma única mente, sendo ambos responsáveis por todos os aspectos do artefato produzido. Um dos parceiros, denominado *driver* (piloto), controla o teclado ou mouse e escreve o código, enquanto o outro parceiro, denominado *navigator* (navegador), observa continuamente e ativamente o trabalho do piloto, procurando por defeitos, pensando em alternativas, consultando recursos e considerando as implicações estratégicas da implementação. Os parceiros alternam deliberadamente seus papéis periodicamente, sendo ambos participantes ativos e iguais em todos os momentos e compartilhando completamente a propriedade do produto do trabalho.

A origem da programação em pares remonta a observações de Larry Constantine em 1995, que reportou ter observado pares de programadores produzindo código mais rápido e com menos bugs do que nunca antes (Constantine, 1995). No mesmo ano, Jim Coplien publicou o que chamou de padrão organizacional de desenvolvimento em pares. Em 1996, Kent Beck, junto com Ward Cunningham e Ron Jeffries, começou a desenvolver a metodologia de desenvolvimento de software *Extreme Programming* (*XP*), da qual a programação em pares é uma parte significativa (Beck, 2000). A metodologia XP tornou-se amplamente praticada por programadores em todo o mundo, e seus fundadores creditam muito de seu sucesso ao uso da programação em pares por todos os programadores XP, tanto especialistas quanto iniciantes. Em 1998, o professor John Nosek da Temple University reportou um estudo com 15 programadores experientes trabalhando em um problema desafiador, onde os pares completaram a tarefa 40% mais rápido que os grupos de controle e produziram algoritmos e código melhores, apesar de terem gasto 60% mais minutos combinados na tarefa (Nosek, 1998).

A eficácia da programação em pares tem sido validada através de evidências quantitativas tanto em ambientes profissionais quanto acadêmicos (Williams et al., 2000). Em 1999, na Universidade de Utah, estudantes de engenharia de software participaram de um experimento estruturado que comparou tempo de ciclo, produtividade e qualidade entre grupos que trabalhavam individualmente e grupos que trabalhavam em pares colaborativos. Os resultados mostraram que os pares sempre passaram em mais casos de teste automatizados pós-desenvolvimento, com diferenças estatisticamente significativas ($p < 0{,}01$). Além disso, os resultados dos pares foram mais consistentes, enquanto os indivíduos variaram mais em relação à média. Após o período inicial de ajuste, o tempo total de programador que os pares gastaram em cada tarefa diminuiu dramaticamente de 60% para um mínimo de 15%, e os pares completaram suas tare-

fas 40% a 50% mais rápido do que os indivíduos. Vários fatores explicam por que as horas de programador não dobram com a programação em pares como se poderia esperar: a colaboração melhora o processo de resolução de problemas, resultando em muito menos tempo gasto nas fases caóticas e demoradas de compilação e teste; os pares descobrem que, ao combinar seu conhecimento conjunto, podem conquistar quase qualquer tarefa técnica imediatamente; e os programadores em pares exercem pressão positiva uns sobre os outros para desempenhar, mantendo o parceiro focado e na tarefa.

Além dos benefícios de qualidade e produtividade, a programação em pares é uma das poucas técnicas propostas para melhorar a qualidade e produtividade de software que os programadores realmente apreciam (Williams et al., 2000). Em uma pesquisa online com programadores profissionais em pares, 96% declararam que apreciaram seu trabalho mais do que quando programavam sozinhos. Em seis pesquisas realizadas com os 41 programadores colaborativos no experimento universitário, consistentemente mais de 90% declararam que apreciaram a programação colaborativa mais do que a programação solo. Virtualmente todos os programadores profissionais pesquisados declararam que estavam mais confiantes em suas soluções quando programavam em pares, e quase 95% dos estudantes ecoaram essa declaração. Esta correlação natural entre satisfação e confiança decorre do fato de que há alguém presente para ajudar se estiverem confusos ou sem conhecimento, podem debater ideias com um parceiro, passam mais tempo fazendo design desafiador e menos tempo fazendo *debugging* irritante, e saem de cada sessão com uma sensação de euforia compartilhada após a conclusão bem-sucedida de uma tarefa colaborativa.

No contexto do ensino de Computação Distribuída, a programação em pares destaca-se por sua adequação à natureza intrinsecamente concorrente e não-determinística dos sistemas estudados. Falhas em programas distribuídos frequentemente decorrem de condições de corrida, *deadlocks* ou inconsistências de estado que são difíceis de reproduzir e cuja detecção exige raciocínio simultâneo sobre múltiplos fluxos de execução. A presença constante do navegador, deliberando enquanto o piloto codifica, atua como um mecanismo de revisão contínua que antecipa esses defeitos no momento da escrita, em vez de adiá-los para a depuração posterior. Adicionalmente, a alternância periódica entre os papéis de piloto e navegador exercita a verbalização explícita de invariantes e suposições sobre ordem de mensagens, visibilidade de escritas e cenários de falha, tornando explícito o raciocínio que costuma permanecer implícito na programação individual. Por fim, o caráter colaborativo da técnica aproxima-se da prática real em equipes de engenharia que operam sistemas distribuídos, oferecendo aos estudantes uma vivência mais fiel ao contexto profissional em que esses conceitos são aplicados.

2.2.4 Outras Estratégias

Além de Andaime, Coding Dojo e Programação em Pares, o cenário do ensino de Computação inclui outras estratégias pedagógicas relevantes. A Gamificação (*Gamification*) incorpora elementos de jogos ao ambiente educacional para aumentar o engajamento e a mo-

tivação dos estudantes, como pontuação, níveis, conquistas e competição. A Programação ao Vivo (*Live Coding*) consiste na demonstração de programação em tempo real pelo instrutor, permitindo que os estudantes observem o processo de pensamento, tomada de decisão e resolução de problemas durante a escrita de código. O Ensino Just-in-Time (*Just-in-Time Teaching* - JiTT) fundamenta-se na preparação pré-aula dos estudantes através de atividades *online*, seguida de ajustes na aula presencial baseados nas respostas recebidas. A Aprendizagem Cooperativa (*Cooperative Learning*) estrutura o trabalho em grupos pequenos onde os estudantes trabalham juntos para alcançar objetivos compartilhados. Estas estratégias não foram aprofundadas porque sua articulação com as atividades práticas propostas neste trabalho é menos direta do que a do trio detalhado nas seções anteriores.

2.3 TECNOLOGIAS UTILIZÁVEIS PARA O ENSINO DE COMPUTAÇÃO PARALELA E DISTRIBUÍDA

2.3.1 Contêineres

Contêineres, constituem uma tecnologia de virtualização ao nível de sistema operacional que permite empacotar aplicações e suas dependências em ambientes isolados e portáveis (Merkel, 2014). O Docker, plataforma mais amplamente utilizada para conteinerização, revolucionou como aplicações são desenvolvidas, distribuídas e executadas ao encapsular código, bibliotecas, dependências e configurações em unidades padronizadas que podem ser executadas de forma consistente em diferentes ambientes computacionais.

No contexto educacional, particularmente no ensino de Computação Paralela e Distribuída, a conteinerização oferece uma solução para um dos desafios mais persistentes: a heterogeneidade de ambientes de desenvolvimento entre estudantes. Ao garantir que todos os alunos operem em ambientes computacionais idênticos, independentemente do sistema operacional hospedeiro ou das configurações específicas de suas máquinas, os contêineres eliminam o problema comum de "funciona na minha máquina", permitindo que o foco pedagógico se concentre nos conceitos fundamentais da disciplina ao invés de questões de configuração de ambiente (Malan, 2022).

Os benefícios da conteinerização para o ensino de Computação Distribuída são múltiplos e significativos. De maneira fundamental, os contêineres proporcionam ambientes consistentes e reprodutíveis, garantindo que todos os estudantes, independentemente de utilizarem Windows, macOS ou Linux, possam executar os mesmos exercícios práticos com resultados idênticos (Malan, 2022). Paralelamente, a tecnologia facilita o aprendizado prático ao permitir que estudantes implantem e gerenciem aplicações distribuídas reais, simulando arquiteturas complexas sem necessidade de infraestrutura física extensa. Neste contexto, ferramentas de orquestração como Docker Compose permitem a criação de sistemas multi-contêiner que simulam clusters e arquiteturas de microsserviços, possibilitando que estudantes experimentem com sistemas distribuídos autênticos em seus próprios computadores. Do ponto de vista de eficiência

computacional, os contêineres oferecem desempenho superior às máquinas virtuais tradicionais, consumindo menos memória e CPU, o que permite a execução simultânea de múltiplas instâncias mesmo em hardware com recursos limitados, característica particularmente relevante em contextos educacionais onde nem todos os estudantes possuem acesso a equipamentos de alto desempenho. Além disso, a exposição a ferramentas de conteinerização prepara os estudantes para ambientes profissionais, uma vez que Docker e tecnologias relacionadas tornaram-se padrão na indústria de software para desenvolvimento, integração contínua e implantação de aplicações distribuídas (O'connor et al., 2024).

As aplicações práticas de contêineres no ensino de Computação Distribuída são vastas. Estudantes podem utilizar contêineres para criar ambientes de desenvolvimento uniformes, onde frameworks, bibliotecas e ferramentas específicas são pré-configurados e distribuídos através de imagens Docker, eliminando horas de configuração manual e permitindo que as aulas comecem imediatamente com atividades práticas. Em laboratórios de redes e sistemas distribuídos, múltiplos contêineres podem simular diferentes nós de uma rede, servidores de aplicação, bancos de dados e sistemas de mensageria, proporcionando experiências práticas com arquiteturas complexas. Docker Compose simplifica a orquestração desses ambientes multi-serviço, permitindo que arquiteturas inteiras sejam definidas em arquivos de configuração declarativos e inicializadas com um único comando. Além disso, a natureza isolada dos contêineres facilita a experimentação: estudantes podem testar diferentes configurações, modificar parâmetros de sistema e até mesmo simular falhas e condições adversas sem risco de comprometer o sistema operacional hospedeiro ou outros projetos. Essa liberdade de experimentação é fundamental para o aprendizado de conceitos como tolerância a falhas, sincronização, consistência e coordenação distribuída, tópicos centrais em Computação Distribuída que exigem prática iterativa para compreensão profunda. Assim, os contêineres funcionam como ponte entre o aprendizado acadêmico e as práticas da indústria, preparando estudantes não apenas conceitualmente, mas também com competências técnicas diretamente aplicáveis em ambientes profissionais (O'connor et al., 2024).

2.3.2 Máquinas Virtuais

Máquinas virtuais, ou *virtual machines* (VMs), constituem tecnologias de virtualização que permitem a execução de múltiplos sistemas operacionais completos e isolados sobre uma única máquina física, através de uma camada de abstração de hardware denominada *hypervisor* ou monitor de máquinas virtuais (Smith; Nair, 2005). Plataformas amplamente utilizadas como VirtualBox, VMware Workstation e Hyper-V possibilitam que estudantes e educadores criem, configurem e gerenciem ambientes computacionais virtualizados com sistemas operacionais distintos, como Linux, Windows, BSD, executando simultaneamente sem interferência mútua.

No contexto do ensino de Sistemas Operacionais e Computação Distribuída, as máquinas virtuais representam uma ferramenta pedagógica fundamental ao proporcionarem ambientes seguros e controlados para experimentação prática com conceitos complexos e potenci-

almente destrutivos. Segundo Laadan; Nieh; Viennot (Laadan; Nieh; Viennot, 2010), o uso de plataformas virtuais no ensino de sistemas operacionais permite que estudantes experimentem com modificações profundas de sistema, incluindo alterações no *kernel*, desenvolvimento de *device drivers* e configurações críticas de rede e segurança. Tais experimentos ocorrem sem risco de comprometer a estabilidade ou segurança do sistema operacional hospedeiro, eliminando barreiras significativas ao aprendizado prático e à experimentação ativa.

Os benefícios pedagógicos das máquinas virtuais para o ensino de Computação Paralela e Distribuída são múltiplos e complementares. Em primeiro lugar, o isolamento completo proporcionado pela virtualização garante que experimentos potencialmente problemáticos, como simulação de falhas de sistema, ataques de segurança, esgotamento de recursos ou configurações incorretas de rede, permaneçam confinados ao ambiente virtual, protegendo tanto o sistema hospedeiro quanto outros ambientes virtuais concorrentes (Laadan; Nieh; Viennot, 2010). Adicionalmente, as máquinas virtuais possibilitam que estudantes executem e experimentem com diferentes sistemas operacionais e distribuições Linux sem necessidade de múltiplas máquinas físicas ou procedimentos complexos de *dual-boot*, facilitando o aprendizado comparativo de arquiteturas de sistemas distintas e suas implicações para sistemas distribuídos. Neste sentido, VMs permitem a simulação de ambientes distribuídos autênticos através da criação de múltiplas instâncias virtuais que comunicam-se via redes virtuais, possibilitando que estudantes implementem e testem arquiteturas de sistemas distribuídos, incluindo clusters, replicação de servidores, balanceamento de carga e protocolos de consenso, em seus próprios computadores. Outro benefício significativo é a capacidade de criar *snapshots* (instantâneos) do estado completo de uma máquina virtual, permitindo experimentação iterativa e exploratória: estudantes podem salvar o estado de um sistema funcional, realizar modificações experimentais, e reverter instantaneamente ao estado anterior caso necessário, eliminando o medo de erros irreversíveis e incentivando a exploração ativa de conceitos (Palicherla; Zhang; Porter, 2015). Por fim, a otimização de recursos físicos através da virtualização permite que laboratórios educacionais executem dezenas de máquinas virtuais em servidores físicos compartilhados, democratizando o acesso a ambientes computacionais variados mesmo em instituições com recursos limitados.

As aplicações práticas de máquinas virtuais no ensino de Computação Paralela e Distribuída são vastas e pedagogicamente significativas. Em cursos de Sistemas Operacionais, estudantes utilizam VMs para realizar atividades que seriam impraticáveis em sistemas físicos: instalação e configuração de múltiplas distribuições Linux e sistemas operacionais especializados, modificação de código do *kernel* para implementar novos algoritmos de escalonamento ou sistemas de arquivos, desenvolvimento de *device drivers* e módulos do kernel, e experimentação com configurações de segurança, firewall e políticas de acesso. Em Computação Distribuída especificamente, máquinas virtuais permitem a criação de laboratórios de redes virtuais onde estudantes simulam topologias complexas, como redes *mesh*, arquiteturas cliente-servidor com múltiplos níveis e sistemas de armazenamento distribuído, e experimentam com protocolos de comunicação, algoritmos de consenso como Paxos e Raft, e sistemas de coordenação distri-

buída. A escalabilidade proporcionada pela virtualização facilita o estudo de conceitos como elasticidade, balanceamento dinâmico de carga e tolerância a falhas através de replicação: estudantes podem inicializar rapidamente múltiplas réplicas de servidores virtuais, simular falhas de nós individuais, e observar comportamentos de recuperação e reconfiguração automática. Além disso, ferramentas modernas de gerenciamento de infraestrutura virtualizada, como Vagrant, que automatiza a criação e provisionamento de ambientes virtuais através de arquivos de configuração declarativos, aproximam estudantes de práticas profissionais de *Infrastructure as Code* e *DevOps*, preparando-os não apenas conceitualmente mas também com competências técnicas diretamente aplicáveis em ambientes de desenvolvimento e produção reais. Assim, as máquinas virtuais servem como ponte essencial entre teoria e prática, permitindo que conceitos abstratos de sistemas operacionais e computação paralela e distribuída sejam experimentados, testados e compreendidos através de implementações e simulações concretas em ambientes seguros e pedagogicamente eficazes.

2.3.3 *Scripts*

Scripts constituem arquivos de texto contendo sequências de instruções que são interpretadas e executadas automaticamente por um sistema computacional, representando uma forma fundamental de automação programável que abstrai operações complexas em comandos declarativos e reproduzíveis (AutomatedLab Project, 2024). Diferentemente de programas compilados, que passam por uma etapa de tradução para código de máquina antes da execução, *scripts* são processados por interpretadores em tempo de execução, conferindo-lhes características distintivas: flexibilidade para modificação rápida, execução imediata sem etapas intermediárias de compilação, e portabilidade entre sistemas que disponham do interpretador adequado. Esta natureza interpretada torna os *scripts* particularmente adequados para tarefas de configuração, automação de processos repetitivos, e orquestração de sistemas, atividades centrais na administração de infraestrutura computacional e, consequentemente, no ensino de disciplinas que envolvem sistemas complexos. Linguagens de *script* variam desde shells de linha de comando até linguagens de propósito geral com capacidades de *scripting*, cada uma oferecendo diferentes níveis de abstração e integração com o sistema operacional hospedeiro.

No contexto da Computação Paralela e Distribuída, *scripts* desempenham papel fundamental ao permitir a automação de tarefas como configuração de ambientes de desenvolvimento, provisionamento de máquinas virtuais e contêineres, estabelecimento de topologias de rede, e inicialização coordenada de múltiplos processos distribuídos (Arroyo, 2013). Os benefícios pedagógicos da utilização de *scripts* no ensino de Computação são múltiplos e substanciais. À semelhança dos contêineres (Subseção 2.3.1), *scripts* contribuem para a consistência ambiental entre estudantes ao automatizar a configuração de cada experimento, eliminando variações que poderiam comprometer atividades práticas (Malan, 2022). Além dessa contribuição compartilhada, *scripts* proporcionam eficiência significativa: processos de configuração que manualmente consumiriam horas podem ser completados em minutos, maximizando o tempo disponí-

vel para atividades de aprendizado efetivo e permitindo que aulas iniciem imediatamente com exercícios práticos. A reprodutibilidade constitui outro benefício fundamental: ambientes podem ser recriados de forma confiável entre diferentes turmas, semestres ou instituições, permitindo que materiais pedagógicos sejam compartilhados, versionados e reutilizados com garantia de funcionamento, prática alinhada ao paradigma de infraestrutura como código (*Infrastructure as Code*). A redução de erros é igualmente importante: ao substituir configurações manuais propensas a falhas humanas por procedimentos automatizados e testados, *scripts* aumentam a confiabilidade dos ambientes de laboratório (Ellis et al., 2019). A natureza programática dos *scripts* também permite customização pedagógica: docentes podem adaptar configurações para diferentes exercícios ou níveis de dificuldade, enquanto estudantes avançados podem modificar *scripts* existentes como parte de atividades de aprendizado, desenvolvendo simultaneamente competências de automação valorizadas profissionalmente. Crucialmente, *scripts* facilitam a restauração rápida de ambientes após experimentos destrutivos, característica essencial em disciplinas onde estudantes precisam testar cenários de falha e condições adversas sem receio de comprometer permanentemente o ambiente de trabalho. Ao reduzir a barreira técnica inicial, *scripts* permitem que estudantes concentrem-se no aprendizado conceitual, alinhando-se com princípios pedagógicos modernos que enfatizam a experimentação ativa como motor fundamental do aprendizado (Berbel, 2011).

2.3.4 Outras Tecnologias

O conjunto de tecnologias aplicáveis ao ensino de Computação Paralela e Distribuída estende-se muito além de contêineres, máquinas virtuais e *scripts*. A Infraestrutura como Código (*Infrastructure as Code* - IaC) permite definir e gerenciar ambientes computacionais através de arquivos de configuração declarativos e versionáveis, utilizando ferramentas como Terraform, Ansible e Puppet para provisionar automaticamente laboratórios completos. A Computação em Nuvem (*Cloud Computing*) oferece infraestrutura escalável e sob demanda através de provedores como Amazon Web Services, Google Cloud Platform e Microsoft Azure, possibilitando que estudantes experimentem com arquiteturas distribuídas autênticas, como *clusters* de máquinas virtuais, serviços de armazenamento distribuído e orquestração de contêineres com Kubernetes, sem investimento em hardware físico. A Inteligência Artificial (*Artificial Intelligence* - AI) tem emergido como ferramenta de apoio ao ensino através de sistemas de tutoria inteligente, assistentes de código baseados em modelos de linguagem de grande escala (*Large Language Models* - LLMs) e ferramentas de *feedback* automatizado. O foco nas três tecnologias detalhadas anteriormente justifica-se pelo critério operacional de baixa barreira de entrada: executam localmente, sem dependência de conta em provedor externo, custo financeiro recorrente ou acesso à Internet durante a atividade.

3 TRABALHOS RELACIONADOS

A literatura sobre ensino de programação paralela e sistemas distribuídos apresenta contribuições relevantes que fundamentam e contextualizam as propostas deste trabalho. Este capítulo analisa trabalhos relacionados organizados em quatro eixos temáticos: (i) abordagens para ensino de programação paralela a iniciantes; (ii) aplicação de metodologias ativas baseadas em projetos; (iii) infraestrutura e ferramentas de apoio ao ensino; e (iv) *design* de exercícios práticos para iniciantes. Para cada trabalho, discutem-se os objetivos, métodos, resultados principais e suas implicações para a proposta apresentada nesta monografia.

A seleção dos trabalhos analisados priorizou estudos que: (a) aplicam metodologias de ensino ativas no contexto de programação paralela ou distribuída; (b) investigam a viabilidade de introduzir esses tópicos para estudantes sem conhecimento prévio avançado; (c) propõem ou avaliam infraestruturas técnicas que reduzem barreiras de configuração; ou (d) oferecem *frameworks* para design e avaliação de atividades práticas. A análise crítica desses trabalhos permite identificar tanto validações empíricas para as escolhas metodológicas deste TCC quanto lacunas que justificam as contribuições propostas.

O levantamento foi conduzido na forma de **revisão exploratória** da literatura, visando delinear o estado da arte de forma suficiente para fundamentar as escolhas do arcabouço proposto, sem pretensão de exaustividade. As consultas foram realizadas nas bases Google Scholar e IEEE Xplore, complementadas por anais do *ACM Special Interest Group on Computer Science Education* (SIGCSE) para captar produção específica de *Computing Education Research*, combinando termos em inglês, como "*teaching distributed systems*", "*distributed computing education*", "*parallel programming education*", "*active learning + computer science*", "*flipped classroom + distributed systems*", "*scaffolding + programming education*" e "*Docker + teaching*", e seus equivalentes em português (por exemplo, "ensino de sistemas distribuídos", "metodologias ativas + computação", "sala de aula invertida + programação"). A triagem priorizou trabalhos com avaliação empírica em contexto de graduação e publicações em fóruns reconhecidos da área. Cabe ressaltar que, em função do escopo deste trabalho, **não foi conduzida uma revisão sistemática nem um mapeamento sistemático da literatura**: protocolos formais de busca exaustiva, extração padronizada de dados e avaliação de viés ficam fora dos objetivos desta monografia, sendo a abordagem aqui adotada deliberadamente exploratória.

