UNIVERSIDADE DE SOROCABA – UNISO
PRÓ-REITORIA DE PÓS-GRADUAÇÃO, PESQUISA, EXTENSÃO E INOVAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM
PROCESSOS TECNOLÓGICOS E AMBIENTAIS

RICHARDSON KENNEDY LUZ

Integração de Metodologias Tradicionais e Ágeis no Ensino de Engenharia de Software: Uma Proposta de Modelo