3.1 ENSINO DE PROGRAMAÇÃO PARALELA PARA INICIANTES

Conte et al. (Conte et al., 2020) investigam a hipótese de que estudantes sem conhecimento prévio em computação podem aprender programação paralela com nível de qualidade similar ao de estudantes avançados. O estudo fundamenta-se na observação de que o ensino de programação paralela é tradicionalmente adiado para fases avançadas da graduação, assumindo pré-requisitos como sistemas operacionais e arquitetura de computadores. Essa postergação, argumentam os autores, dificulta que futuros profissionais desenvolvam o "pensamento paralelo"

naturalmente e o apliquem em outras disciplinas.

A metodologia do estudo envolveu cinco experimentos com um total de 252 alunos, divididos em dois grupos: 88 estudantes sem conhecimento prévio em computação (calouros ou de outras áreas) e 164 com conhecimento prévio (alunos avançados de Ciência da Computação, Engenharia de Computação e pós-graduação). As intervenções variaram em duração (de 8 a 45 horas), tecnologias abordadas (OpenMP e Pthreads) e metodologias de ensino empregadas, incluindo abordagem tradicional, Aprendizagem Baseada em Problemas (*Problem-Based Learning* – PBL) e Aprendizagem Baseada em Equipe (*Team-Based Learning* – TBL).

Os resultados demonstraram que estudantes sem conhecimento prévio em computação podem aprender programação paralela e atingir desempenho análogo ou até superior ao de alunos experientes. Nos experimentos onde ambos os grupos participaram das mesmas turmas, os resultados foram estatisticamente equivalentes: estudantes iniciantes obtiveram média de 93,3% nas atividades práticas, enquanto alunos com conhecimento prévio obtiveram 87,9%. Pré e pós-testes demonstraram evolução significativa no conhecimento teórico: no Experimento IV, alguns alunos evoluíram de 0% no pré-teste para 100% no pós-teste. A aplicação de TBL elevou o conhecimento conceitual de uma média de 75,4% (avaliação individual) para 97,2% (avaliação em equipe), evidenciando o valor da discussão colaborativa para consolidação de conceitos.

A principal contribuição deste trabalho para o presente TCC reside na validação empírica do uso de metodologias ativas, especificamente TBL e PBL, para ensino de tópicos tradicionalmente considerados avançados. A abordagem de ensino em camadas proposta pelos autores, que abstrai conceitos mais profundos de hardware e sistema operacional para permitir foco na lógica de programação paralela, alinha-se diretamente com a proposta deste trabalho de utilizar tecnologias educacionais (contêineres, *scripts*) como "camada habilitadora" que abstrai complexidades de infraestrutura. Entretanto, o foco do trabalho em programação paralela de memória compartilhada (*threads* em um único processo) não aborda os desafios específicos de sistemas distribuídos, como comunicação em rede, falhas parciais, consistência, que constituem o escopo central deste TCC. Além disso, o trabalho não explora a infraestrutura técnica utilizada nos experimentos, aspecto que este TCC aborda como pilar complementar.

3.2 APRENDIZAGEM BASEADA EM PROJETOS COM *SCAFFOLDING*

Manoharan; Ye (Manoharan; Ye, 2023) relatam experiência de aplicação de Aprendizagem Baseada em Projetos para ensinar programação paralela introdutória. O projeto centraliza-se na construção de um servidor de jogos *online* para dois jogadores, utilizado como ponto focal motivador para engajar os estudantes e contextualizar conceitos de programação concorrente, como *threads*, seções críticas e sincronização.

A abordagem foi aplicada a uma turma de 28 alunos em um curso introdutório de programação paralela. Os estudantes desenvolveram um servidor de jogo *multithreaded* em C#, utilizando *sockets* síncronos e comunicação via HTTP REST, sem uso de APIs de alto

nível. O servidor genérico permitiu que estudantes concentrassem seus esforços de design e desenvolvimento exclusivamente nos aspectos de *multithreading*, enquanto escolhiam jogos de seu interesse (Xadrez, Damas, Go, Jogo da Velha) para implementar como clientes.

Diante de dificuldades iniciais, a média da turma na primeira submissão foi de somente 48%, os instrutores forneceram um plano de desenvolvimento iterativo (*scaffolding*) estruturado em seis passos progressivos, começando com um servidor síncrono que atende um único cliente e evoluindo para um servidor *multithreaded* que gerencia desconexões. Além disso, realizaram sessões de *feedback* individualizadas antes de permitir uma segunda submissão. Essa intervenção elevou a nota média de 48% para 76%, com ganho médio de 28% (4,2 pontos em 15). O *feedback* informal indicou que os alunos apreciaram a abordagem e sentiram-se motivados pela possibilidade de desenvolver jogos completos.

Este trabalho oferece duas contribuições importantes para o presente TCC. Primeiro, valida a eficácia da Aprendizagem Baseada em Projetos quando combinada com *scaffolding* estruturado, demonstrando que o suporte progressivo é crucial para ajudar estudantes a superar dificuldades com habilidades pré-requisito. Os autores identificaram que "as habilidades-chave que uma grande proporção de estudantes não possuía são desenvolvimento incremental e estratégias eficazes de teste", conhecimentos que, embora não sejam o foco da disciplina, são essenciais para o sucesso em projetos práticos. Segundo, o projeto do servidor de jogos constitui excelente ponto de partida para extensão em direção a sistemas distribuídos. A replicação do estado do jogo em múltiplos nós, por exemplo, introduziria naturalmente conceitos de consistência e eleição de líder.

As limitações do estudo incluem o foco exclusivo em programação paralela (*multithreading* em uma única máquina), sem abordar os desafios de comunicação em rede entre processos distintos. Além disso, o *feedback* individualizado, embora eficaz, apresenta desafios de escalabilidade para turmas maiores. Este TCC propõe mitigar essa limitação por meio de estratégias como Programação em Pares e *Coding Dojo*, que distribuem a carga de mentoria entre os próprios estudantes.

3.3 INFRAESTRUTURA E FERRAMENTAS DE APOIO AO ENSINO

A literatura reconhece que a complexidade de configuração de ambientes constitui barreira significativa ao ensino efetivo de sistemas distribuídos. Ellis et al. (Ellis et al., 2019) abordam diretamente os desafios de ensinar sistemas distribuídos rigorosos, destacando a dificuldade de iniciantes em produzir código correto. Os autores propõem o uso de verificação de modelos (*model checking*) eficiente como ferramenta pedagógica, permitindo que estudantes verifiquem propriedades de correção de suas implementações automaticamente. Embora técnicas como *model checking* sejam poderosas, podem ser onerosas em contextos educacionais que não dispõem de tempo curricular para introduzir formalismos adicionais. Este TCC complementa essa abordagem ao propor infraestrutura que reduz a barreira de entrada sem exigir conhecimento de técnicas formais avançadas.

No contexto de padronização de ambientes de programação, Malan (Malan, 2022) descrevem a adoção de contêineres Docker e GitHub Codespaces no curso CS50 da Universidade de Harvard. A motivação central foi eliminar inconsistências de ambiente que historicamente consumiam tempo significativo de suporte técnico e desviavam o foco pedagógico dos conceitos fundamentais para questões de configuração. A solução permitiu que todos os estudantes trabalhassem em ambientes idênticos, independentemente de seus sistemas operacionais hospedeiros, democratizando o acesso e garantindo reprodutibilidade.

Complementarmente, Laadan; Nieh; Viennot (Laadan; Nieh; Viennot, 2010) propõem o uso de *virtual appliances* (máquinas virtuais pré-configuradas) e controle de versão distribuído para ensino de sistemas operacionais. A combinação permite que estudantes experimentem modificações em nível de *kernel* sem comprometer seus sistemas pessoais, enquanto o controle de versão facilita a distribuição de atualizações e a colaboração. Os autores argumentam que a virtualização reduz significativamente o tempo de configuração inicial, permitindo foco imediato em atividades de aprendizagem de alto valor.

Esses trabalhos fundamentam a escolha tecnológica deste TCC de utilizar contêineres Docker e, quando necessário, máquinas virtuais como base da infraestrutura padronizada. A contribuição específica deste trabalho está na articulação intencional dessas tecnologias com metodologias e estratégias de ensino ativas, aplicadas especificamente ao contexto de Computação Distribuída, combinação não explorada pelos trabalhos citados.

3.4 ELABORAÇÃO DE EXERCÍCIOS PRÁTICOS PARA INICIANTES

Wang (Wang, 2023) desenvolve um *framework* para projetar e avaliar exercícios de programação paralela amigáveis para iniciantes. O trabalho parte da premissa de que a maioria dos cursos de graduação introduz programação paralela tardiamente, dificultando a naturalização do "pensamento paralelo" nos estudantes. O autor propõe seis exercícios em Java, utilizando a classe `Thread`, com diferentes níveis de dificuldade focados em paralelismo de dados e de tarefas.

A avaliação dos exercícios combinou métricas objetivas e subjetivas. A análise de desempenho mensurou *speedup* e escalabilidade executando os programas em diferentes plataformas (Windows/AMD e macOS/Intel). A avaliação subjetiva utilizou questionário com escalas de 0 a 10 para os critérios de interesse, dificuldade e facilidade de verificação. Os resultados revelaram que exercícios com contexto de aplicação no mundo real, como Criptografia de Código Morse (7,52/10) e Renderização de Mandelbrot (7,05/10), foram considerados mais interessantes pelos estudantes do que exercícios puramente computacionais como Processamento de *Arrays* (3,0/10). A visualização dinâmica do progresso no exercício Mandelbrot mostrou-se particularmente eficaz para tornar o conceito de paralelismo mais concreto e intuitivo.

A principal contribuição deste trabalho para o presente TCC é o *framework* de avaliação que combina métricas de desempenho com *feedback* subjetivo dos estudantes. Essa abordagem será adaptada para avaliar as atividades de Computação Distribuída propostas, incorpo-

rando critérios específicos como latência de comunicação, comportamento sob falhas parciais, e percepção de complexidade de configuração. Além disso, os problemas base desenvolvidos (multiplicação de matrizes, *Scrabble*, *Mandelbrot*) constituem pontos de partida para criação de versões distribuídas que introduzam os desafios de comunicação em rede e coordenação entre processos.

As limitações do trabalho incluem o foco exclusivo em Java e na classe `Thread`, sem explorar ferramentas de conteinerização que garantam portabilidade do ambiente. Além disso, o autor reconhece que os exercícios abordam somente conceitos básicos, sem cobrir tópicos como condições de corrida, *locks* ou sincronização avançada, lacunas que este TCC busca preencher ao propor atividades que progridem para cenários de sistemas distribuídos completos.

3.5 SÍNTESE E POSICIONAMENTO DESTE TRABALHO

A análise dos trabalhos relacionados revela convergências e lacunas que posicionam as contribuições deste TCC. Em termos de convergências, os estudos analisados validam empiricamente: (i) a viabilidade de introduzir tópicos de programação paralela para estudantes sem conhecimento prévio avançado, utilizando metodologias ativas como TBL e PBL (Conte et al., 2020); (ii) a eficácia de *scaffolding* estruturado e ciclos de *feedback* formativo para superar dificuldades com habilidades pré-requisito (Manoharan; Ye, 2023); (iii) a importância de infraestrutura padronizada (contêineres, VMs) para eliminar inconsistências de ambiente e maximizar tempo de aprendizagem efetiva (Malan, 2022; Laadan; Nieh; Viennot, 2010); e (iv) o maior engajamento de estudantes com exercícios que apresentam contexto de aplicação no mundo real (Wang, 2023).

Em termos de lacunas, identifica-se que os trabalhos focam predominantemente em programação paralela de memória compartilhada (*multithreading*), não abordando os desafios específicos de sistemas distribuídos: comunicação em rede entre processos distintos, falhas parciais, assincronia, modelos de consistência e algoritmos de consenso. Além disso, nenhum dos trabalhos analisados propõe articulação intencional entre as três dimensões do arcabouço (metodologia, estratégia e tecnologia) enunciado no Capítulo 1, dimensões tratadas isoladamente na literatura.

Este TCC diferencia-se ao propor um arcabouço didático-metodológico que integra explicitamente essas três dimensões, aplicando-as especificamente ao ensino de Computação Distribuída. A matriz conceitual apresentada no Capítulo 4 sistematiza essas correlações, fundamentada no referencial teórico do Capítulo 2 e nas evidências empíricas dos trabalhos relacionados aqui analisados.

4 PROPOSTAS DE ABORDAGENS PARA O ENSINO

Este capítulo apresenta propostas de abordagens pedagógicas que articulam as metodologias de ensino, estratégias e tecnologias educacionais apresentadas no Capítulo 2, aplicando-as aos desafios específicos do ensino de Computação Distribuída no contexto das disciplinas de Computação Distribuída (INE5418 e INE5625), Programação Paralela e Distribuída (INE5645), Programação Concorrente (INE5410) e Sistemas Operacionais (INE5412, INE5424 e INE5611), oferecidas nos cursos de Ciência da Computação e Sistemas de Informação da UFSC.

A hipótese central deste trabalho é que a combinação intencional e estruturada dessas três dimensões pode reduzir as barreiras identificadas na literatura e promover aprendizagem mais efetiva em Computação Distribuída.

As propostas aqui apresentadas concentram-se em quatro tópicos centrais que permeiam as disciplinas foco deste trabalho, cada um operacionalizado por uma das atividades práticas detalhadas na Seção 4.2:

- **Comunicação assíncrona entre processos**;
- **Sincronização de relógios em sistemas distribuídos**;
- **Tolerância a falhas e consenso distribuído**;
- **Consistência de réplicas**.

A organização deste capítulo segue uma progressão lógica: inicialmente, apresenta-se uma matriz conceitual que sistematiza as correlações entre metodologias, estratégias, tecnologias, tópicos de ensino e atividades propostas. Em seguida, detalham-se propostas concretas de atividades práticas que operacionalizam essas correlações, cada uma especificando objetivos pedagógicos, processos de implementação e conexões com o referencial teórico estabelecido. Por fim, descreve-se a arquitetura conceitual de um ambiente padronizado que viabiliza a execução dessas atividades, garantindo reprodutibilidade e reduzindo a barreira técnica de entrada. Os instrumentos de avaliação aplicados a essas atividades, junto aos resultados obtidos nas turmas da UFSC, são apresentados no Capítulo 5.

4.1 MATRIZ CONCEITUAL DE ABORDAGENS PEDAGÓGICAS

A construção de abordagens pedagógicas efetivas para o ensino de Computação Distribuída requer a articulação intencional entre diferentes dimensões do processo educacional. Esta seção apresenta uma matriz conceitual que sistematiza as correlações entre as metodologias de ensino apresentadas na Seção 2.1, as estratégias pedagógicas descritas na Seção 3.2, as tecnologias educacionais da Seção 2.3, e os quatro tópicos centrais que orientam as atividades práticas deste trabalho.

Em termos de metodologias de ensino, as quatro atividades utilizam a mesma combinação: **Sala de Aula Invertida** (Subseção 2.1.4), que transfere a exposição inicial dos conceitos para uma fase pré-atividade individual com múltiplos formatos de material (Lage; Platt; Treglia, 2000), e **Aprendizagem Baseada em Investigação** (Subseção 2.1.6), que estrutura o trabalho presencial em ciclos sucessivos de execução, observação, formulação de hipóteses e validação experimental, alinhados às fases de orientação, conceituação, investigação, conclusão e discussão propostas por Pedaste et al. (Pedaste et al., 2015).

Em termos de estratégias pedagógicas, as quatro atividades adotam o mesmo par de escolhas: **Andaime** (Subseção 2.2.1) como estratégia transversal, materializado pela progressão em três níveis, da execução observacional do sistema pronto (Nível 0) à modificação substantiva do código ou da topologia (Nível 2), e por escolhas tecnológicas que reduzem a complexidade inicial de cada domínio (Wood; Bruner; Ross, 1976; van de Pol; Volman; Beishuizen, 2010); e **Programação em Pares** (Subseção 2.2.3) como estratégia *opcional* e complementar, que, quando adotada, força a verbalização das hipóteses ao longo do ciclo investigativo (Williams et al., 2000).

A opção por manter fixa essa combinação de metodologias e estratégias nas quatro atividades, variando entre elas apenas o tópico e a tecnologia empregada, é deliberada e apoia-se em três justificativas. A primeira é a **adequação ao formato de aplicação previsto para o piloto**, composto por atividades pontuais de curta duração oferecidas em caráter extracurricular (Capítulo 5): a Sala de Aula Invertida desloca a exposição conceitual para a fase pré-atividade, e a Aprendizagem Baseada em Investigação estrutura sessões autocontidas de experimentação; em contraste, alternativas como a Aprendizagem Baseada em Projetos, a Aprendizagem Baseada em Equipes e a Instrução pelos Pares pressupõem janelas longas de desenvolvimento ou tempo de aula dedicado com mediação docente, indisponíveis nesse formato. A avaliação dessas alternativas em formato semestral pleno é registrada como trabalho futuro em Seção 6.3. A segunda é o **controle de variação**: com metodologia e estratégia constantes, as diferenças observadas entre as atividades tornam-se atribuíveis primordialmente ao tópico e à tecnologia, o que preserva a comparabilidade dos instrumentos de avaliação aplicados repetidamente ao longo do semestre (Capítulo 5). A terceira é a **consistência do andaime**: a progressão idêntica em três níveis em todas as atividades reduz o custo de reaprender a dinâmica de trabalho a cada nova prática, permitindo que o esforço cognitivo do estudante se concentre no conteúdo novo de cada tópico (van de Pol; Volman; Beishuizen, 2010).

Quanto às tecnologias educacionais, as Atividades 1, 2 e 4 utilizam **contêineres** Docker orquestrados via Docker Compose (Subseção 2.3.1), apoiados por ***scripts* de automação** (Subseção 2.3.3) que garantem reprodutibilidade ambiental e reduzem a barreira técnica de entrada (Merkel, 2014; Malan, 2022; AutomatedLab Project, 2024). A Atividade 3 constitui exceção e adota **máquinas virtuais** leves provisionadas via Multipass sobre KVM/QEMU (Subseção 2.3.2); a justificativa pedagógica é detalhada em Subseção 4.2.4.

A Tabela 1 sintetiza, em seis colunas, as correlações principais entre tópicos, a disciplina em que os tópicos são abordados, metodologias, estratégias, tecnologias e a atividade

prática que materializa cada combinação, indicando os alinhamentos adotados nas atividades propostas neste capítulo. É importante ressaltar que essas correlações não são exclusivas ou prescritivas. A coluna "Atividade" antecipa o detalhamento da Seção 4.2, em que cada combinação é descrita em termos de objetivo pedagógico, processo de implementação e conexão com o referencial teórico.

Tabela 1 – Correlações entre tópicos, disciplina, metodologias, estratégias, tecnologias e atividades propostas.

Tópico	**Disciplina**	**Metodologias**	**Estratégias**	**Tecnologias**	**Ativ.**
Comunicação assíncrona (*sockets* TCP)	INE5418	Sala Invertida, IBL	Andaimagem, Pair Prog. (opc.)	Contêineres, *Scripts*	A1
Comunicação assíncrona (memória compartilhada)	INE5418	Sala Invertida, IBL	Andaimagem, Pair Prog. (opc.)	Contêineres, *Scripts*	A2
Sincronização de relógios	INE5418	Sala Invertida, IBL	Andaimagem, Pair Prog. (opc.)	VMs (Multipass), *Scripts*	A3
Tolerância a falhas e consistência de réplicas	INE5418	Sala Invertida, IBL	Andaimagem, Pair Prog. (opc.)	Contêineres, *Scripts*	A4

Fonte: Elaborado pelo autor (2026).

A coluna "Disciplina" amarra cada tópico ao conteúdo curricular da disciplina de Computação Distribuída do curso de Ciência da Computação (INE5418), na qual o projeto piloto deste trabalho foi aplicado (Capítulo 5) e cujo material de aula ancora a fase pré-atividade das quatro atividades. Os quatro tópicos integram o núcleo programático dessa disciplina, que cobre comunicação entre processos, sincronização, coordenação e acordo, e confiabilidade de sistemas distribuídos. As demais disciplinas enumeradas na abertura deste capítulo abordam recortes correlatos dos mesmos tópicos e permanecem como alvo de aplicação futura do arcabouço (Seção 6.3); as disciplinas de Programação Concorrente (INE5410) e Sistemas Operacionais (INE5412, INE5424 e INE5611), em particular, fornecem os pré-requisitos conceituais de processos, *threads* e sincronização local sobre os quais os quatro tópicos se apoiam.

4.2 PROPOSTAS DE ATIVIDADES PRÁTICAS

Esta seção apresenta propostas concretas de atividades práticas que operacionalizam as correlações estabelecidas na matriz conceitual da Seção 4.1. Cada atividade especifica o tópico abordado, os objetivos pedagógicos, as tecnologias utilizadas, e uma descrição detalhada do processo de implementação. As propostas fundamentam-se no referencial teórico estabelecido no Capítulo 2 e destinam-se à aplicação nas disciplinas foco deste trabalho.

4.2.1 Estrutura comum às quatro atividades

As quatro atividades compartilham a combinação metodológica registrada na matriz conceitual da Seção 4.1: Sala de Aula Invertida e Aprendizagem Baseada em Investigação como metodologias, Andaimagem em três níveis (Nível 0, executar o sistema pronto; Nível 1, inspecionar e responder questões conceituais; Nível 2, modificar código ou topologia) como estratégia transversal e Programação em Pares como estratégia opcional. O ciclo investigativo encerra-se no relatório entregue ao final de cada atividade, cuja seção "Dúvida para a próxima aula" alimenta o encontro subsequente e fecha o ciclo da Sala de Aula Invertida.

As subseções a seguir descrevem, para cada atividade, o tópico abordado, os objetivos pedagógicos, as tecnologias específicas e o conteúdo concreto dos três níveis de andaimagem. Desvios em relação a esse padrão comum, como a escolha de máquinas virtuais em vez de contêineres na Atividade 3, são justificados localmente.

4.2.2 Atividade 1: Cliente/Servidor Eco com Sockets TCP

Tópico:[1] Comunicação entre processos via *sockets* TCP (introdução à comunicação distribuída).

Objetivo Pedagógico: Desenvolver compreensão prática sobre o fluxo de uma conexão TCP cliente/servidor, capacitando o estudante a (1) explicar o papel de cada chamada de *socket* no servidor (`socket`, `bind`, `listen`, `accept`) e no cliente (`socket`, `connect`, `send`, `recv`); (2) justificar por que dois processos em contêineres distintos precisam de endereço e porta para se comunicar, mesmo executando no mesmo computador hospedeiro; e (3) modificar um servidor simples e reconstruir a imagem Docker correspondente para validar a alteração. Estes objetivos estabelecem a base conceitual mínima sobre comunicação entre processos que sustentará atividades posteriores envolvendo sincronização, consistência e tolerância a falhas em sistemas distribuídos.

Material pré-atividade: além da disciplina INE5418, o estudante selecionou dois vídeos curtos sobre *sockets*, documentação oficial de *sockets* em Python e material de aula em PDF.

Tecnologias: contêineres Docker orquestrados via Docker Compose (Subseção 2.3.1) representam cliente e servidor como dois processos em contêineres distintos comunicando-se através de uma rede interna nomeada, tornando concreta a separação entre processos comunicantes mesmo executando em um único hospedeiro (Merkel, 2014; Malan, 2022). A reprodutibilidade vem do `docker compose up -build` (Subseção 2.3.3).

Descrição do Processo: no *Nível 0* (orientação), os alunos clonam o repositório e executam `docker compose up -build`, observando que os *logs* de cliente e servidor aparecem intercalados, evidência concreta de dois processos paralelos compartilhando uma rede. No *Nível 1* (conceituação e investigação inicial), os estudantes inspecionam `servidor.py` e

[1] Repositório: https://github.com/jcconnects/atividade-sockets-eco.

`cliente.py` lado a lado e respondem por escrito, no relatório, questões sobre o papel de cada chamada de *socket* (`socket`, `bind`, `listen`, `accept`, `recv`, `sendall`, `connect`, `close`). Hipóteses são formuladas e testadas experimentalmente. Por exemplo, remover `depends_on` no `docker-compose.yml` e observar a falha de conexão quando o cliente inicia antes do servidor estar escutando. No *Nível 2* (investigação aprofundada), os estudantes modificam o servidor para devolver a mensagem em maiúsculas e expor um contador acumulado de mensagens, reconstruindo a imagem com `-build` e validando a alteração nos *logs*.

Pelo escopo restrito da atividade, a Programação em Pares é particularmente dispensável aqui: cada estudante pode conduzi-la individualmente sem prejuízo dos objetivos pedagógicos, com a escrita do relatório assumindo o papel de explicitação do raciocínio.

4.2.3 Atividade 2: Espaço de Tuplas com Apache River

Tópico:[2] Comunicação assíncrona entre processos distribuídos por meio de memória associativa compartilhada, com ênfase em desacoplamento espacial e temporal.

Objetivo Pedagógico: Desenvolver compreensão prática sobre o modelo de coordenação por espaço de tuplas, capacitando o estudante a (1) explicar as três operações fundamentais do modelo Linda, isto é, `write` (`OUT`), `take` (`IN`) e `read` (`RD`), e distinguir leitura destrutiva de não-destrutiva, bem como o comportamento bloqueante de `take` e `read` quando não há tupla correspondente; (2) identificar o desacoplamento espacial (produtor e consumidor não se referenciam diretamente) e o desacoplamento temporal (produtor e consumidor não precisam estar ativos simultaneamente) a partir da observação direta dos *logs* do sistema; (3) reconhecer o papel de um serviço de descoberta (`reggie`) na coordenação de componentes distribuídos sem endereços fixos; e (4) conectar o modelo de espaço de tuplas às abstrações modernas de mensageria (RabbitMQ, Kafka) e de registro de serviços, reconhecendo continuidades conceituais e diferenças de escala. Estes objetivos consolidam, a noção de comunicação indireta mediada por um repositório compartilhado, preparando o estudante para tópicos posteriores envolvendo consistência e tolerância a falhas em sistemas com múltiplos consumidores.

Material pré-atividade: slides da disciplina INE5418 sobre modelo Linda e documento de contexto teórico contendo linhagem histórica de Linda, JavaSpaces e Apache River, detalhamento das três operações fundamentais e conexão com sistemas modernos de mensageria.

Tecnologias: contêineres Docker orquestrados via Docker Compose (Subseção 2.3.1) representam cinco serviços distintos (`reggie` para descoberta Jini, `javaspaces` como servidor Outrigger do espaço de tuplas, `produtor`, `consumidor` e `monitor`) em uma rede interna nomeada, com dependências de inicialização explícitas que garantem que o serviço de descoberta esteja disponível antes dos demais. Essa orquestração torna concreta a complexidade real de um *middleware* de coordenação distribuída, em que múltiplos processos coordenados não podem ser substituídos por uma instalação local simples (Merkel, 2014; Malan, 2022). O Apache

[2] Repositório: https://github.com/jcconnects/atividade-espaco-de-tuplas.

River, sucessor direto do JavaSpaces da Sun Microsystems e mantido pela Apache Software Foundation, oferece a única implementação de nível de produção de espaço de tuplas executável de forma reproduzível, e todas as dependências (JDK 8, classes do River, configurações de *policy* de segurança) ficam encapsuladas nas imagens, eliminando instalação local de Java. A flag `-scale consumidor=2` habilita o experimento de escalabilidade horizontal sem qualquer alteração no código.

Descrição do Processo: no *Nível 0* (orientação), os estudantes executam `docker compose up -build` e observam a ordem de inicialização nos *logs*: `reggie` sobe primeiro, `javaspaces` registra-se nele, e somente então produtor e consumidor descobrem o espaço e iniciam a troca de tarefas. Um experimento subsequente de desacoplamento temporal (iniciar somente `reggie`, `javaspaces` e `produtor`, aguardar o término do produtor, encerrar o produtor e iniciar apenas o consumidor em um segundo momento) evidencia que as tuplas persistiram no espaço e foram consumidas posteriormente, comportamento impossível em comunicação síncrona direta via *sockets*. No *Nível 1* (conceituação e investigação inicial), os estudantes preenchem uma tabela comparativa das três operações (bloqueio, alteração de estado, equivalente Linda) e conduzem experimentos guiados: derrubar o `reggie` com serviços já conectados (observando que clientes ativos continuam funcionando, porém novos clientes não conseguem descobrir o espaço), iniciar o consumidor antes do produtor (observando o bloqueio de `take` até a primeira tupla ser depositada) e executar com `-scale consumidor=2` (observando que cada tarefa é consumida por exatamente um consumidor, sem necessidade de coordenação explícita entre eles). Cada experimento exige formulação de hipótese, observação e registro escrito de conclusões. No *Nível 2* (investigação aprofundada), a Modificação A é guiada (alterar `consumidor/config.properties` para ativar busca por alta prioridade e validar nos *logs*), enquanto a Modificação B é aberta: preencher a implementação do serviço `monitor` escolhendo entre `read` e `take` (justificando por que `take` seria problemático ao remover tuplas que outros consumidores precisam) e descomentar o serviço no `docker-compose.yml`. A inexistência de operação `count` nativa força reflexão sobre limitações do modelo puro e contraste com sistemas modernos como Redis (`LLEN`).

O relatório final inclui síntese que conecta o modelo de espaço de tuplas a sistemas modernos de mensageria (RabbitMQ, Kafka) e de registro de serviços (Consul, Kubernetes DNS), reconhecendo continuidades conceituais e diferenças de escala.

4.2.4 Atividade 3: Sincronização de Relógios com NTP/chrony

Tópico:[3] Sincronização de relógios físicos em sistemas distribuídos por meio do *Network Time Protocol* (NTP), com ênfase em convergência, estratégias de correção (*step* e *slew*) e métricas de qualidade da fonte de tempo (*offset*, RTT, dispersão, *stratum*).

Objetivo Pedagógico: Desenvolver compreensão prática sobre sincronização de relógios em sistemas distribuídos, capacitando o estudante a (1) explicar o significado de *offset*,

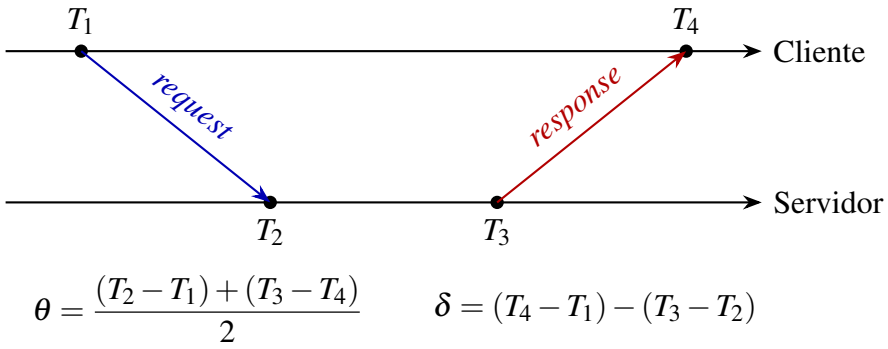
[3] Repositório: https://github.com/jcconnects/atividade-ntp-sincronizacao.

RTT, dispersão e *stratum* como métricas de qualidade da fonte de tempo, identificando os quatro instantes ($T1..T4$) que compõem o cálculo de *offset* segundo o algoritmo de Cristian (Cristian, 1989); (2) distinguir as duas estratégias de correção de relógio, *step* e *slew*, e justificar quando cada uma é apropriada em produção, considerando o *trade-off* entre velocidade de convergência e monotonicidade do relógio; (3) observar empiricamente a convergência de relógios desalinhados sob um protocolo de sincronização real, medindo o tempo dessa convergência em cenários distintos de drift inicial; e (4) reconhecer por que sistemas distribuídos modernos continuam investindo em sincronização de relógios, mesmo com NTP há quatro décadas no campo. Estes objetivos consolidam o entendimento de que abstrações temporais aparentemente simples (TTLs, *leases*, certificados, logs ordenados) dependem implicitamente de garantias de sincronização que precisam ser construídas explicitamente em sistemas distribuídos.

Material pré-atividade: slides da disciplina INE5418 sobre sincronização de relógios e documento de contexto teórico com linhagem histórica de NTP, detalhamento dos algoritmos de Cristian e Berkeley (Cristian, 1989; Gusella; Zatti, 1989), fórmula de *offset* a partir dos quatro instantes $T1..T4$, discussão sobre *slew* em comparação a *step* e conexão com o estado-da-arte (NTS, PTP, TrueTime, HLC) (Mills, 1985; Lichvar, 2024; Franke et al., 2020).

A Figura 2 ilustra os quatro instantes T_1, T_2, T_3, T_4 do algoritmo de Cristian, a partir dos quais o cliente calcula o *offset* em relação ao servidor de tempo e o tempo de ida e volta (RTT) da mensagem.

Figura 2 – Os quatro instantes (T_1, T_2, T_3, T_4) do algoritmo de Cristian e fórmulas de *offset* (θ) e RTT (δ).



Fonte: Adaptado de Cristian (Cristian, 1989).

Tecnologias: esta atividade rompe com o padrão das demais e adota **máquinas virtuais** leves provisionadas via Multipass (Subseção 2.3.2) em vez de contêineres. A justificativa é pedagogicamente deliberada: contêineres compartilham o relógio do *kernel* hospedeiro (os *namespaces* do Linux isolam rede, processos e montagens, mas não isolam o relógio), tornando impossível observar o desalinhamento que a atividade investiga. VMs sob KVM/QEMU possuem relógio independente mantido pelo `kvm-clock` ou pelo RTC emulado, permitindo dessincronização real entre nós (Smith; Nair, 2005; Laadan; Nieh; Viennot, 2010). O ambiente compreende quatro nós distintos: `ntp-servidor` (servidor chrony da sub-rede), `ntp-cliente-1` (drift de $+30$s com `makestep` ativo), `ntp-cliente-2` (drift de -45s com `makestep` ativo) e

`ntp-cliente-slew` (drift de +60s com `makestep` desabilitado, apenas para Parte B). *Scripts* de orquestração (Subseção 2.3.3) encapsulados em um `Makefile` com *targets* declarativos (`up`, `up-b`, `logs`, `plot`, `clean`, `purge`) automatizam provisionamento, configurações de chrony por papel, injeção controlada de *drift* via `timedatectl` e coleta dos CSVs produzidos por um *logger* instalado em cada VM via *cloud-init* (AutomatedLab Project, 2024). O padrão de *template* de VM, isto é, VM base com chrony e *logger* pré-instalados, clonada para cada papel, materializa o conceito de imagem base e reduz a primeira execução de cerca de quatro minutos para trinta segundos nas seguintes, evidenciando o paralelo com `docker pull` + `docker run`.

Descrição do Processo: no *Nível 0* (orientação), os estudantes executam `make up`, que provisiona o *template*, clona o servidor e os dois clientes, injeta os *drifts* de +30s e −45s e inicia o *logger* CSV em cada VM, e inspecionam o estado de cada cliente via `multipass exec ntp-cliente-1 - chronyc tracking`, observando que o servidor já aparece sincronizado enquanto os clientes ainda apresentam *offsets* elevados. Um momento didático intercalado pede que tentem alterar o relógio via duas vias distintas (`sudo date -s` e `sudo timedatectl set-time`) e registrem a mensagem exata de erro produzida pelo *systemd-timedated* quando há um serviço de sincronização ativo, revelando que o bloqueio é uma defesa em nível de *userspace* e não proteção do *kernel*. No *Nível 1* (conceituação e investigação inicial), os estudantes inspecionam `chrony-cliente.conf` e preenchem uma tabela que relaciona cada conceito do protocolo, endereço da fonte de tempo, convergência rápida no primeiro contato com `iburst`, fronteira entre *step* e *slew* via `makestep`, persistência de *drift* entre reinícios e destino dos *logs*, à linha exata do `.conf` que o implementa. Em seguida preenchem uma segunda tabela com os valores observados e o significado físico de cada campo de `chronyc tracking` (`Reference ID`, `Stratum`, `Last offset`, `RMS offset`, `Frequency`, `Root delay`, `Root dispersion`), por exemplo identificando os quatro instantes $T1..T4$ que compõem `Last offset` e explicando por que `Root dispersion` é cota superior do erro, não erro exato. Finalmente, examinam `/var/log/chrony/tracking.log` e extraem dois trechos, uma linha de *step* (correção grande e instantânea) e uma linha de *slew* (correção pequena e contínua), explicitando os critérios de distinção.

No *Nível 2* (investigação aprofundada), a atividade transita para experimentação em duas partes complementares. Na *Parte A* (convergência em três VMs), os estudantes aguardam pelo menos dois minutos de execução, coletam os CSVs via `make logs` e geram o gráfico `convergencia-parte-a.png` via `make plot-a`, respondendo no relatório quantos segundos foram necessários para o `Last offset` de cada cliente cair abaixo de 1 milissegundo e por que clientes com *drifts* iniciais simétricos (+30s e −45s) convergem em tempos parecidos, formulação que exige reconhecimento de que o mecanismo dominante nesse cenário é o *step* configurado por `makestep 1.0 3` e não o *slew*. Na *Parte B* (*slew* puro versus *step*+*slew*), acrescenta-se um quarto nó (`ntp-cliente-slew`) com `makestep` desabilitado e *drift* de +60s, e os estudantes aguardam ao menos cinco minutos para observar a lentidão do *slew* puro (limitado a cerca de 500 ppm no *kernel* Linux, equivalente a um segundo de correção a cada dois mil segundos reais). O gráfico comparativo `comparacao-parte-b.png` sobrepõe as curvas das quatro VMs, e o re-

latório exige que o estudante justifique, em produção, quando se escolheria *slew* puro mesmo sabendo de sua lentidão, pensando em sistemas que dependem de monotonicidade (TTLs, *leases*, logs ordenados, certificados), e em que situação `makestep` poderia ser perigoso. Na *Parte B.1* (*step* manual forçado), o estudante executa `multipass exec ntp-cliente-slew - sudo chronyc makestep` sobre o cliente que ainda corrige lentamente, observa o salto instantâneo no gráfico atualizado e reflete sobre a diferença entre política configurada (declarativa, no `.conf`) e comando administrativo (imperativo, via `chronyc`), evidenciando que o operador pode sobrepor a política persistente quando o contexto operacional o exige.

O relatório final inclui os dois gráficos de convergência e a reflexão sobre escolhas de *step* versus *slew* em diferentes contextos operacionais, alimentando o encontro subsequente com discussão sobre o estado-da-arte contemporâneo (NTS contra *spoofing*, PTP em aplicações de baixa latência, TrueTime e HLC em bancos de dados distribuídos).

4.2.5 Atividade 4: Consenso com Raft e Visualizador ao Vivo

Tópico:[4] Tolerância a falhas e consenso distribuído. A atividade aborda o problema de manter consistência de estado entre réplicas na presença de falhas de nó e partições de rede, explorando o algoritmo Raft através de um cluster real construído sobre a biblioteca `hashicorp/raft`.

Objetivo Pedagógico: Desenvolver compreensão prática sobre consenso distribuído via Raft, capacitando o estudante a (1) identificar os três papéis de um nó (*Follower*, *Candidate*, *Leader*), as cores correspondentes no visualizador e os gatilhos que causam transições entre eles; (2) explicar o conceito de *term* como relógio lógico simplificado que ordena eleições e justificar por que ele deve ser monotonicamente não-decrescente, construindo cenários em que a violação dessa propriedade quebraria a garantia de *Election Safety* (no máximo um líder por *term*); (3) reconhecer empiricamente a importância do quórum (maioria estrita) tanto para eleição de líder quanto para comprometimento de entradas, observando que em cluster de três nós duas confirmações bastam, que clusters de tamanho par não melhoram tolerância a falhas, e que partições minoritárias travam progresso sem comprometer segurança; e (4) observar diretamente as cinco garantias de segurança de Raft em ação, particularmente a *Leader Completeness*, materializada quando um líder antigo isolado em partição minoritária tem suas entradas não comprometidas sobrescritas pelo log do novo líder após reconciliação.

Material pré-atividade: slides da disciplina INE5418 sobre consenso e tolerância a falhas e documento de contexto teórico com problema do consenso, impossibilidade FLP, linhagem Paxos–Raft, tabela dos três papéis com cores e transições, definição de *term* como relógio lógico simplificado, duas fases do algoritmo (eleição e replicação) e enumeração das cinco garantias de segurança (*Election Safety*, *Leader Append-Only*, *Log Matching*, *Leader Completeness*, *State Machine Safety*).

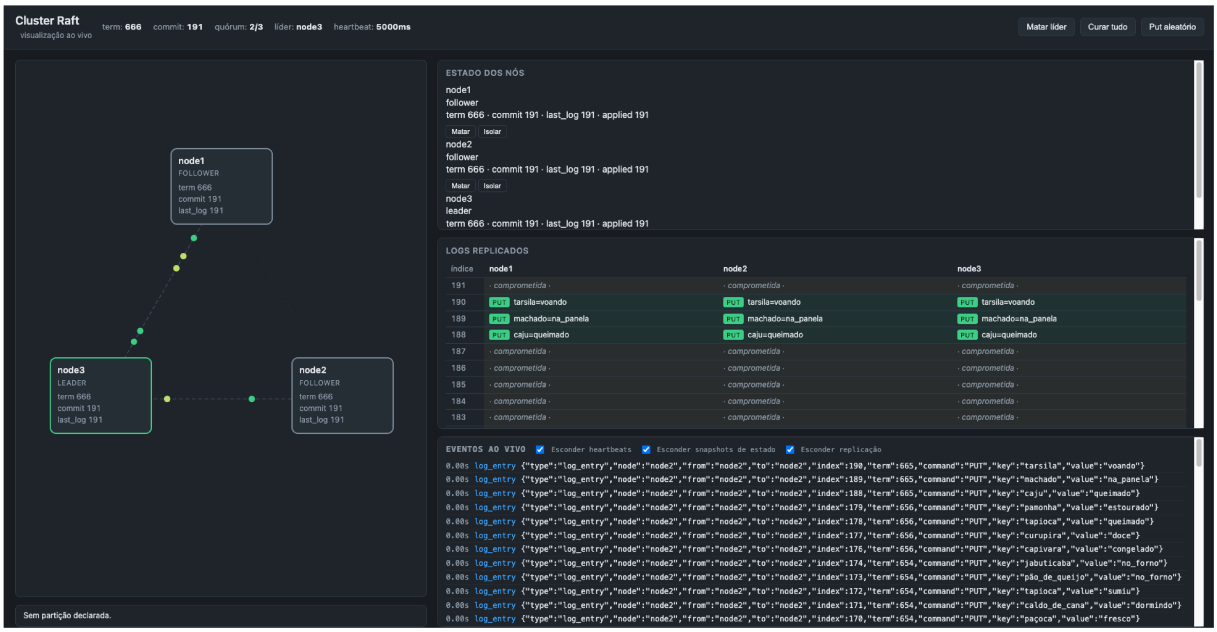
[4] Repositório: https://github.com/jcconnects/atividade-consenso-raft.

Tecnologias: contêineres Docker (Subseção 2.3.1) orquestrados via Docker Compose materializam um cluster Raft de três nós comunicantes em rede *bridge* isolada (`rede-raft`), acompanhado de dashboard web servido por nginx e *broker* Python auxiliar que traduz cliques do dashboard em ações de controle (`docker stop` para o `Matar`, regras `iptables` sobre porta Raft 7000 para `Aplicar partição` e `Curar tudo`). Cada nó executa um binário Go que combina três responsabilidades: a biblioteca `hashicorp/raft` (que implementa o protocolo na sua íntegra), uma máquina de estados que materializa um *key-value store* replicado e um *event tap* que publica via *Server-Sent Events* (SSE) cada transição de papel, cada incremento de *term*, cada `AppendEntries`, cada `RequestVote` e cada avanço de `commitIndex`. Como o foco pedagógico está no comportamento lógico do algoritmo, contêineres bastam e oferecem inicialização inferior a cinco segundos, permitindo iterações rápidas entre experimentos (Merkel, 2014). A variável `RAFT_HEARTBEAT_MS=5000` amplia o *heartbeat* típico de 150 ms para 5 s e o *election timeout* para 10 s, tornando cada transição humanamente observável. O dashboard renderiza os nós em **layout circular** ligados por **linhas tracejadas cinzas** que representam a topologia ativa (a linha some quando partição é detectada), e cada RPC aparece como **pacote animado**, círculo verde para `AppendEntries` e círculo azul para `RequestVote`, viajando do remetente ao destinatário. Os papéis são codificados por cor (cinza = *Follower*, amarelo = *Candidate*, verde = *Leader*, vermelho = `DISCONNECTED`), e o painel `Logs replicados` pulsa verde a célula correspondente quando uma entrada é aplicada em determinado nó. Os controles dividem-se em *ações globais no header* (`Matar líder`, `Curar tudo`, `Put aleatório`, `Aplicar partição (N marcados)`) e *ações por nó* nos *cards* (`Matar`, `Isolar`, que alterna para `Cancelar/Liberar`, e `Curar este nó`). Um *badge* `QUÓRUM ATINGIDO idx=N` aparece momentaneamente no *card* do líder quando ele contabiliza acks da maioria. *Scripts* de orquestração (Subseção 2.3.3), como `kill-leader.sh`, `partition.sh`, `heal.sh` e `put.sh`, fornecem o equivalente em linha de comando, permitindo reproduzir os experimentos sem o dashboard quando se quer inspecionar o canal de controle subjacente.

Descrição do Processo: no *Nível 0* (orientação), os estudantes executam `docker compose up -build`, abrem `http://localhost:8080` e observam o cluster passar pela sequência canônica em ritmo humano imposto por `RAFT_HEARTBEAT_MS=5000`: três nós cinzas em círculo ligados por linhas tracejadas cinzas, um deles vira amarelo após o primeiro *timeout* (incrementou *term*, iniciou eleição), pacotes azuis (`RequestVote`) partem do candidato, pacotes azuis de resposta retornam (votos concedidos), o candidato vira verde e pacotes verdes (`AppendEntries`) passam a pulsar continuamente do líder para os seguidores. Em seguida, o experimento *Nível 0b* torna concreto o mecanismo de recuperação automática: o estudante clica em `Matar líder` no *header*, anota o nó morto e o *term*, observa por alguns segundos os *followers* órfãos sem *heartbeat*, vê um deles virar amarelo (candidato), incrementar *term* e virar verde (novo líder); reanima o nó derrubado clicando em `Curar este nó`, observa-o voltar como *follower* recebendo `AppendEntries` do novo líder, e registra o tempo aproximado de *recovery* comparando-o à indisponibilidade total de um servidor único.

No *Nível 1* , cinco experimentos guiados produzem dados que alimentam respostas

Figura 3 – Visualizador ao vivo do cluster Raft: três nós em layout circular (node3 como líder em verde, node1 e node2 como *follower* em cinza), painel de logs replicados e painel de eventos ao vivo.



Fonte: Elaborado pelo autor (2026).

conceituais no relatório. No *1.1* (três papéis), o estudante preenche uma tabela relacionando cada papel à cor no dashboard, ao envio e recebimento de `AppendEntries` e ao gatilho de saída. No *1.2* (*term* monotônico), aciona `Matar líder` três vezes seguidas, anota a sequência de *term*s observados, e constrói um cenário hipotético em que *term* decrementável quebraria *Election Safety*. No *1.3* (quórum e comprometimento), clica em `Put aleatório` algumas vezes e acompanha a sequência: `last_log` sobe no *card* do líder (entrada anexada localmente, branca), pacotes verdes escuros partem do líder para cada seguidor (replicação), pacotes verde-amarelados retornam (ack), o *badge* `QUÓRUM ATINGIDO` pisca no líder no instante em que ele contabiliza maioria de acks, a célula do índice começa a pulsar verde no painel `Logs replicados` primeiro na coluna do líder e depois, no próximo *heartbeat* carregando `leader_commit` atualizado, na coluna de cada seguidor, materializando o papel do `commitIndex` propagado em `AppendEntries` subsequentes. No *1.4* (partição minoritária), o estudante clica em `Isolar` no *card* de um único nó (a borda vira tracejada vermelha), depois em `Aplicar partição (1 marcado)` no *header*; as linhas tracejadas que ligavam aquele nó aos demais desaparecem (o *broker* aplica regras `iptables` na porta 7000), e o nó isolado permanece visível porque sua porta de eventos 8100 não é bloqueada. Tenta uma escrita contra o nó isolado (que falha ou é redirecionada) e outra contra o líder vivo no lado majoritário (que sucede), e observa o *term* do nó isolado subir perpetuamente sem nunca virar verde, materializando a regra de quórum. No *1.5* (partição majoritária e reconciliação de log), o experimento mais importante do nível, o estudante isola o líder atual via `Isolar + Aplicar partição`, escreve duas entradas *zumbi* contra ele que permanecem brancas para sempre, observa o lado majoritário eleger novo líder em *term* maior, escreve uma entrada *real* contra esse novo líder, cura a partição via `Curar tudo`, e observa o log do líder

antigo convergir para o do novo líder ao receber `AppendEntries` com *term* maior, materializando diretamente a *Leader Completeness* (§5.2 do artigo de Ongaro e Ousterhout). Uma distinção conceitual reforçada ao longo do nível: *partição* (via `Isolar` + `Aplicar partição` ou `partition.sh`) mantém o processo do nó vivo e visível no dashboard, bloqueando apenas o tráfego Raft via `iptables`; *parada* (via `Matar`) derruba o processo todo, fazendo o nó aparecer vermelho como `DISCONNECTED`.

No *Nível 2* (modificação), o estudante edita o `docker-compose.yml` adicionando `node4` e `node5` no padrão dos existentes, atualiza `RAFT_PEERS` em todos os nós para listar os cinco endereços, ajusta `RAFT_NODES` nos serviços `dashboard` e `broker` para `"node1, node2,node3,node4,node5"`, replica `RAFT_HEARTBEAT_MS` nos novos nós, acrescenta volumes `node4-data` e `node5-data`, recompila com `docker compose down -v && docker compose up -build` e observa no dashboard cinco *cards* com o cabeçalho passando a indicar `quórum: 3/5`. Em seguida derruba dois nós (incluindo o líder) e confirma que o cluster sobrevive, derruba um terceiro e observa o cluster travar com candidatos amarelos cujo *term* sobe perpetuamente sem nunca atingir maioria (precisariam de três votos, restam apenas dois), materializando empiricamente a tolerância a $\lfloor (N-1)/2 \rfloor$ falhas e justificando por que tamanhos pares são desencorajados (seis nós toleram apenas duas falhas, igual a cinco, ao custo de mais mensagens). O relatório exige que o estudante compare a tolerância de três e cinco nós, justifique o desencorajamento de tamanhos pares, discuta o *trade-off* entre tamanho de cluster e latência de comprometimento, e descreva em que cenário operacional escolheria cinco nós em vez de três.

O relatório final inclui capturas de tela do *dashboard* nos momentos-chave (eleição inicial, partição majoritária com líder isolado, log do nó isolado após reconciliação, cluster de cinco nós com quórum esgotado) e reflexão sobre o uso real de Raft em sistemas como etcd, Consul e CockroachDB. A Programação em Pares é particularmente útil no experimento 1.5: isolar o líder, escrever no líder isolado, observar a eleição no lado majoritário, escrever no novo líder e curar a partição exige sequência precisa de ações sobre múltiplos *cards* em janela curta de tempo.

4.3 AMBIENTE PADRONIZADO: ARQUITETURA E PRINCÍPIOS

As atividades propostas na Seção 4.2 requerem infraestrutura técnica que viabilize sua execução prática, reduzindo barreiras de entrada e garantindo experiências consistentes entre estudantes. Esta seção apresenta a arquitetura conceitual de um ambiente padronizado que materializa os princípios de andaimagem, reprodutibilidade e portabilidade discutidos no Capítulo 2.

A concepção do ambiente fundamenta-se em três princípios norteadores. O primeiro, **Reprodutibilidade**, estabelece que todos os estudantes devem poder replicar ambientes idênticos independentemente de suas configurações locais, eliminando a barreira de heterogeneidade discutida em Subseção 2.3.1 (Malan, 2022). O segundo, **Portabilidade**, determina que o am-

biente deve funcionar em computadores com recursos variados, desde *laptops* modestos até estações de trabalho potentes, e em diferentes sistemas operacionais hospedeiros (Windows, macOS, Linux), democratizando o acesso e evitando que limitações de equipamento impeçam a participação plena de estudantes. O terceiro, **Baixa Barreira de Entrada**, estabelece que o tempo e esforço necessários para configurar e iniciar o ambiente devem ser minimizados, permitindo que estudantes concentrem-se rapidamente em atividades de aprendizagem de alto valor (Ellis et al., 2019).

A arquitetura do ambiente organiza-se em três camadas conceituais que materializam esses princípios. A **Camada de Virtualização** adota contêineres Docker (Subseção 2.3.1) nas Atividades 1, 2 e 4, e máquinas virtuais leves provisionadas via Multipass sobre KVM/QEMU (Subseção 2.3.2) na Atividade 3, isolando os processos comunicantes e provendo portabilidade entre sistemas operacionais hospedeiros. A **Camada de Orquestração** utiliza Docker Compose para os clusters de contêineres e um `Makefile` declarativo para o conjunto de VMs da Atividade 3, coordenando de forma declarativa a inicialização, a rede e a parametrização dos nós. A **Camada de Automação e Instrumentação** reúne *scripts* auxiliares, *loggers* embarcados e o *dashboard* web do Raft alimentado via *Server-Sent Events*, automatizando experimentos de falha, injeção de *drift* e visualização de estado. As escolhas concretas de cada camada para cada atividade, incluindo imagens, serviços, variáveis de ambiente e *scripts*, são detalhadas no parágrafo **Tecnologias** de cada atividade na Seção 4.2.

Um aspecto fundamental da arquitetura é a progressão no nível de andaimagem ao longo das atividades. As Atividades 1 e 2 fornecem ambientes altamente estruturados com suporte extensivo: imagens Docker pré-construídas, *scripts* de inicialização completos, código de referência funcional como ponto de partida. À medida que os estudantes desenvolvem competência, o suporte gradualmente se desvanece, princípio de *fading* do andaime (van de Pol; Volman; Beishuizen, 2010). No Nível 2 da Atividade 4, por exemplo, o estudante edita diretamente o `docker-compose.yml` para escalar o cluster de três para cinco nós, atualizando `RAFT_PEERS`, `RAFT_NODES` e adicionando volumes `node4-data` e `node5-data`, assumindo responsabilidade pela própria definição da topologia. Essa progressão materializa os princípios de andaimagem discutidos na Subseção 2.2.1, controlando inicialmente elementos além da capacidade dos aprendizes e gradualmente transferindo responsabilidade à medida que desenvolvem competência (Wood; Bruner; Ross, 1976).

A distribuição e versionamento do ambiente seguem práticas modernas de desenvolvimento de software. Todas as definições de ambiente (*Dockerfiles*, arquivos Docker Compose, *Makefile* e configurações de *cloud-init* para Multipass, *scripts* auxiliares e código-fonte das atividades) residem no repositório Git público de cada atividade, permitindo versionamento, rastreamento de mudanças e contribuições colaborativas. As imagens Docker são construídas localmente via `docker compose up -build` ou poderão, em iterações futuras, ser publicadas em registro público para eliminar tempos de *build* e permitir que estudantes iniciem ambientes através de comandos ainda mais sucintos. Documentação abrangente, incluindo guias de início rápido e descrição dos experimentos, é mantida como *markdown* junto ao código de cada

atividade, garantindo sincronização entre código e documentação. Esse modelo de distribuição alinha-se com práticas de código aberto e infraestrutura como código, preparando os estudantes para ambientes profissionais enquanto garante sustentabilidade e evolução contínua do ambiente ao longo de múltiplos semestres.

Este capítulo concentrou-se no *design* do arcabouço: a matriz conceitual que correlaciona metodologias, estratégias, tecnologias e tópicos de ensino; as quatro atividades práticas que operacionalizam essa matriz; e a arquitetura do ambiente padronizado que viabiliza sua execução. A aplicação empírica das atividades em turmas reais da disciplina de Computação Distribuída da UFSC, os dois instrumentos de avaliação efetivamente utilizados (questionário pós-atividade fundamentado em NASA-TLX, TAM e itens de Aprendizagem Percebida, e relatório reflexivo entregue ao final de cada atividade) e os resultados coletados em sala são apresentados no Capítulo 5.

5 RELATO DE EXPERIÊNCIA NO AMBIENTE DA UFSC

Este capítulo relata a aplicação piloto do arcabouço proposto no Capítulo 4 em uma turma real da disciplina de Computação Distribuída da UFSC.

5.1 OBJETIVO E CARACTERIZAÇÃO DO ESTUDO

O estudo empírico descrito neste capítulo operacionaliza o quinto objetivo específico enunciado no Capítulo 1 e persegue três objetivos concretos: (i) **verificar a viabilidade operacional** do arcabouço em condições reais de oferta, isto é, se o ambiente padronizado é executável pelos estudantes em seus próprios equipamentos, se os instrumentos de avaliação são aplicáveis no fluxo regular da disciplina e se os artefatos produzidos são analisáveis pelos procedimentos previstos; (ii) **coletar sinalizações preliminares** sobre aprendizagem percebida, usabilidade do ambiente e carga de trabalho das atividades, que orientem a calibragem de dificuldade e de suporte; e (iii) **identificar refinamentos** nas atividades e em seus materiais, alimentando o ciclo iterativo de aprimoramento alinhado à pesquisa baseada em design (*design-based research*) (The Design-Based Research Collective, 2003), conforme o delineamento apresentado em Seção 1.1.

5.2 FORMA DE AVALIAÇÃO

A avaliação das atividades práticas propostas neste trabalho apoia-se em dois instrumentos complementares aplicados a cada atividade: um **questionário pós-atividade** respondido individualmente e um **relatório reflexivo**. A combinação justifica-se pela complementaridade entre as fontes de evidência. O questionário captura a percepção subjetiva do estudante (aprendizagem percebida, usabilidade do ambiente, carga de trabalho), enquanto o relatório materializa, a compreensão conceitual efetivamente articulada após a experimentação. A triangulação entre dado autorreportado e artefato textual fortalece a validade das conclusões sobre a eficácia de cada atividade. A escolha por esses dois instrumentos, em detrimento de protocolos de avaliação mais extensos, justifica-se pela natureza pontual das atividades, aplicadas como práticas de curta duração que reforçam conteúdo teórico já apresentado em sala, e pela necessidade de instrumentos de baixo custo de aplicação, replicáveis a cada nova oferta da disciplina.

O questionário pós-atividade é respondido em poucos minutos, sem demandar infraestrutura adicional, e combina itens quantitativos em escala Likert de cinco pontos (Likert, 1932) com questões abertas qualitativas, permitindo tanto a análise estatística comparativa entre atividades quanto a identificação de percepções e dificuldades específicas. Por serem os itens Likert variáveis qualitativas ordinais, e não intervalares, o tratamento descritivo dos dados emprega mediana e intervalo interquartílico como medidas de tendência central e de dispersão, e os procedimentos comparativos previstos são não paramétricos, como o teste de Mann-Whitney

(Mann; Whitney, 1947). A estrutura do instrumento é mantida fixa entre todas as atividades do semestre, garantindo comparabilidade longitudinal dos dados.

O questionário é organizado em quatro blocos. O **Bloco 1**, denominado *Aprendizagem Percebida*, é o único que varia entre atividades: contém cinco afirmações na escala Likert diretamente alinhadas aos objetivos de aprendizagem declarados em cada atividade. Os estudantes indicam seu grau de concordância com afirmações do tipo "após esta atividade, consigo explicar [conceito central da atividade]", permitindo uma autoavaliação do ganho conceitual específico ao tópico trabalhado. A variação desse bloco entre atividades é intencional: ela captura o aprendizado percebido de cada tópico sem impor um instrumento genérico desconectado do conteúdo abordado.

O **Bloco 2**, denominado *Usabilidade do Ambiente*, contém quatro itens fixos baseados nos construtos centrais do *Technology Acceptance Model* (TAM), proposto por Davis: a facilidade de uso percebida (*Perceived Ease of Use*) e a utilidade percebida (*Perceived Usefulness*) do ambiente tecnológico da atividade (Davis, 1989). Os itens são formulados de forma genérica em relação à tecnologia empregada, referindo-se ao "ambiente da atividade", de modo que o bloco se aplica sem alteração a qualquer atividade do conjunto, independentemente das ferramentas utilizadas. Esse bloco permite rastrear a evolução da familiarização dos estudantes com os ambientes práticos ao longo do semestre, visto que esse é um dos objetivos específicos desse trabalho.

O **Bloco 3**, denominado *Carga de Trabalho*, é composto por seis itens fixos correspondentes às seis subescalas finais do *NASA Task Load Index* (NASA-TLX), instrumento desenvolvido por Hart; Staveland a partir de um programa de pesquisa de três anos envolvendo 16 experimentos distintos (Hart; Staveland, 1988). As subescalas são: Demanda Mental, Demanda Física, Demanda Temporal, Desempenho, Esforço e Nível de Frustração. O item de Desempenho possui polaridade invertida na análise: maior concordância com "fui bem-sucedido na atividade" indica *menor* carga de trabalho percebida, em conformidade com a escala original. As respostas (1 a 5) são convertidas para a escala 0 a 100 do NASA-TLX pela fórmula $\text{score} = (\text{resposta} - 1) \times 25$. A pontuação composta é calculada como o *Raw TLX*, ou seja, a média simples das seis subescalas, cuja validade é equivalente à versão ponderada por comparações par-a-par, conforme demonstrado por Hart em revisão retrospectiva do instrumento (Hart, 2006), sendo consideravelmente mais simples de aplicar no contexto educacional. O Raw TLX é, portanto, um escore composto calculado por respondente, conforme a definição do instrumento; a agregação de escores entre respondentes, como as demais análises descritivas deste trabalho, é reportada por mediana e intervalo interquartílico, dada a natureza ordinal da escala de origem.

O **Bloco 4** contém três questões abertas fixas: o que foi mais difícil na atividade, o que o estudante mudaria para facilitar o aprendizado e que conceito ou habilidade gostaria de explorar a seguir. As respostas passam por análise temática simples, isto é, categorização dos temas recorrentes entre os respondentes, cujos resultados alimentam o ciclo de revisão e aprimoramento das atividades subsequentes. A confiabilidade interna de cada bloco quantitativo

é verificada pelo coeficiente alfa de Cronbach, adotando-se o limiar $\alpha \geq 0{,}70$ como critério de aceitabilidade (Tavakol; Dennick, 2011).

O segundo instrumento de avaliação é o **relatório reflexivo** entregue ao final de cada atividade. Cada atividade disponibiliza, junto ao código no repositório, um *template* em formato *Markdown* (`relatorio-template.md`) cuja estrutura espelha os níveis de andaimagem da atividade, orientação, inspeção sistemática e modificação, contendo tabelas de inspeção (por exemplo, "descreva o que cada chamada da API faz"), respostas conceituais às perguntas-guia, registro das modificações realizadas no código ou no ambiente, observações livres sobre comportamentos inesperados e uma "dúvida para a próxima aula". O relatório cumpre, assim, função pedagógica dupla: por um lado, força os estudantes a articularem o "porquê" das observações experimentais, indo além do "fazer funcionar" característico de práticas puramente operacionais; por outro, alimenta o encontro seguinte da disciplina com dúvidas reais emergidas da experimentação, fechando o ciclo da Sala de Aula Invertida.

A análise dos relatórios é qualitativa e estruturada por comparação com um gabarito específico de cada atividade, mantido junto ao material didático no repositório (`atividade-*-relatorio-gabarito.md`). O gabarito enumera as respostas esperadas e os conceitos-chave que devem aparecer em cada item, permitindo identificar de forma sistemática equívocos conceituais recorrentes, como por exemplo, confusão entre *partição* e *parada* de nó na atividade de consenso. Esses padrões alimentam o ciclo de refinamento das atividades em ofertas subsequentes da disciplina, complementando os temas categorizados nas questões abertas do Bloco 4 do questionário.

Em conjunto, os dois instrumentos sustentam uma avaliação triangulada: o questionário oferece dado quantificável sobre percepção, usabilidade e carga, comparável longitudinalmente entre atividades e entre ofertas da disciplina; o relatório oferece evidência qualitativa da compreensão conceitual efetivamente construída pelos estudantes. Discrepâncias entre os dois, como por exemplo, alta concordância no Bloco 1 ("consigo explicar X") combinada a equívocos sistemáticos sobre X no relatório, são indicadores de excesso de confiança autopercebida e sinalizam pontos da atividade que merecem reforço pedagógico em iterações futuras.

5.3 CONTEXTO DE APLICAÇÃO

A aplicação empírica das quatro atividades propostas no Capítulo 4 ocorreu durante o semestre 2026.1, em uma turma da disciplina de Computação Distribuída ofertada no curso de Ciência da Computação da UFSC. As quatro atividades foram disponibilizadas em sequência ao longo do semestre, na ordem estabelecida pela matriz conceitual da Seção 4.1, acompanhando a progressão dos tópicos teóricos cobertos em sala. Cada atividade foi liberada após a apresentação em aula do conteúdo correspondente.

O formato de aplicação adotado foi predominantemente extracurricular. As atividades não integraram a carga horária regular da disciplina nem foram associadas a uma avaliação formal componente da nota final. O único incentivo institucional formal foi a possibilidade

de *arredondamento* da nota final para estudantes que terminassem o semestre a até 0,5 ponto do corte de aprovação, mecanismo cuja abrangência é estruturalmente restrita: atinge apenas o subconjunto de estudantes posicionados nessa faixa específica do desempenho final e, mesmo dentro desse subconjunto, oferece benefício marginal. Essa configuração de incentivo é discutida no presente capítulo como hipótese causal principal para o padrão de adesão observado, retomada de forma sintética em Seção 6.2.

O convite à participação foi feito de forma combinada: anúncio presencial em sala, divulgação do repositório Git de cada atividade pelo ambiente virtual de aprendizagem da disciplina e disponibilização do *link* para o questionário pós-atividade junto ao material de cada prática. A participação no questionário foi anônima, sem identificação obrigatória do respondente, conforme descrito no Bloco de Identificação opcional do instrumento.

A infraestrutura computacional adotada baseou-se nos próprios equipamentos dos estudantes (*notebooks* pessoais), conforme o princípio de Portabilidade do ambiente padronizado (Seção 4.3). As Atividades 1, 2 e 4 exigiram apenas Docker e Docker Compose; a Atividade 3 exigiu Multipass sobre KVM/QEMU. Não foi necessário o uso de laboratório institucional, e a documentação de instalação de cada ferramenta foi disponibilizada junto ao material de preparação.

O material de preparação efetivamente disponibilizado variou entre as atividades. Para as Atividades 1, 3 e 4 foram disponibilizados, além do código e do enunciado no repositório, slides da disciplina e documento de contexto teórico em português contendo o referencial conceitual mínimo necessário para a fase pré-atividade da Sala de Aula Invertida. A Atividade 2 (Espaço de Tuplas) constituiu exceção: a escassez de literatura didática em português sobre o tópico restringiu o material pré-atividade ao documento elaborado especificamente para este trabalho, restrição discutida em Subseção 5.6.2.

Ao final do prazo de coleta considerado para a escrita deste capítulo, foram obtidas **sete respostas** ao questionário pós-atividade, seis referentes à Atividade 1 (*Sockets* TCP) e uma referente à Atividade 2 (Espaço de Tuplas), e **oito relatórios reflexivos**, sete da Atividade 1 e um da Atividade 2. As Atividades 3 e 4 não receberam respostas ao questionário nem relatórios no período considerado. Esse tamanho amostral, embora superior ao registrado em iterações anteriores deste capítulo, permanece aquém do mínimo necessário para os procedimentos inferenciais originalmente previstos. O teste de Mann-Whitney entre atividades e a análise longitudinal da carga de trabalho exigem amostras significativamente maiores, e o coeficiente alfa de Cronbach, embora computável para a Atividade 1, deve ser interpretado com cautela substancial dado o tamanho amostral abaixo do recomendado. O tratamento dos dados nas seções subsequentes é condicionado por essa restrição, conforme detalhado na abertura da Seção 5.4.

5.4 RESPOSTAS DO QUESTIONÁRIO

Os procedimentos analíticos descritos na Seção 5.2, pressupõem amostras de tamanho mínimo que viabilizem inferência estatística. O número de respostas obtidas até o presente momento (seis para a Atividade 1 e uma para a Atividade 2, conforme indicado em Seção 5.3) torna os procedimentos comparativos inaplicáveis: testes de hipóteses entre atividades apresentariam poder estatístico essencialmente nulo dado $n_{A2} = 1$, e a ausência de respostas nas Atividades 3 e 4 inviabiliza qualquer análise longitudinal. O coeficiente alfa de Cronbach é computado para os três blocos quantitativos da Atividade 1 e reportado em Subseção 5.4.2, mas com a ressalva explícita de que $n = 6$ situa-se abaixo do mínimo de 10–15 respondentes recomendado na literatura para estimativa estável de consistência interna (Tavakol; Dennick, 2011), devendo os valores obtidos ser tratados como indicador preliminar de confiabilidade. Esta seção apresenta, portanto, uma leitura **descritiva** das respostas recebidas, sem pretensão inferencial. Os resultados aqui registrados constituem sinalização preliminar destinada a alimentar iterações futuras da aplicação do arcabouço e não evidência empírica conclusiva sobre a eficácia das atividades.

5.4.1 Perfil e cobertura das respostas

A Tabela 2 sistematiza a distribuição das respostas ao questionário e dos relatórios reflexivos entregues por atividade, consolidando em um único quadro a cobertura dos dois instrumentos de avaliação. As colunas “Questionários” e “Relatórios” indicam o número de artefatos submetidos por instrumento; a coluna “Cobertura” indica se houve ao menos um artefato (de qualquer instrumento) para a atividade (•) ou nenhum (○).

Tabela 2 – Cobertura dos instrumentos de avaliação (questionário pós-atividade e relatório reflexivo) por atividade aplicada.

Atividade	**Questionários**	**Relatórios**	**Cobertura**
A1 – Sockets TCP	6	7	•
A2 – Espaço de Tuplas	1	1	•
A3 – NTP/chrony	0	0	○
A4 – Raft	0	0	○
Total	**7**	**8**	—

Fonte: Elaborado pelo autor (2026).

A concentração das respostas na Atividade 1, a cobertura isolada da Atividade 2 (um questionário e um relatório) e a ausência total nas Atividades 3 e 4 são discutidas em Subseção 5.6.1. As subseções seguintes apresentam a leitura descritiva dos seis questionários da Atividade 1 e, ao final, uma observação sintética sobre a única resposta da Atividade 2.

5.4.2 Leitura descritiva por bloco

Esta subseção apresenta, para cada um dos quatro blocos do questionário (Seção 5.2), as respostas obtidas em forma tabular ou sintética, acompanhadas de uma leitura qualitativa breve. A organização por bloco, em vez de por atividade, evita a multiplicação de subseções com baixa densidade informacional e permite que o leitor acompanhe transversalmente a percepção dos respondentes em cada dimensão avaliada. Por se tratarem de dados ordinais (escala Likert), as estatísticas descritivas reportadas nas tabelas desta seção são a mediana, como medida de tendência central, e o intervalo interquartílico (IIQ, do primeiro ao terceiro quartil), como medida de dispersão[1]. A pontuação composta Raw TLX do Bloco 3 constitui exceção parcial: por definição do instrumento, ela é calculada, para cada respondente, como a média das seis subescalas convertidas (Seção 5.2); a agregação entre respondentes, contudo, também é reportada por mediana e IIQ.

Bloco 1 – Aprendizagem Percebida

Os itens do Bloco 1 variam por atividade, conforme detalhado em Seção 5.2. A Tabela 3 registra, para cada um dos seis respondentes da Atividade 1, a pontuação atribuída na escala Likert de 1 a 5, acompanhada de mediana e intervalo interquartílico por item.

Tabela 3 – Respostas ao Bloco 1 (Aprendizagem Percebida) da Atividade 1 ($n = 6$).

Respondente	**I1**	**I2**	**I3**	**I4**	**I5**
R1	4	4	3	4	5
R2	4	5	5	5	5
R3	5	5	5	5	5
R4	4	5	5	4	4
R5	5	5	5	5	3
R6	5	4	5	4	5
Mediana	4,5	5	5	4,5	5
IIQ	4–5	4–5	5–5	4–5	4–5

Fonte: Elaborado pelo autor (2026).

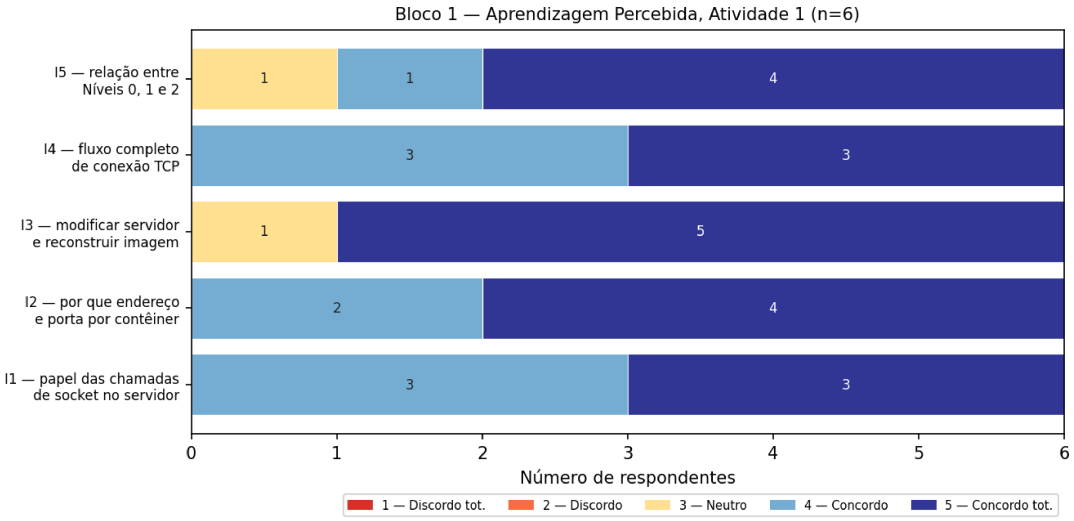
As medianas dos cinco itens situam-se entre 4,5 e 5 (I1 e I4: 4,5; I2, I3 e I5: 5), indicando autoavaliação positiva do ganho conceitual em todos os objetivos de aprendizagem declarados na Atividade 1. Todos os respondentes registraram pelo menos *Concordo* (4) em pelo menos quatro dos cinco itens, e duas respostas (R3 e R2) registraram *Concordo totalmente* (5) em todos os itens. Os itens I2 (“por que dois processos em contêineres distintos precisam de endereço e porta”) e I3 (“modificar um servidor TCP e reconstruir a imagem Docker”) registraram a mesma mediana (5), porém com perfis distintos: I2 apresenta dispersão mínima

[1] Com n par, a mediana de dados ordinais pode assumir valor intermediário entre dois pontos consecutivos da escala (por exemplo, 4,5), correspondendo à média dos dois valores centrais da amostra ordenada.

(apenas valores 4 e 5), enquanto I3 concentra cinco respostas no valor máximo e uma resposta isolada em *Neutro* (R1 = 3), que amplia a amplitude total do item (3 a 5) sem deslocar a mediana. A dispersão dos itens I5 e I3 está concentrada em respostas individuais isoladas, R5 em I5 e R1 em I3, e não em divisões sistemáticas entre subgrupos da amostra. O coeficiente alfa de Cronbach calculado para o Bloco 1 foi $\alpha = 0{,}31$, valor abaixo do limiar de 0,70 adotado em Seção 5.2; esse resultado, contudo, deve ser interpretado considerando dois fatores: o tamanho amostral $n = 6$ está aquém do mínimo recomendado para estimativa estável de α (Tavakol; Dennick, 2011), e o forte efeito de teto observado (variância intra-item baixa porque a maioria das respostas concentra-se nos valores 4 e 5) reduz mecanicamente o numerador da razão de variâncias que compõe o coeficiente. Em outras palavras, o α baixo aqui não indica necessariamente inconsistência interna do construto, mas a combinação entre amostra pequena e respostas homogeneamente altas.

A Figura 4 apresenta a distribuição das respostas do Bloco 1 em formato de barras empilhadas, evidenciando visualmente a concentração nos valores 4 e 5 em todos os itens.

Figura 4 – Distribuição das respostas ao Bloco 1 (Aprendizagem Percebida) da Atividade 1 ($n = 6$), por item e valor Likert.



Fonte: Elaborado pelo autor (2026).

Bloco 2 – Usabilidade do Ambiente

A Tabela 4 consolida as respostas aos quatro itens do Bloco 2, idêntico entre atividades e fundamentado no *Technology Acceptance Model* (Davis, 1989).

A facilidade de uso percebida do ambiente Docker (I6, mediana 5) e a clareza das instruções dos três níveis (I7, mediana 5) receberam concordância forte e consistente entre os seis respondentes, com nenhum respondente atribuindo pontuação inferior a 3 a esses itens. A adequação da complexidade do código fornecido (I8, mediana 4) e a autonomia para concluir a atividade (I9, mediana 4,5) registram valores centrais ligeiramente menores e maior hetero-

Tabela 4 – Respostas ao Bloco 2 (Usabilidade do Ambiente) da Atividade 1 ($n = 6$).

Respondente	**I6**	**I7**	**I8**	**I9**
R1	4	3	4	2
R2	5	5	4	5
R3	4	5	5	5
R4	5	5	3	5
R5	5	5	4	4
R6	5	4	5	4
Mediana	5	5	4	4,5
IIQ	4–5	4–5	4–5	4–5

Fonte: Elaborado pelo autor (2026).

geneidade nas caudas, com destaque para a amplitude total do item I9 (2 a 5). Essa amplitude elevada é integralmente explicada pela resposta isolada de R1 (*Discordo*, valor 2); os cinco demais respondentes declararam autonomia satisfatória ou plena (valores 4 ou 5). A combinação entre autoavaliação positiva de aprendizagem no Bloco 1 e baixa autonomia declarada por R1 no Bloco 2 sugere que, no caso desse respondente, a ajuda externa eventualmente acionada funcionou como andaime efetivo e não como obstáculo ao aprendizado, padrão compatível com a estratégia de andaimagem discutida em Subseção 2.2.1. O coeficiente alfa de Cronbach calculado para o Bloco 2 foi $\alpha = 0{,}58$; abaixo do limiar adotado, mas com a mesma ressalva de $n = 6$ e efeito de teto que se aplica ao Bloco 1.

Bloco 3 – Carga de Trabalho (NASA-TLX)

A Tabela 5 registra as respostas brutas em escala Likert e a pontuação Raw TLX convertida pela fórmula $\text{score} = (\text{resposta} - 1) \times 25$, com inversão de polaridade no item de Desempenho ($\text{score}_{\text{OP}} = (5 - \text{resposta}) \times 25$), conforme Seção 5.2.

Tabela 5 – Respostas brutas ao Bloco 3 (NASA-TLX) e Raw TLX (0–100) por respondente da Atividade 1 ($n = 6$).

Respondente	**MD**	**PD**	**TD**	**OP⋆**	**EF**	**FR**	**Raw TLX**
R1	4	4	3	5	3	2	45,8
R2	3	2	1	5	3	1	20,8
R3	4	3	2	5	3	1	33,3
R4	4	2	1	4	1	1	20,8
R5	2	1	1	5	1	1	4,2
R6	1	1	2	4	1	3	16,7
Mediana	3,5	2	1,5	5	2	1	20,8
IIQ	2–4	1–3	1–2	4–5	1–3	1–2	16,7–33,3

Fonte: Elaborado pelo autor (2026).

O símbolo $\star$ em OP indica polaridade invertida, conforme Seção 5.2: pontuação 5 (“fui bem-sucedido”) converte-se em 0 na escala NASA-TLX. Todos os respondentes declararam pontu-

ação 4 ou 5 em desempenho (OP), indicando que se consideraram bem-sucedidos no cumprimento dos objetivos da atividade. Os valores Raw TLX obtidos variam de 4,2 (R5) a 45,8 (R1), com mediana 20,8 e IIQ de 16,7 a 33,3 em escala 0–100. A mediana situa-se claramente na faixa de *carga baixa a moderada*, abaixo dos valores típicos relatados na literatura para tarefas que exigem programação substancial (Hart, 2006), compatível com o desenho da Atividade 1 como primeira prática do semestre, com escopo deliberadamente restrito ao Nível 2 de modificação. A análise por subescala revela que a Demanda Mental (MD, mediana 62,5 na escala 0–100) é o principal vetor de carga percebida, seguida por Demanda Física e Esforço (PD e EF, 25,0 cada), Demanda Temporal (TD, 12,5) e, com medianas nulas, Frustração e Desempenho (FR e OP, 0,0, este após inversão). Esse perfil, carga concentrada na dimensão mental e baixa em frustração e pressão temporal, é consistente com o desenho declarado da atividade: trabalho cognitivo de inspeção e modificação de código, sem prazo apertado nem dimensão emocional adversa. A heterogeneidade entre os respondentes mantém-se: R1 reporta o maior Raw TLX (45,8), enquanto R5 e R6 reportam valores muito baixos (4,2 e 16,7), reproduzindo o padrão de heterogeneidade já observado no Bloco 2. O coeficiente alfa de Cronbach calculado para o Bloco 3 (com o item OP invertido antes do cálculo, para alinhar a polaridade ao construto de carga) foi $\alpha = 0{,}61$, abaixo do limiar adotado mas o mais alto entre os três blocos, sob as mesmas ressalvas de $n = 6$. Importa observar que essas pontuações **não** constituem evidência de eficácia da atividade, dada a inviabilidade de inferência comparativa entre atividades neste estágio da coleta; servem como sinalização inicial sobre a calibragem do nível de dificuldade da Atividade 1.

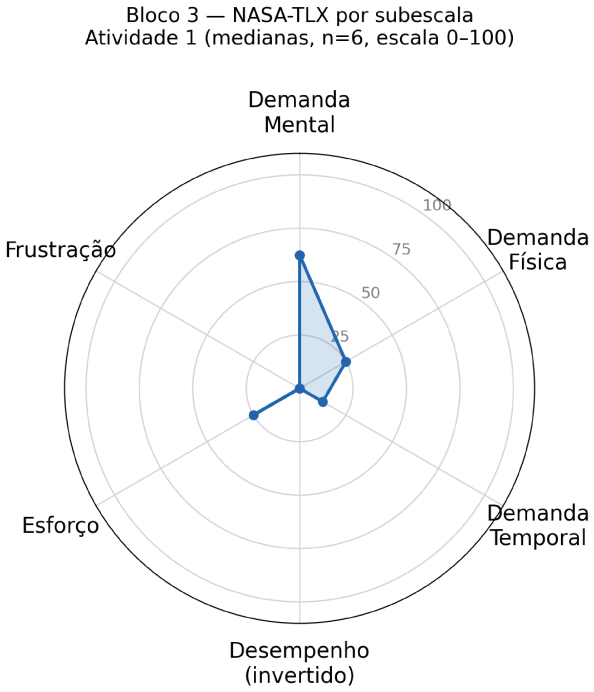
A Figura 5 ilustra o perfil de carga por subescala (medianas) em formato de gráfico radar, tornando visível a dominância da Demanda Mental e as medianas nulas de Frustração e de Desempenho (invertido) no centro do gráfico.

Bloco 4 – Questões Abertas

As respostas às três questões abertas do Bloco 4 (o que foi mais difícil, o que mudaria e o que gostaria de explorar a seguir) são sintetizadas na Tabela 6. Apenas os respondentes que preencheram ao menos uma das três questões são listados; os demais omitiram totalmente o bloco.

Das seis respostas obtidas, três respondentes (R2, R4 e R6) omitiram todas as questões abertas; os três restantes preencheram ao menos uma. Mesmo com o tamanho reduzido, dois temas qualitativamente distintos emergem das respostas substantivas. O primeiro tema é a **compreensão de Docker e contêineres** como superfície de aprendizado: tanto R3 quanto R5 mencionam Docker explicitamente, com R3 declarando dificuldade inicial superada ao longo da atividade e R5 indicando interesse em aprofundamento posterior, sinalizando que, embora não seja o tópico central da Atividade 1 (*sockets* TCP), Docker ocupa papel cognitivo relevante nas percepções dos respondentes. O segundo tema é a **renderização do *template* de relatório**: R3 reporta confusão com a formatação do arquivo `relatorio-template.md` antes de visualizar

Figura 5 – Perfil NASA-TLX por subescala (medianas) para a Atividade 1 ($n = 6$, escala 0–100). Valores maiores indicam maior carga; Desempenho aparece invertido conforme a escala original.



Fonte: Elaborado pelo autor (2026).

Tabela 6 – Síntese das respostas ao Bloco 4 (Questões Abertas) da Atividade 1.

Resp.	**Mais difícil**	**O que mudaria**	**Explorar a seguir**
R1	Seguir um padrão comum como estipulado na atividade, exigindo maior formalismo, e manter um código bem estruturado e legível	—	—
R3	Entender Docker e contêineres, mas a atividade ajudou a compreender de maneira clara e objetiva	Formatação do arquivo `relatorio-template.md` foi confusa; só ficou clara após visualizar a versão renderizada no GitHub	—
R5	Nada foi difícil na atividade	Nada; a atividade é completa e didática	Conceitos de Docker

Fonte: Elaborado pelo autor (2026).

a versão renderizada via GitHub. Esse achado é operacionalmente acionável e direto: a forma como o *template Markdown* é apresentado aos estudantes pode ser revisada (por exemplo, fornecendo pré-visualização HTML lado a lado com a fonte) para reduzir a barreira de entrada. A resposta de R1, sobre dificuldade com padrão formal de código, permanece como tema isolado. As respostas a “o que gostaria de explorar a seguir”, cujo propósito é alimentar o ciclo da Sala

de Aula Invertida (Seção 5.2), restringem-se a uma única menção (R5: “Conceitos de Docker”), limitando o insumo qualitativo previsto para o refinamento da próxima atividade.

Atividade 2 – observação isolada

A única resposta da Atividade 2 (Espaço de Tuplas), obtida em 6 de junho de 2026, registrou *Concordo* ou *Concordo totalmente* nos cinco itens do Bloco 1 específico da atividade (pontuações 4 ou 5 em I1–I5, abordando as três operações `write`/`take`/`read`, os dois tipos de desacoplamento, o papel do `reggie`, a identificação de cenário computacional atual e a ampliação da compreensão sobre coordenação sem comunicação direta), bem como *Concordo totalmente* em todos os quatro itens do Bloco 2 (TAM) e em todas as seis subescalas do Bloco 3 (NASA-TLX). A pontuação Raw TLX correspondente é 83,3 – valor isolado que, lido em conjunto com o *Concordo totalmente* simultâneo em Desempenho (OP, polaridade invertida) e em todas as demais subescalas de carga, configura um padrão sugestivo de resposta uniforme sem diferenciação entre itens (*straight-lining*). Esse padrão, combinado ao tamanho amostral $n = 1$, recomenda explicitamente que essa resposta não seja interpretada como sinal sobre a usabilidade ou a carga real da Atividade 2, e sim arquivada como dado bruto cuja leitura informativa depende de ampliação substantiva da amostra em iterações futuras.

5.4.3 Indicações iniciais e limitações

A leitura descritiva apresentada nas subseções anteriores oferece, no atual estágio da coleta, apenas **sinalizações preliminares** sobre a percepção dos respondentes em relação às atividades aplicadas. As medianas das três dimensões quantitativas da Atividade 1 ($n = 6$), com Aprendizagem Percebida entre 4,5 e 5, Usabilidade entre 4 e 5 e Raw TLX mediano de 20,8 em escala 0–100, compõem perfil internamente coerente: autoavaliação positiva de ganho conceitual, percepção positiva do ambiente padronizado e carga de trabalho baixa a moderada concentrada na dimensão mental. Essas sinalizações, contudo, não constituem evidência estatística sobre a eficácia pedagógica do arcabouço: os coeficientes alfa de Cronbach computados situam-se abaixo do limiar de 0,70 (Bloco 1: $\alpha = 0{,}31$; Bloco 2: $\alpha = 0{,}58$; Bloco 3: $\alpha = 0{,}61$), e a interpretação desses valores está condicionada ao tamanho amostral $n = 6$ e ao forte efeito de teto observado nos Blocos 1 e 2, combinação que reduz mecanicamente o numerador da razão de variâncias da fórmula de α. Quaisquer afirmações categóricas sobre eficácia derivadas dessa amostra incorreriam em superinterpretação dos dados disponíveis. As respostas obtidas servem, portanto, a três funções limitadas: (i) demonstrar a operacionalização efetiva do instrumento de coleta junto ao público-alvo; (ii) fornecer pistas qualitativas iniciais, particularmente os temas de Docker e de renderização do *template* de relatório identificados no Bloco 4, que podem ser exploradas em iterações futuras e alimentam a discussão de refinamento na Seção 5.7; e (iii) compor, junto à análise dos relatórios reflexivos (Seção 5.5) e às observações da aplicação (Seção 5.6), o triangulado mínimo de evidência sobre o processo. A ampliação do

tamanho amostral por meio de aplicações subsequentes da disciplina, conforme registrado na frente de continuidade descrita em Seção 6.3, é condição necessária para que os procedimentos inferenciais originalmente previstos passem a ser aplicáveis.

5.5 ANÁLISE DOS RELATÓRIOS REFLEXIVOS

Em paralelo às respostas ao questionário, foram analisados os relatórios reflexivos entregues pelos respondentes. A unidade de análise nesta seção é o *artefato textual produzido*, não o respondente: cada relatório, comparado ao seu gabarito específico, fornece evidência qualitativa sobre a compreensão conceitual articulada após a experimentação, independentemente do número total de respondentes. Esse deslocamento metodológico permite que um pequeno número de relatórios produza observações empíricas defensáveis, embora a baixa quantidade absoluta limite a generalização das conclusões a categorias amplas de equívoco, e não a frequências relativas estatisticamente robustas.

5.5.1 Cobertura e procedimento de análise

A distribuição dos relatórios recebidos por atividade está consolidada na Tabela 2 (Subseção 5.4.1), na coluna "Relatórios". O procedimento de análise consistiu na comparação item a item entre cada relatório entregue e o respectivo gabarito mantido no repositório da atividade (`atividade-*-relatorio-gabarito.md`), identificando: (i) acertos conceituais relevantes, (ii) equívocos conceituais explicitamente articulados, (iii) lacunas (itens do gabarito não abordados pelo relatório), e (iv) observações inesperadas que extrapolam o gabarito e sinalizam compreensão acima do esperado.

Os oito relatórios obtidos são identificados ao longo desta seção como **D1** a **D8**. D1 a D7 referem-se à Atividade 1 (Sockets TCP) e foram produzidos por seis duplas/solos e um trio: D1 (dupla), D2 (dupla), D3 (solo), D4 (trio, única ocorrência de formato trio no conjunto), D5 (solo), D6 (solo) e D7 (solo). D8 refere-se à Atividade 2 (Espaço de Tuplas), produzido pelo mesmo estudante de D7.

Em D1 a D7, todos os sete artefatos preencheram textualmente os Níveis 1 e 2 do gabarito de A1, com profundidade heterogênea mas conceitualmente consistente: as tabelas 1.1 e 1.2 (chamadas de *socket* no servidor e no cliente), a pergunta 1.3 (rede e contêineres) e as três justificativas do Nível 2 (modificação `upper()`, contador acumulado e necessidade do `-build`) aparecem em pelo menos seis dos sete relatórios. As exceções e desvios pontuais, erro de sintaxe *Markdown* em D2, articulação ambígua em 1.3 em alguns relatórios e ausência de texto no Nível 2 em D7 mesmo com as modificações de código presentes em `servidor.py`, são consolidados na tabela de equívocos e acertos da Subseção 5.5.2. As seções abertas "Observações livres" e "Dúvida para a próxima aula" apresentam padrão de não-preenchimento majoritário em A1, com a notável exceção de D6, que formulou dúvida conceitual substantiva (`sendall()` versus `send()`) e fechou efetivamente o ciclo da Sala de Aula Invertida previsto em Seção 5.2.

D8 (A2) preencheu textualmente todos os subitens dos Níveis 0, 1 e 2, incluindo a tabela das três operações `write/take/read`, os experimentos guiados de `reggie` (§1.2), desacoplamento espacial (§1.3), bloqueio (§1.4) e escalabilidade horizontal (§1.5), além das duas modificações do Nível 2 (prioridade de tarefas e serviço monitor). O trecho de `Monitor.java` foi colado, evidenciando completude da modificação aberta. "Observações livres" foi preenchido ("Bom exercício, instiga a pesquisa sobre um assunto que não foi comentado tanto em aula"); "Dúvida para a próxima aula" ficou em branco. Esse perfil de profundidade desigual entre os artefatos, e em particular o achado emergente no experimento de desacoplamento temporal, é discutido na Subseção 5.5.2.

5.5.2 Equívocos conceituais e acertos relevantes

A Tabela 7 consolida os equívocos conceituais identificados nos relatórios analisados, indicando a atividade de origem, a categoria do equívoco e o conceito do gabarito que não foi corretamente articulado. Apenas atividades com observações concretas são listadas; a ausência de uma atividade na tabela não implica ausência de equívocos, apenas ausência de dados para essa atividade no momento da escrita.

Além dos pontos acima, foram identificados acertos relevantes. Em A1, **seis dos sete relatórios** (D1 a D6) articularam textualmente as decisões do Nível 2 e a necessidade de `-build`, demonstrando conexão entre alteração de código-fonte e reconstrução de imagem Docker, o que corresponde à articulação majoritária do conceito-chave da modificação. Todos os sete grupos executaram tecnicamente a Atividade 1 nos Níveis 0, 1 e 2: os sete `servidor.py` modificados estão presentes nos repositórios entregues, com a substituição de `resposta = mensagem` por `resposta = mensagem.upper()` e a introdução de `total_mensagens` como contador acumulado. **D6**, além do preenchimento dos níveis, formulou a única dúvida conceitual substantiva do conjunto, "Por que usar `sendall()` ao invés de `send()`, já que o método visto em aula foi `send()`?", fechando o ciclo da Sala de Aula Invertida ao alimentar o encontro presencial seguinte com dúvida real emergida da experimentação. Em A2, **D8** preencheu todos os subitens dos Níveis 0, 1 e 2 com profundidade conceitual adequada, capturando corretamente a limitação do uso de `read()` para contagem de tarefas pendentes ("não há garantia de que essas tuplas sejam diferentes"), o papel do `reggie` como serviço de descoberta cuja queda preserva sessões já estabelecidas, e o paralelo com sistemas modernos (Consul, Eureka), evidenciando articulação ativa entre o modelo histórico do Apache River e o estado-da-arte contemporâneo.

Para além dos equívocos e acertos individuais consolidados na tabela acima, a análise do relatório D8 revelou um achado pedagógico de natureza distinta: a observação cuidadosa do estudante expôs um *defeito de design no procedimento experimental* da própria atividade, motivando refinamento iterativo do material didático. O experimento de desacoplamento temporal do Nível 0 da Atividade 2 prescrevia, na versão originalmente disponibilizada, a execução de `docker compose up reggie javaspaces produtor` seguida de `Ctrl+C` após o produtor encerrar, e em seguida a execução de `docker compose up consumidor`. O estu-

Tabela 7 – Equívocos conceituais e lacunas identificados nos relatórios reflexivos das Atividades 1 e 2.

Ativ.	**Artefato**	**Categoria**	**Conceito do gabarito não articulado**
A1	D7	Lacuna textual no Nível 2	As modificações de *upper-case* e contador de mensagens estão presentes no código de `servidor.py`, mas os campos textuais "o que foi alterado e por quê" e "por que `-build` é necessário" não foram preenchidos. A execução prática ocorreu; a articulação reflexiva, não.
A1	D2	Sintaxe *Markdown* quebrada	As tabelas 1.1 e 1.2 receberam um separador `|` extra por linha, empurrando o conteúdo das respostas para fora da célula. O texto das respostas está presente em todas as linhas; apenas a renderização ficou comprometida.
A1	D2 e D7	Articulação parcial em 1.3	A resposta à pergunta sobre rede e contêineres captura o núcleo do conceito ("cada contêiner tem seu próprio `localhost`", em D7; "ambiente externo", em D2) sem articular o mecanismo subjacente: rede *bridge* do Docker com resolução de nome de serviço para IP interno do contêiner.
A1	5 de 7 (D2, D3, D4, D5, D7)	Lacuna nas seções abertas	"Observações livres" e "Dúvida para a próxima aula" deixadas em branco. D1 ("Nenhuma") registra ausência sem reflexão; apenas D6 formulou dúvida conceitual substantiva (`sendall()` versus `send()`), fechando efetivamente o ciclo da Sala de Aula Invertida.

Fonte: Elaborado pelo autor (2026).

dante executou exatamente conforme a instrução e registrou, na pergunta sobre se o consumidor encontrou as tarefas depositadas anteriormente, a resposta "Não" acompanhada do trecho de logs "`Container atividade-espaco-de-tuplas-javaspaces-1 Stopping ... javaspaces-1 exited with code 143`", concluindo: "o experimento dá a entender que o desacoplamento temporal só funciona se o espaço de tuplas continuar existindo entre a execução do produtor e consumidor".

A observação está empiricamente correta e a conclusão conceitual também: o servidor Outrigger do Apache River mantém as tuplas em memória apenas (sem persistência em disco), e o `Ctrl+C` sobre `docker compose up reggie javaspaces produtor` derruba os três serviços simultaneamente, eliminando o estado que o experimento pretende demonstrar persistir. A redação original do experimento, portanto, não conseguia produzir o comportamento que se queria observar, ou seja, a propriedade central de desacoplamento temporal, e o estudante chegou à conclusão conceitual correta sobre o pré-requisito de persistência *apesar de* a evidência experimental imediata ter saído invertida. Esse caso ilustra de forma direta o valor do raciocínio crítico estruturado pela combinação Sala Invertida + Aprendizagem Baseada em Investigação: o estudante, preparado previamente pelo material teórico, observou empiricamente, detectou a contradição entre teoria e experiência, e articulou conclusão metacognitiva consistente com o modelo conceitual. A partir dessa observação, o procedimento experimental do README da Atividade 2 foi revisto em duas fases sequenciais executadas em dois terminais distintos: o Terminal 1 mantém `reggie` e `javaspaces` ativos continuamente (`docker compose up reggie javaspaces`), enquanto o Terminal 2 executa primeiro `docker compose up produtor`, aguarda seu término, e em seguida executa `docker compose up consumidor`, permitindo que o espaço de tuplas permaneça vivo entre as duas execuções e que o desacoplamento temporal emerja como propriedade observável. O refinamento iterativo decorrente desse achado é registrado em nível operacional na Seção 5.7.

Esse padrão, com execução técnica completa em sete dos sete grupos de A1 e em A2, articulação textual majoritariamente presente em A1 com ocorrências isoladas de lacunas, dúvida conceitual substantiva apenas em D6, e raciocínio metacognitivo de D8 expondo limitação do próprio material, é um achado qualitativo relevante e fornece evidência direta para a discussão de refinamento na Seção 5.7.

5.6 OBSERVAÇÕES DA APLICAÇÃO

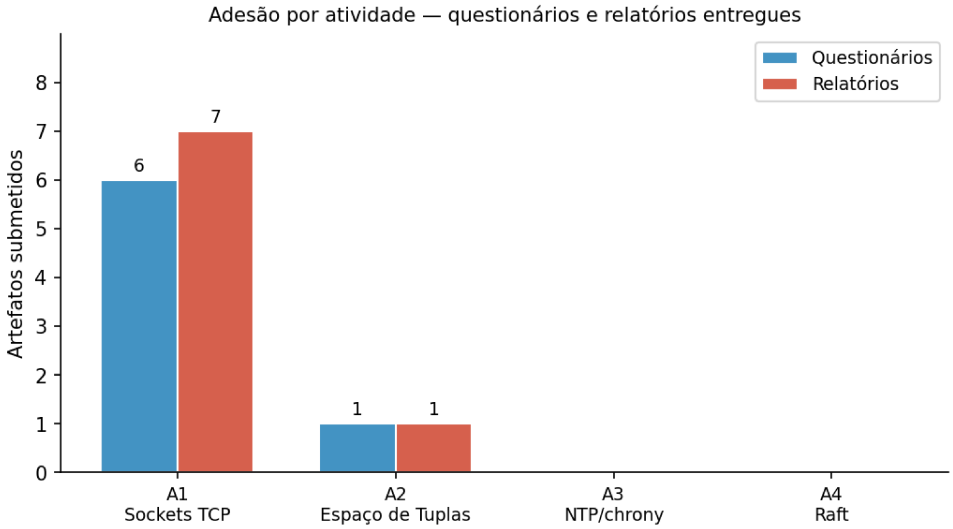
Em face do tamanho amostral reduzido dos instrumentos de avaliação, as observações registradas durante o processo de aplicação das atividades constituem evidência empírica de primeira ordem para este capítulo. Esta seção sistematiza duas dimensões observadas: o padrão de adesão dos estudantes às atividades e a disponibilidade de material didático como fator de viabilidade metodológica.

5.6.1 Adesão e engajamento

O padrão de adesão observado constitui dado empírico em si. Das quatro atividades disponibilizadas, registraram-se sete respostas ao questionário (seis concentradas na Atividade 1 e uma na Atividade 2) e oito relatórios reflexivos (sete da Atividade 1 e um da Atividade 2), com adesão nula nas Atividades 3 e 4. A distribuição é monotonicamente decrescente ao longo do semestre, com adesão substantiva na primeira atividade, queda abrupta na segunda e ausência

total nas duas últimas, e é consistente com a configuração de incentivo descrita em Seção 5.3: o engajamento inicial existe mas não se sustenta ao longo do semestre sem reforço institucional. Trata-se de evidência relevante sobre as condições de viabilidade pedagógica de atividades práticas em Computação Distribuída no formato extracurricular, retomada na Seção 6.2 junto à recomendação de aplicação plena em formato semestral integrado à carga horária regular.

Figura 6 – Adesão por atividade: número de questionários e relatórios submetidos ao longo do semestre 2026.1.



Fonte: Elaborado pelo autor (2026).

5.6.2 Material didático e viabilidade metodológica

A preparação dos materiais de apoio para a fase pré-atividade da Sala de Aula Invertida (Seção 5.2) revelou um fator de viabilidade metodológica não previsto no planejamento original do arcabouço: a disponibilidade de material didático em diferentes formatos e no idioma da turma. Esse fator manifestou-se de forma concreta na Atividade 2 (Espaço de Tuplas), conforme já indicado em Seção 5.3: a literatura didática em português sobre o modelo Linda, JavaSpaces e Apache River mostrou-se escassa e em geral em nível incompatível com o da disciplina, restringindo o material pré-atividade ao documento de contexto teórico elaborado especificamente para este trabalho.

Diferentemente das restrições físicas tipicamente consideradas no planejamento de atividades práticas (carga horária, infraestrutura computacional, tamanho de turma), a disponibilidade de material em formatos variados é uma restrição associada ao tópico abordado: alguns tópicos da Computação Distribuída possuem ampla cobertura didática em português e em múltiplos formatos (texto, vídeo, áudio), enquanto outros, particularmente os mais especializados ou históricos, contam apenas com referências em inglês e em formato acadêmico denso. Essa observação implica que a escolha de uma metodologia ativa que dependa estruturalmente da preparação prévia do estudante, como a Sala de Aula Invertida, não pode considerar apenas os

princípios pedagógicos e a infraestrutura disponível: também precisa considerar se há, para o tópico específico, material acessível ao perfil de estudante atendido. Esse achado é retomado na interpretação consolidada da Seção 6.2 como um critério adicional a ser incorporado em aplicações futuras do arcabouço.

5.7 LIÇÕES PARA REFINAMENTO DAS ATIVIDADES

Esta seção consolida, em nível operacional, as lições derivadas das três fontes de evidência apresentadas anteriormente: respostas ao questionário (Seção 5.4), análise dos relatórios reflexivos (Seção 5.5) e observações da aplicação (Seção 5.6). As lições aqui registradas referem-se a ajustes pontuais nas atividades e em seus materiais; as direções estratégicas de continuidade do trabalho, como escalonamento, ampliação de metodologias, distribuição do ambiente e disseminação, são tratadas em Seção 6.3.

Atividade 1 – Sockets TCP. (i) Esclarecer no enunciado o nível de exigência sobre estilo e formalismo de código no Nível 2, em resposta à dificuldade reportada por R1 na questão aberta do Bloco 4 (Subseção 5.4.2). (ii) Revisar a forma de apresentação do *template* `relatorio-template.md` para reduzir a confusão com a sintaxe *Markdown* bruta reportada por R3, por exemplo fornecendo pré-visualização HTML lado a lado com a fonte ou um *commit* de referência já renderizado no repositório da atividade. Adicionalmente, incluir nota explícita no *template* sobre o uso correto do separador `|` nas tabelas das seções 1.1 e 1.2, em resposta ao caso de quebra de renderização observado em D2 (Subseção 5.5.2). (iii) Considerar a inclusão de uma breve seção introdutória sobre Docker no material de preparação, alinhada ao interesse explícito de R5 (“Conceitos de Docker”) na questão “o que gostaria de explorar a seguir” e à percepção declarada por R3 de Docker como principal superfície inicial de dificuldade. (iv) Reforçar no enunciado da pergunta 1.3 (rede e contêineres) o conceito de *stack* de rede isolado por contêiner e a resolução de nomes de serviço pela rede *bridge* do Docker, conceito articulado de forma parcial em D2 e D7 (Subseção 5.5.2). (v) Tornar mais explícita a obrigatoriedade do preenchimento textual das modificações no Nível 2 e das seções abertas (“Observações livres” e “Dúvida para a próxima aula”), eventualmente reformulando o *template* para sinalizar que esses campos compõem a entrega e não são opcionais, em resposta à lacuna de Nível 2 observada em D7 e ao não-preenchimento majoritário das seções abertas em A1. (vi) Revisar a formulação das três questões abertas do Bloco 4, possivelmente substituindo a pergunta genérica “o que gostaria de explorar a seguir” por *prompts* mais específicos que provoquem articulação de dúvidas conceituais, em resposta à omissão sistemática observada em Subseção 5.5.2. O caso de D6, que formulou dúvida conceitual substantiva nos termos previstos pelo desenho do instrumento, sugere que a formulação atual funciona para um subconjunto dos respondentes; *prompts* mais específicos têm o objetivo de ampliar a cobertura sem perder a contribuição desse subconjunto.

Atividade 2 – Espaço de Tuplas. (i) **Refinamento iterativo do procedimento experimental do Nível 0 (desacoplamento temporal)**, motivado pelo achado emergente descrito em Subseção 5.5.2: a redação original prescrevia execução conjunta de `reggie`, `javaspaces` e `produtor` seguida de `Ctrl+C`, procedimento que derruba o servidor Outrigger e impede a observação da propriedade que se quer demonstrar. O procedimento foi reescrito em três passos sobre dois terminais distintos: Terminal 1 mantém `reggie` e `javaspaces` ativos continuamente; Terminal 2 executa primeiro `docker compose up produtor` (aguardando seu término) e em seguida `docker compose up consumidor`. Esse procedimento garante que o espaço de tuplas permaneça vivo entre as duas execuções e que o desacoplamento temporal emerja como propriedade observável. Esse ajuste, decorrente diretamente da articulação metacognitiva de D8, ilustra o ciclo de pesquisa baseada em design (*design-based research*) (The Design-Based Research Collective, 2003) previsto em Seção 6.3: a evidência empírica do piloto não apenas alimentou refinamentos de redação, mas expôs limitação estrutural do material que não havia sido detectada no planejamento prévio. (ii) Elaborar material didático complementar em português e em formato acessível, como vídeo curto introdutório e/ou áudio do tipo *podcast*, cobrindo o modelo Linda, a linhagem JavaSpaces–Apache River e o papel do serviço de descoberta, suprindo a lacuna identificada em Subseção 5.6.2.

5.8 SÍNTESE E AMEAÇAS À VALIDADE

Este capítulo descreveu a aplicação empírica das quatro atividades práticas propostas em uma turma de Computação Distribuída da UFSC no semestre 2026.1 e apresentou as evidências obtidas por meio de três fontes complementares: sete respostas ao questionário pós-atividade (seis para a Atividade 1, uma para a Atividade 2), análise dos oito relatórios reflexivos entregues (sete da Atividade 1 e um da Atividade 2) e observações registradas durante o processo de aplicação. O capítulo conseguiu demonstrar a operacionalização efetiva do arcabouço em um contexto real: o ambiente padronizado foi executado pelos estudantes, os instrumentos de avaliação foram aplicados, os artefatos textuais foram produzidos e analisados qualitativamente, e observações sistemáticas sobre a aplicação foram registradas. O capítulo também sinalizou, em caráter preliminar, autoavaliação positiva de ganho conceitual e usabilidade na Atividade 1, e Raw TLX mediano em faixa baixa-moderada (20,8 em escala 0–100). Adicionalmente, a análise dos relatórios produziu um **resultado empírico de natureza distinta**: o relatório D8 (Atividade 2) expôs limitação estrutural do procedimento experimental originalmente prescrito para o experimento de desacoplamento temporal e motivou refinamento iterativo do material da Atividade 2 (Subseção 5.5.2, Seção 5.7), caso concreto que ilustra o ciclo de pesquisa baseada em design previsto em Seção 6.3, no qual a evidência empírica do piloto não apenas alimenta ajustes de redação, mas detecta defeitos de design não antecipados no planejamento. Esses resultados, contudo, **não** constituem evidência estatística inferencial sobre a eficácia pedagógica das atividades: a comparação entre atividades, a análise longitudinal da

carga cognitiva e da usabilidade percebida e a estimativa estável de consistência interna dos blocos exigem amostras maiores, procedimentos previstos no desenho original do instrumento e que permanecem inaplicáveis no tamanho amostral obtido.

Ameaças relevantes à validade dos resultados aqui apresentados são enunciadas explicitamente. A **ameaça primária** é o tamanho amostral reduzido das respostas e dos relatórios, que impede generalização das observações qualitativas para a turma ou para a população de estudantes da disciplina. A **autosseleção** dos respondentes constitui ameaça correlata: aqueles que efetivamente participaram não constituem amostra aleatória da turma, e a hipótese mais plausível é que apresentem perfil de maior engajamento ou maior interesse específico nos tópicos abordados, viesando positivamente as percepções autorreportadas e a qualidade dos relatórios. A **análise qualitativa** dos relatórios é suscetível ao viés do pesquisador, mitigada pela comparação sistemática com o gabarito específico de cada atividade mas não eliminada. Por fim, a **ausência de controle ou *baseline***, dado que não foi aplicada uma versão das atividades sem o arcabouço para comparação, nem foram aplicadas as atividades em outra turma para contraste, impede atribuição causal das observações ao arcabouço proposto e não a outros fatores contextuais.

6 CONSIDERAÇÕES FINAIS

Este trabalho propôs um arcabouço didático-metodológico para o ensino de Computação Distribuída e aplicou um piloto na turma da disciplina de Computação Distribuída da UFSC, semestre 2026.1. Este capítulo retoma os cinco objetivos específicos enunciados em Capítulo 1 para registrar, de forma direta, o que foi entregue para cada um e qual evidência sustenta esse cumprimento; em seguida, discute as limitações observadas durante a aplicação e organiza as direções de continuidade da pesquisa.

6.1 SÍNTESE DAS CONTRIBUIÇÕES POR OBJETIVO ESPECÍFICO

Os parágrafos a seguir percorrem os cinco objetivos específicos do trabalho, indicando onde cada um foi efetivamente cumprido e qual contribuição concreta dele decorre.

6.1.1 Objetivo 1 – Conduzir revisão exploratória da literatura.

Este objetivo foi cumprido em dois capítulos complementares, a partir de uma revisão exploratória conduzida nas bases Google Scholar e IEEE Xplore, complementadas por anais do *ACM Special Interest Group on Computer Science Education* (SIGCSE), conforme descrito em Capítulo 3. O Capítulo 2 apresenta seis metodologias ativas (Aprendizagem Baseada em Problemas, Aprendizagem Baseada em Projetos, Aprendizagem Baseada em Equipes, Sala de Aula Invertida, Instrução pelos Pares e Aprendizagem Baseada em Investigação), três estratégias pedagógicas (Andaimagem, *Coding Dojo* e Programação em Pares) e três tecnologias educacionais (Contêineres, Máquinas Virtuais e *Scripts*), cada uma com definição, contexto histórico, processo de implementação, características e evidências empíricas obtidas em estudos prévios. O Capítulo 3 revisa quatro eixos temáticos de trabalhos correlatos sobre ensino de Computação Distribuída e Paralela, sintetizando ao final o conjunto de barreiras recorrentes enunciadas em Capítulo 1 que sustenta a motivação central deste trabalho. Em função do escopo desta monografia, não se adotou protocolo de revisão sistemática nem de mapeamento sistemático da literatura.

6.1.2 Objetivo 2 – Propor uma matriz metodologias × estratégias × tópicos × atividades.

A matriz conceitual é apresentada na Seção 4.1 e sintetizada na Tabela 1 (Capítulo 4). Ela correlaciona, em seis colunas, os quatro tópicos centrais (comunicação assíncrona via *sockets* TCP, comunicação assíncrona via memória compartilhada, sincronização de relógios, e tolerância a falhas com consistência de réplicas) à disciplina de Computação Distribuída da UFSC (INE5418), em cujo conteúdo curricular os tópicos são abordados e na qual o piloto foi aplicado, às metodologias (Sala de Aula Invertida e Aprendizagem Baseada em Investigação), às estratégias (Andaimagem como transversal e Programação em Pares como opcional),

às tecnologias (Contêineres com *scripts* para três das quatro atividades, Máquinas Virtuais para a Atividade 3) e à atividade prática que materializa cada combinação. A matriz é a contribuição articuladora do trabalho: explicita os critérios pelos quais cada combinação foi escolhida e fornece um *template* de raciocínio que pode ser estendido a tópicos adicionais em iterações futuras.

6.1.3 Objetivo 3 – Prototipar ambiente padronizado e conjunto de atividades práticas.

Este objetivo é o de maior densidade material e foi cumprido em duas frentes interligadas. As quatro atividades práticas, Cliente/Servidor Eco com *sockets* TCP (Subseção 4.2.2), Espaço de Tuplas com Apache River (Subseção 4.2.3), Sincronização de Relógios com NTP/chrony (Subseção 4.2.4) e Consenso com Raft acompanhado de *dashboard* ao vivo (Subseção 4.2.5), estão integralmente desenvolvidas, com código, *Dockerfile*s ou `Makefile`, material de preparação para a fase pré-atividade e gabarito de relatório reflexivo, sendo cada atividade mantida em repositório público próprio no GitHub. O ambiente padronizado que viabiliza sua execução é descrito na Seção 4.3, organizando-se em três camadas conceituais, virtualização (Docker para A1, A2, A4; Multipass/KVM para A3), orquestração (Docker Compose e `Makefile` declarativo) e automação/instrumentação (*scripts*, *loggers* embarcados e *dashboard* SSE), alinhadas aos três princípios norteadores de reprodutibilidade, portabilidade e baixa barreira de entrada.

6.1.4 Objetivo 4 – Desenvolver instrumentos de avaliação.

Os dois instrumentos foram detalhados em Seção 5.2. O **questionário pós-atividade** combina quatro blocos: Aprendizagem Percebida (cinco itens Likert alinhados aos objetivos específicos de cada atividade), Usabilidade do Ambiente (quatro itens fixos fundamentados no *Technology Acceptance Model* (Davis, 1989)), Carga de Trabalho (seis itens fixos correspondentes às subescalas do NASA-TLX (Hart; Staveland, 1988; Hart, 2006)) e Questões Abertas (três itens qualitativos). O **relatório reflexivo** acompanha cada atividade na forma de *template Markdown* cuja estrutura espelha os três níveis de andaimagem da atividade e é analisado por comparação contra um gabarito específico mantido no repositório. A triangulação entre dado autorreportado (questionário) e artefato textual (relatório) constitui a estratégia de avaliação adotada e é, em si, uma contribuição replicável a futuras ofertas da disciplina e a outros docentes.

6.1.5 Objetivo 5 – Aplicar em turmas reais, analisar e identificar refinamentos.

A aplicação em uma turma de Computação Distribuída da UFSC no semestre 2026.1 é descrita no Capítulo 5: as quatro atividades foram disponibilizadas ao longo do semestre em formato predominantemente extracurricular (Seção 5.3), os instrumentos de avaliação foram efetivamente operacionalizados (Seção 5.4, Seção 5.5) e observações sistemáticas sobre o

processo foram registradas (Seção 5.6). O tamanho amostral obtido, sete respostas ao questionário e oito relatórios reflexivos, concentrados nas Atividades 1 e 2 e com adesão nula nas Atividades 3 e 4, inviabilizou os procedimentos inferenciais originalmente previstos, conforme discutido em Seção 5.8, mas viabilizou uma leitura qualitativa dos artefatos e a identificação de refinamentos operacionais por atividade (Seção 5.7). O cumprimento parcial deste objetivo é, em si, evidência empírica relevante sobre as condições de viabilidade pedagógica em formato extracurricular, retomada na Seção 6.2.

A integração entre os cinco objetivos sustenta a hipótese central enunciada em Capítulo 1. A fundamentação teórica dessa hipótese está consolidada (Objetivos 1 e 2); a operacionalização concreta está disponível e reprodutível (Objetivos 3 e 4); a validação empírica está iniciada porém condicionada pelo tamanho amostral (Objetivo 5) e demanda continuidade nos termos discutidos em Seção 6.3.

6.2 LIMITAÇÕES E LIÇÕES APRENDIDAS

A aplicação descrita no Capítulo 5 expõe duas limitações cujas lições generalizam para aplicações futuras do arcabouço, para além do contexto específico do piloto.

A primeira – a escassez de material didático em português e em formatos variados para o tópico de Espaço de Tuplas (Subseção 5.6.2), revelou um fator de viabilidade não previsto no planejamento inicial: a *disponibilidade de material didático no idioma da turma e em formatos variados* influencia diretamente quais metodologias ativas são realmente viáveis, ao lado das restrições físicas (carga horária, infraestrutura computacional, tamanho de turma) tradicionalmente consideradas. Metodologias que pressupõem preparação prévia estruturada, Sala de Aula Invertida em particular, ficam efetivamente bloqueadas para tópicos cuja literatura didática acessível não existe no idioma da turma. Esse critério deve ser incorporado, ao lado dos demais, à seleção de metodologia em iterações futuras.

A segunda – baixa adesão sob formato extracurricular de curta duração, evidenciada pelo tamanho amostral obtido (Subseção 5.6.1), reforça a hipótese de que a aplicação plena do arcabouço requer inserção no contexto regular da disciplina, em formato semestral, com as atividades integradas ao percurso avaliativo. Vincular a realização das atividades a componente da nota da disciplina é a estratégia mais direta para criar incentivo equivalente ao do tempo curricular regular, e é objeto da frente de continuidade descrita em Seção 6.3.

6.3 TRABALHOS FUTUROS

A aplicação do arcabouço nas turmas da UFSC, descrita no Capítulo 5, evidenciou tanto a viabilidade do conjunto proposto quanto pontos cujo aprofundamento ultrapassa o escopo deste trabalho. Esta seção organiza os próximos passos identificados em cinco frentes complementares.

1. **Escalonamento da aplicação em sala**: replicar o piloto em ofertas subsequentes da disciplina, ampliando o tamanho amostral e permitindo análises estatísticas mais robustas, incluindo comparações longitudinais entre semestres e entre cortes de estudantes com diferentes níveis de familiaridade com Linux, contêineres e linhas de comando. Um tamanho amostral maior também viabiliza a estratificação por perfil de estudante e a aplicação de testes estatísticos com maior poder.

2. **Refinamento das atividades a partir dos dados coletados**: incorporar sistematicamente as categorias temáticas emergidas do Bloco 4 do questionário e os equívocos conceituais recorrentes identificados nos relatórios, ajustando enunciados, materiais de preparação e nível de andaimagem das atividades. Esse refinamento alinha-se aos princípios de pesquisa baseada em design (*design-based research*), que enfatizam ciclos sucessivos de teste e ajuste em contextos autênticos de sala de aula (The Design-Based Research Collective, 2003).

3. **Ampliação das metodologias e estratégias contempladas**: durante a aplicação observou-se que algumas metodologias e estratégias inicialmente consideradas mostraram-se inviáveis no formato extracurricular de curta duração utilizado. A Instrução pelos Pares, por exemplo, demanda a presença de tutores que acompanhem os estudantes ao longo das atividades, o que tipicamente exige tempo de aula maior que o disponível neste piloto. A Aprendizagem Baseada em Projetos requer um período mais longo de desenvolvimento, incompatível com a janela utilizada. A Aprendizagem Baseada em Equipes seria significativamente favorecida pela disponibilização de tempo em sala para organização de grupos, discussão e orientação direta do professor. Estratégias como Coding Dojo e Programação em Pares também seriam mais efetivas se executadas em tempo dedicado de aula. Em trabalhos futuros, propõe-se a aplicação do arcabouço em formato semestral pleno, no qual essas metodologias e estratégias poderão ser efetivamente avaliadas.

4. **Distribuição e versionamento do ambiente**: publicar as imagens Docker das atividades em registro público (conforme apontado no Seção 4.3), eliminando tempos de *build* locais e permitindo que estudantes iniciem o ambiente com comandos ainda mais sucintos. Adicionalmente, padronizar a estrutura dos repositórios de cada atividade e consolidar a documentação como *site* navegável para facilitar a adoção por outros docentes.

5. **Aplicação em disciplinas correlatas da UFSC**: o escopo inicial do arcabouço, conforme delimitado em Capítulo 4, contemplava as disciplinas de Computação Distribuída (INE5418 e INE5625), Programação Paralela e Distribuída (INE5645), Programação Concorrente (INE5410) e Sistemas Operacionais (INE5412, INE5424 e INE5611), oferecidas nos cursos de Ciência da Computação e Sistemas de Informação. O piloto descrito no Capítulo 5, contudo, restringiu-se a uma turma de Computação Distribuída, de modo que a viabilidade pedagógica das atividades nas demais disciplinas permanece em aberto. Propõe-se, em continuidade, estender a aplicação a essas disciplinas, considerando que

cada uma enfatiza recortes distintos dos tópicos centrais. Atividades semelhantes também podem ser propostas, ampliando o conjunto inicial.

6. **Disseminação para outras instituições e contextos**: empacotar o arcabouço (matriz conceitual, conjunto de atividades, ambiente padronizado e instrumentos de avaliação) como guia de adoção replicável em outras instituições de ensino superior. A disseminação inclui submissão a fóruns de educação em Computação (SBC – WEI, ENINED, periódicos da área) e diálogo com docentes de disciplinas correlatas em outras instituições para identificar adaptações necessárias a diferentes contextos curriculares.

REFERÊNCIAS

ARROYO, D. Teaching parallel and distributed computing. **IEEE Distributed Systems Online**, 2013.

AutomatedLab Project. **AutomatedLab: Automated Lab Environment Setup**. 2024. https://automatedlab.org/. Accessed: 2024-12-04.

BECK, K. **Extreme Programming Explained: Embrace Change**. Reading, MA: Addison-Wesley Longman, 2000. ISBN 0201616416.

BERBEL, N. A. N. As metodologias ativas e a promoção da autonomia de estudantes. **Semina: Ciências Sociais e Humanas**, Universidade Estadual de Londrina, v. 32, n. 1, p. 25–40, 2011.

Conselho Nacional de Desenvolvimento Científico e Tecnológico. **Portaria nº 2.664, de 6 de março de 2026: Institui a Política de Integridade na Atividade Científica do CNPq**. 2026. https://www.gov.br/cnpq/pt-br/assuntos/noticias/cnpq-em-acao/cnpq-publica-portaria-que-institui-politica-de-integridade-na-atividade-cientifica. Acesso em: 15 jun. 2026.

CONSTANTINE, L. L. **Constantine on Peopleware**. Englewood Cliffs, NJ: Yourdon Press, 1995.

CONTE, D. J. et al. Teaching parallel programming for beginners in computer science. In: **Proceedings of the 2020 IEEE Frontiers in Education Conference (FIE)**. [S.l.]: IEEE, 2020. p. 1–9.

CRISTIAN, F. Probabilistic clock synchronization. **Distributed Computing**, Springer, v. 3, n. 3, p. 146–158, 1989.

CROUCH, C. H.; MAZUR, E. Peer instruction: Ten years of experience and results. **American Journal of Physics**, American Association of Physics Teachers, v. 69, n. 9, p. 970–977, 2001.

DAVIS, F. D. Perceived usefulness, perceived ease of use, and user acceptance of information technology. **MIS Quarterly**, v. 13, n. 3, p. 319–340, 1989.

ELLIS, M. et al. Teaching rigorous distributed systems with efficient model checking. In: **Proceedings of the ACM European Conference on Computer Systems**. [S.l.: s.n.], 2019.

FRANKE, D. F. et al. **Network Time Security for the Network Time Protocol**. 2020. RFC 8915, Internet Engineering Task Force. Disponível em: https://www.rfc-editor.org/rfc/rfc8915.

GIL, A. C. **Como Elaborar Projetos de Pesquisa**. 4. ed. São Paulo: Atlas, 2002.

GUSELLA, R.; ZATTI, S. The accuracy of the clock synchronization achieved by TEMPO in Berkeley UNIX 4.3BSD. In: . [S.l.]: IEEE, 1989. v. 15, n. 7, p. 847–853.

HART, S. G. NASA-Task Load Index (NASA-TLX): 20 years later. In: **Proceedings of the Human Factors and Ergonomics Society Annual Meeting**. [S.l.]: SAGE Publications, 2006. v. 50, n. 9, p. 904–908.

HART, S. G.; STAVELAND, L. E. Development of NASA-TLX (Task Load Index): Results of empirical and theoretical research. In: HANCOCK, P. A.; MESHKATI, N. (Ed.). **Human Mental Workload**. Amsterdam: Elsevier, 1988. p. 139–183.

KILPATRICK, W. H. The project method: The use of the purposeful act in the educative process. **Teachers College Record**, Teachers College, Columbia University, v. 19, n. 4, p. 319–335, 1918.

KOKOTSAKI, D.; MENZIES, V.; WIGGINS, A. Project-based learning: A review of the literature. **Improving Schools**, SAGE Publications, v. 19, n. 3, p. 267–277, 2016.

LAADAN, O.; NIEH, J.; VIENNOT, N. Teaching operating systems using virtual appliances and distributed version control. In: **Proceedings of the 41st ACM Technical Symposium on Computer Science Education**. New York, USA: ACM, 2010. (SIGCSE '10), p. 480–484.

LAGE, M. J.; PLATT, G. J.; TREGLIA, M. Inverting the classroom: A gateway to creating an inclusive learning environment. **Journal of Economic Education**, Taylor & Francis, v. 31, n. 1, p. 30–43, 2000.

LAZONDER, A. W.; HARMSEN, R. Meta-analysis of inquiry-based learning: Effects of guidance. **Review of Educational Research**, SAGE Publications, v. 86, n. 3, p. 681–718, 2016.

LICHVAR, M. **chrony: Versatile Implementation of the Network Time Protocol**. 2024. https://chrony-project.org/. Accessed: 2026-05-31.

LIKERT, R. A technique for the measurement of attitudes. **Archives of Psychology**, v. 22, n. 140, p. 1–55, 1932.

MALAN, D. J. Standardizing students' programming environments with docker containers: Using visual studio code in the cloud with github codespaces. In: **Proceedings of the 27th ACM Conference on Innovation and Technology in Computer Science Education**. New York, USA: ACM, 2022. (ITiCSE '22, v. 1), p. 191–197.

MANN, H. B.; WHITNEY, D. R. On a test of whether one of two random variables is stochastically larger than the other. **The Annals of Mathematical Statistics**, Institute of Mathematical Statistics, v. 18, n. 1, p. 50–60, 1947.

MANOHARAN, S.; YE, X. Teaching introductory parallel programming using two-player online games. In: **Proceedings of the 46th MIPRO Conference on Computers and Informatics**. [S.l.]: IEEE, 2023. p. 1–6.

MERKEL, D. Docker: Lightweight linux containers for consistent development and deployment. **Linux Journal**, Belltown Media, v. 2014, n. 239, p. 2, 2014.

MILLS, D. L. Network time protocol (NTP). **RFC 958, Internet Engineering Task Force**, 1985. Disponível em: https://www.rfc-editor.org/rfc/rfc958.

MORÁN, J. et al. Mudando a educação com metodologias ativas. **Coleção mídias contemporâneas. Convergências midiáticas, educação e cidadania: aproximações jovens**, v. 2, n. 1, p. 15–33, 2015.

NOSEK, J. T. The case for collaborative programming. **Communications of the ACM**, ACM, v. 41, n. 3, p. 105–108, 1998.

O'CONNOR, T. et al. PWN lessons made easy with docker: Toward an undergraduate vulnerability research cybersecurity class. In: **Proceedings of the 55th ACM Technical Symposium on Computer Science Education**. New York, USA: ACM, 2024. (SIGCSE '24, v. 1), p. 998–1004.

PALICHERLA, A.; ZHANG, T.; PORTER, D. E. Teaching virtualization by building a hypervisor. In: **Proceedings of the 46th ACM Technical Symposium on Computer Science Education**. New York, USA: ACM, 2015. (SIGCSE '15), p. 141–146.

PARMELEE, D. et al. Team-based learning: a practical guide: Amee guide no. 65. **Medical teacher**, Taylor & Francis, v. 34, n. 5, p. e275–e287, 2012.

PECOTCHE, C. B. G. **Introdução ao Conhecimento Logosófico**. São Paulo: Editora Logosófica, 1951. Publicado sob o pseudônimo Raumsol. Disponível em: https://logosofia.org.br/livros/introducao-ao-conhecimento-logosofico/.

PEDASTE, M. et al. Phases of inquiry-based learning: Definitions and the inquiry cycle. **Educational Research Review**, Elsevier, v. 14, p. 47–61, 2015.

SATO, D. T.; CORBUCCI, H.; BRAVO, M. V. Coding dojo: An environment for learning and sharing agile practices. In: **Agile 2008 Conference**. [S.l.]: IEEE, 2008. p. 459–464.

SMITH, J. E.; NAIR, R. The architecture of virtual machines. **Computer**, IEEE, v. 38, n. 5, p. 32–38, 2005.

TAVAKOL, M.; DENNICK, R. Making sense of Cronbach's alpha. **International Journal of Medical Education**, v. 2, p. 53–55, 2011.

The Design-Based Research Collective. Design-based research: An emerging paradigm for educational inquiry. **Educational Researcher**, v. 32, n. 1, p. 5–8, 2003.

van de Pol, J.; VOLMAN, M.; BEISHUIZEN, J. Scaffolding in teacher–student interaction: A decade of research. **Educational Psychology Review**, Springer, v. 22, n. 3, p. 271–296, 2010.

WANG, H. **Novice-Friendly Parallel Programming Exercises**. Dissertação (4th Year Project Report) — University of Edinburgh, Edinburgh, UK, 2023.

WILLIAMS, L. et al. Strengthening the case for pair programming. **IEEE Software**, IEEE, v. 17, n. 4, p. 19–25, 2000.

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

WOOD, D. F. Problem based learning. **BMJ**, BMJ Publishing Group Ltd, v. 326, n. 7384, p. 328–330, 2003.

Apêndices

APÊNDICE A – ARTIGO CIENTÍFICO NO FORMATO SBC

Este apêndice reproduz, no formato da Sociedade Brasileira de Computação (SBC), um artigo científico que sintetiza o presente trabalho. O artigo é mantido e compilado de forma independente e incluído a seguir na íntegra.

Um Arcabouço Didático-Metodológico para o Ensino de Computação Distribuída

João Pedro Schmidt Cordeiro[1], **Odorico Machado Mendizabal**[1]

[1]Departamento de Informática e Estatística, Universidade Federal de Santa Catarina (UFSC)
Florianópolis, SC, Brasil

jschmidtcordeiro@gmail.com, odorico.mendizabal@ufsc.br

Abstract. *Teaching distributed computing is hindered by environment heterogeneity, the intrinsic complexity of topics such as synchronization, consistency and fault tolerance, and the high cognitive load of setup and debugging. This paper presents a didactic-methodological framework that integrates three complementary dimensions: teaching methodologies, pedagogical strategies and educational technologies, organized around four core topics. The framework is materialized in a conceptual matrix, four hands-on activities and a standardized, reproducible environment based on containers and lightweight virtual machines. A pilot applied in a Distributed Computing course at UFSC provides preliminary evidence of operational feasibility and perceived learning, and identifies refinements following a design-based research cycle.*

Resumo. *O ensino de Computação Distribuída enfrenta barreiras como a heterogeneidade de ambientes, a complexidade intrínseca de tópicos como sincronização, consistência e tolerância a falhas, e a alta carga cognitiva de configuração e depuração. Este artigo apresenta um arcabouço didático-metodológico que integra três dimensões complementares: metodologias de ensino, estratégias pedagógicas e tecnologias educacionais, organizadas em torno de quatro tópicos centrais. O arcabouço materializa-se em uma matriz conceitual, quatro atividades práticas e um ambiente padronizado e reprodutível baseado em contêineres e máquinas virtuais leves. Um piloto aplicado na disciplina de Computação Distribuída da UFSC fornece evidência preliminar de viabilidade operacional e de aprendizagem percebida, e identifica refinamentos segundo um ciclo de pesquisa baseada em design.*

1. Introdução

O avanço da computação e a ubiquidade de sistemas distribuídos exigem profissionais com base conceitual sólida e fluência prática. A literatura educacional recomenda distribuir os conteúdos de Paralelismo e Distribuição ao longo da graduação, apoiados por práticas que promovam compreensão e aplicação [Arroyo 2013]. Contudo, persistem barreiras relevantes: heterogeneidade de ambientes (hardware e sistemas operacionais), complexidade de tópicos como assincronia, sincronização, consistência e tolerância a falhas, e alta carga cognitiva para configuração e depuração. Ellis Michael et al. [Ellis et al. 2019] destacam a dificuldade de iniciantes em produzir código correto em sistemas distribuídos; embora técnicas como verificação de modelos e depuração avançada sejam úteis, costumam ser onerosas em contextos educacionais.

Diante dessa problemática, este trabalho propõe um arcabouço didático-metodológico organizado em três frentes complementares: (i) **metodologias de ensino**, que definem os princípios pedagógicos orientadores da construção do conhecimento, colocando o estudante como protagonista; (ii) **estratégias de ensino**, ações e recursos específicos que operacionalizam as metodologias; e (iii) **tecnologias educacionais**, ferramentas viabilizadoras que asseguram reprodutibilidade e praticidade nos laboratórios. A hipótese central é que a combinação intencional dessas três dimensões pode reduzir as barreiras identificadas e promover aprendizagem mais efetiva em Computação Distribuída, permitindo que os estudantes concentrem-se nos conceitos fundamentais enquanto desenvolvem competências técnicas aplicáveis.

O objetivo geral é definir, instrumentar e validar empiricamente esse arcabouço. Quanto ao delineamento, o trabalho caracteriza-se como pesquisa aplicada, com objetivos exploratório-descritivos e abordagem predominantemente qualitativa [Gil 2002], alinhada aos princípios da pesquisa baseada em design (*design-based research*), que preconiza ciclos sucessivos de projeto, intervenção em contexto autêntico, análise e ajuste [The Design-Based Research Collective 2003]. Este artigo condensa a monografia de conclusão de curso homônima e está organizado como segue: a Seção 2 resume a fundamentação teórica e os trabalhos relacionados; a Seção 3 apresenta o arcabouço proposto; a Seção 4 relata a aplicação piloto na UFSC; e a Seção 5 conclui.

2. Fundamentação e Trabalhos Relacionados

O arcabouço apoia-se em três corpos de literatura. Em **metodologias de ensino ativas**, consideram-se Aprendizagem Baseada em Problemas (*Problem-Based Learning*, PBL) [Wood 2003], Aprendizagem Baseada em Projetos (*Project-Based Learning*, PjBL) [Kokotsaki et al. 2016], Aprendizagem Baseada em Equipes (*Team-Based Learning*, TBL) [Parmelee et al. 2012], Sala de Aula Invertida (*Flipped Classroom*) [Lage et al. 2000], Instrução pelos Pares (*Peer Instruction*) [Crouch and Mazur 2001] e Aprendizagem Baseada em Investigação (*Inquiry-Based Learning*, IBL) [Pedaste et al. 2015, Lazonder and Harmsen 2016]. Em **estratégias pedagógicas**, destacam-se a Andaimagem (*scaffolding*) [Wood et al. 1976, van de Pol et al. 2010], o *Coding Dojo* [Sato et al. 2008] e a Programação em Pares [Williams et al. 2000]. Em **tecnologias educacionais**, contêineres Docker [Merkel 2014, Malan 2022], máquinas virtuais [Smith and Nair 2005, Laadan et al. 2010] e *scripts* de automação [AutomatedLab Project 2024] viabilizam ambientes reprodutíveis de laboratório.

No ensino específico de Computação Paralela e Distribuída, trabalhos correlatos abordam a introdução de programação paralela para iniciantes [Conte et al. 2020], o uso de jogos [Manoharan and Ye 2023] e exercícios amigáveis a novatos [Wang 2023], além de infraestruturas de laboratório baseadas em contêineres [O'Connor et al. 2024, Malan 2022]. A síntese dessa literatura revela uma lacuna recorrente: a ausência de um arranjo que articule, de forma intencional e reprodutível, metodologia, estratégia e tecnologia em torno dos tópicos centrais da área. É essa lacuna que o presente trabalho endereça.

3. Arcabouço Proposto

As propostas concentram-se em quatro tópicos centrais das disciplinas foco: **comunicação assíncrona entre processos**, **sincronização de relógios**, **tolerância a falhas e consenso** e **consistência de réplicas**. Cada tópico é operacionalizado por uma atividade prática.

3.1. Matriz Conceitual

A contribuição articuladora do trabalho é uma matriz que sistematiza as correlações entre metodologias, estratégias, tecnologias, tópicos e atividades (Tabela 1). As quatro atividades compartilham a mesma combinação metodológica: Sala de Aula Invertida [Lage et al. 2000], que desloca a exposição conceitual para uma fase pré-atividade individual, e Aprendizagem Baseada em Investigação [Pedaste et al. 2015], que estrutura o trabalho presencial em ciclos de execução, observação, formulação de hipóteses e validação. As estratégias também são fixas: Andaimagem como estratégia transversal, materializada por uma progressão em três níveis (Nível 0, executar o sistema pronto; Nível 1, inspecionar e responder questões conceituais; Nível 2, modificar código ou topologia) [van de Pol et al. 2010], e Programação em Pares como estratégia opcional [Williams et al. 2000]. Manter metodologia e estratégia constantes, variando apenas o tópico e a tecnologia, preserva a comparabilidade dos instrumentos de avaliação e reduz o custo de reaprender a dinâmica a cada nova atividade.

Tabela 1. Correlações entre tópicos, metodologias, estratégias, tecnologias e atividades propostas.

Tópico	Metodologias	Estratégias	Tecnologias	Ativ.
Comunicação assíncrona (*sockets* TCP)	Sala Invertida, IBL	Andaimagem, Pair Prog. (opc.)	Contêineres, *Scripts*	A1
Comunicação assíncrona (memória compart.)	Sala Invertida, IBL	Andaimagem, Pair Prog. (opc.)	Contêineres, *Scripts*	A2
Sincronização de relógios	Sala Invertida, IBL	Andaimagem, Pair Prog. (opc.)	VMs (Multipass), *Scripts*	A3
Tolerância a falhas e consistência de réplicas	Sala Invertida, IBL	Andaimagem, Pair Prog. (opc.)	Contêineres, *Scripts*	A4

3.2. Atividades Práticas

Atividade 1: Cliente/Servidor Eco com *Sockets* TCP. Introduz comunicação entre processos. Cliente e servidor executam em contêineres distintos sobre uma rede interna nomeada, tornando concreta a separação entre processos comunicantes mesmo em um único hospedeiro. Os estudantes progridem de executar o sistema (`docker compose up --build`), inspecionar as chamadas de *socket* e testar hipóteses (por exemplo, remover `depends_on` e observar a falha de conexão), até modificar o servidor e reconstruir a imagem.

Atividade 2: Espaço de Tuplas com Apache River. Aborda comunicação assíncrona por memória associativa compartilhada, com ênfase em desacoplamento

espacial e temporal. Cinco serviços em contêineres (descoberta `reggie`, servidor `javaspaces`, produtor, consumidor e monitor) materializam a complexidade de um *middleware* de coordenação. Experimentos evidenciam a persistência de tuplas entre execuções (desacoplamento temporal), o bloqueio de `take`/`read` e a escalabilidade horizontal via `--scale consumidor=2`, conectando o modelo Linda a sistemas modernos de mensageria.

Atividade 3: Sincronização de Relógios com NTP/chrony. Rompe deliberadamente com o padrão e adota **máquinas virtuais** leves provisionadas via Multipass sobre KVM/QEMU: contêineres compartilham o relógio do *kernel* hospedeiro, impossibilitando observar o desalinhamento que a atividade investiga. Quatro nós com *drifts* controlados permitem observar empiricamente a convergência sob NTP, medir *offset*, RTT, dispersão e *stratum*, e contrastar as estratégias de correção *step* e *slew*, aplicando o algoritmo de Cristian [Cristian 1989] no cálculo de *offset*.

Atividade 4: Consenso com Raft e Visualizador ao Vivo. Aborda tolerância a falhas e consenso distribuído sobre um cluster real construído com a biblioteca `hashicorp/raft`. Um *dashboard* web alimentado por *Server-Sent Events* torna visível cada transição de papel, incremento de *term* e replicação de log. O *heartbeat* ampliado (`RAFT_HEARTBEAT_MS=5000`) torna cada transição humanamente observável. Os estudantes provocam eleições, partições de rede e reconciliação de log, observando diretamente as garantias de segurança do Raft, e escalam o cluster de três para cinco nós editando o `docker-compose.yml`.

3.3. Ambiente Padronizado

O ambiente fundamenta-se em três princípios: **reprodutibilidade** (todos replicam ambientes idênticos independentemente da configuração local), **portabilidade** (funciona em *laptops* modestos e em Windows, macOS e Linux) e **baixa barreira de entrada**. Organiza-se em três camadas: virtualização (Docker para A1, A2 e A4; Multipass/KVM para A3), orquestração (Docker Compose e `Makefile` declarativo) e automação/instrumentação (*scripts*, *loggers* embarcados e o *dashboard* SSE do Raft). A progressão de andaimagem materializa o princípio de *fading*: as primeiras atividades fornecem ambientes altamente estruturados, e o suporte se desvanece gradualmente até que o estudante assuma a definição da topologia [Wood et al. 1976]. Todo o material reside em repositórios Git públicos, um por atividade.

4. Aplicação Piloto na UFSC

O arcabouço foi aplicado no semestre 2026.1 em uma turma da disciplina de Computação Distribuída (INE5418) da UFSC. As quatro atividades foram disponibilizadas em sequência, em formato predominantemente extracurricular, executadas nos próprios equipamentos dos estudantes.

4.1. Instrumentos de Avaliação

A avaliação apoia-se em dois instrumentos complementares por atividade. O **questionário pós-atividade** combina quatro blocos: Aprendizagem Percebida (cinco itens Likert [Likert 1932] alinhados aos objetivos de cada atividade), Usabilidade do Ambiente (quatro itens fixos baseados no *Technology Acceptance Model* [Davis 1989]), Carga de

Trabalho (seis subescalas do NASA-TLX [Hart and Staveland 1988, Hart 2006], agregadas no *Raw TLX*) e Questões Abertas. O **relatório reflexivo**, em *template Markdown* que espelha os três níveis de andaimagem, é analisado por comparação com um gabarito específico. A triangulação entre dado autorreportado e artefato textual fortalece a validade das conclusões. Dada a natureza ordinal das escalas, o tratamento é descritivo (mediana e intervalo interquartílico), e a confiabilidade interna é verificada pelo alfa de Cronbach [Tavakol and Dennick 2011].

4.2. Resultados

Ao final do prazo de coleta, obtiveram-se sete respostas ao questionário (seis da Atividade 1, uma da Atividade 2) e oito relatórios reflexivos (sete da Atividade 1, um da Atividade 2), com adesão nula nas Atividades 3 e 4. Esse tamanho amostral inviabiliza os procedimentos inferenciais originalmente previstos (teste de Mann-Whitney [Mann and Whitney 1947], análise longitudinal), de modo que os resultados constituem **sinalização preliminar**, não evidência conclusiva.

Para a Atividade 1 ($n = 6$), a Aprendizagem Percebida apresentou medianas entre 4,5 e 5, a Usabilidade entre 4 e 5, e o *Raw TLX* mediano foi 20,8 (escala 0–100), com a Demanda Mental como principal vetor de carga e valores nulos de Frustração e pressão temporal, perfil coerente com o desenho da atividade como primeira prática, de escopo restrito. Os coeficientes alfa situaram-se abaixo de 0,70 (0,31; 0,58; 0,61), resultado condicionado pelo n reduzido e por forte efeito de teto.

A análise dos relatórios produziu um achado de natureza distinta: o relatório da Atividade 2 expôs um **defeito de *design* no procedimento experimental** da própria atividade. O experimento de desacoplamento temporal prescrevia derrubar produtor e servidor simultaneamente (via `Ctrl+C`), eliminando o estado em memória que deveria persistir. O estudante detectou a contradição entre teoria e evidência e articulou a conclusão correta sobre o pré-requisito de persistência. O procedimento foi então reescrito em dois terminais, mantendo o espaço de tuplas vivo entre execuções. Esse caso ilustra diretamente o ciclo de pesquisa baseada em design [The Design-Based Research Collective 2003]: a evidência do piloto não apenas alimentou refinamentos de redação, mas expôs uma limitação estrutural não antecipada no planejamento.

4.3. Ameaças à Validade

As ameaças principais são o tamanho amostral reduzido, a autosseleção dos respondentes (perfil provavelmente mais engajado), o viés do pesquisador na análise qualitativa (mitigado pelo gabarito) e a ausência de *baseline* para atribuição causal. A adesão monotonicamente decrescente ao longo do semestre é, ela própria, evidência sobre a viabilidade pedagógica de atividades práticas no formato extracurricular, reforçando a recomendação de aplicação plena, integrada à carga horária regular.

5. Considerações Finais

Este trabalho apresentou um arcabouço didático-metodológico para o ensino de Computação Distribuída, integrando metodologias de ensino, estratégias pedagógicas e tecnologias educacionais em torno de quatro tópicos centrais. As contribuições concretas são a matriz conceitual articuladora, quatro atividades práticas integralmente desenvolvidas e mantidas em repositórios públicos, um ambiente padronizado reprodutível e um

par de instrumentos de avaliação triangulados. A aplicação piloto na UFSC demonstrou a operacionalização efetiva do arcabouço em contexto real e forneceu sinalização preliminar positiva de aprendizagem percebida e usabilidade, ainda que sem poder estatístico inferencial dado o tamanho amostral.

Como trabalhos futuros, destacam-se o escalonamento da aplicação em ofertas subsequentes para ampliar a amostra, o refinamento das atividades a partir dos dados coletados, a ampliação das metodologias avaliadas em formato semestral pleno, a publicação das imagens Docker em registro público e a disseminação do arcabouço a disciplinas correlatas e a outras instituições.

Referências

Arroyo, D. (2013). Teaching parallel and distributed computing. *IEEE Distributed Systems Online*.

AutomatedLab Project (2024). Automatedlab: Automated lab environment setup. https://automatedlab.org/. Accessed: 2024-12-04.

Conte, D. J., de Souza, P. S. L., Martins, G., and Bruschi, S. M. (2020). Teaching parallel programming for beginners in computer science. In *Proceedings of the 2020 IEEE Frontiers in Education Conference (FIE)*, pages 1–9. IEEE.

Cristian, F. (1989). Probabilistic clock synchronization. *Distributed Computing*, 3(3):146–158.

Crouch, C. H. and Mazur, E. (2001). Peer instruction: Ten years of experience and results. *American Journal of Physics*, 69(9):970–977.

Davis, F. D. (1989). Perceived usefulness, perceived ease of use, and user acceptance of information technology. *MIS Quarterly*, 13(3):319–340.

Ellis, M. et al. (2019). Teaching rigorous distributed systems with efficient model checking. In *Proceedings of the ACM European Conference on Computer Systems*.

Gil, A. C. (2002). *Como Elaborar Projetos de Pesquisa*. Atlas, São Paulo, 4 edition.

Hart, S. G. (2006). NASA-Task Load Index (NASA-TLX): 20 years later. In *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, volume 50, pages 904–908. SAGE Publications.

Hart, S. G. and Staveland, L. E. (1988). Development of NASA-TLX (Task Load Index): Results of empirical and theoretical research. In Hancock, P. A. and Meshkati, N., editors, *Human Mental Workload*, pages 139–183. Elsevier, Amsterdam.

Kokotsaki, D., Menzies, V., and Wiggins, A. (2016). Project-based learning: A review of the literature. *Improving Schools*, 19(3):267–277.

Laadan, O., Nieh, J., and Viennot, N. (2010). Teaching operating systems using virtual appliances and distributed version control. In *Proceedings of the 41st ACM Technical Symposium on Computer Science Education*, SIGCSE '10, pages 480–484, New York, USA. ACM.

Lage, M. J., Platt, G. J., and Treglia, M. (2000). Inverting the classroom: A gateway to creating an inclusive learning environment. *Journal of Economic Education*, 31(1):30–43.

Lazonder, A. W. and Harmsen, R. (2016). Meta-analysis of inquiry-based learning: Effects of guidance. *Review of Educational Research*, 86(3):681–718.

Likert, R. (1932). A technique for the measurement of attitudes. *Archives of Psychology*, 22(140):1–55.

Malan, D. J. (2022). Standardizing students' programming environments with docker containers: Using visual studio code in the cloud with github codespaces. In *Proceedings of the 27th ACM Conference on Innovation and Technology in Computer Science Education*, volume 1 of *ITiCSE '22*, pages 191–197, New York, USA. ACM.

Mann, H. B. and Whitney, D. R. (1947). On a test of whether one of two random variables is stochastically larger than the other. *The Annals of Mathematical Statistics*, 18(1):50–60.

Manoharan, S. and Ye, X. (2023). Teaching introductory parallel programming using two-player online games. In *Proceedings of the 46th MIPRO Conference on Computers and Informatics*, pages 1–6. IEEE.

Merkel, D. (2014). Docker: Lightweight linux containers for consistent development and deployment. *Linux Journal*, 2014(239):2.

O'Connor, T., Schmith, A., Stricklan, C., Carvalho, M., and Sudhakaran, S. (2024). PWN lessons made easy with docker: Toward an undergraduate vulnerability research cybersecurity class. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education*, volume 1 of *SIGCSE '24*, pages 998–1004, New York, USA. ACM.

Parmelee, D., Michaelsen, L. K., Cook, S., and Hudes, P. D. (2012). Team-based learning: a practical guide: Amee guide no. 65. *Medical teacher*, 34(5):e275–e287.

Pedaste, M., Mäeots, M., Siiman, L. A., de Jong, T., van Riesen, S. A., Kamp, E. T., Manoli, C. C., Zacharia, Z. C., and Tsourlidaki, E. (2015). Phases of inquiry-based learning: Definitions and the inquiry cycle. *Educational Research Review*, 14:47–61.

Sato, D. T., Corbucci, H., and Bravo, M. V. (2008). Coding dojo: An environment for learning and sharing agile practices. In *Agile 2008 Conference*, pages 459–464. IEEE.

Smith, J. E. and Nair, R. (2005). The architecture of virtual machines. *Computer*, 38(5):32–38.

Tavakol, M. and Dennick, R. (2011). Making sense of Cronbach's alpha. *International Journal of Medical Education*, 2:53–55.

The Design-Based Research Collective (2003). Design-based research: An emerging paradigm for educational inquiry. *Educational Researcher*, 32(1):5–8.

van de Pol, J., Volman, M., and Beishuizen, J. (2010). Scaffolding in teacher–student interaction: A decade of research. *Educational Psychology Review*, 22(3):271–296.

Wang, H. (2023). Novice-friendly parallel programming exercises. 4th year project report, University of Edinburgh, Edinburgh, UK.

Williams, L., Kessler, R. R., Cunningham, W., and Jeffries, R. (2000). Strengthening the case for pair programming. *IEEE Software*, 17(4):19–25.

Wood, D., Bruner, J. S., and Ross, G. (1976). The role of tutoring in problem solving. *Journal of Child Psychology and Psychiatry*, 17(2):89–100.

Wood, D. F. (2003). Problem based learning. *BMJ*, 326(7384):328–330.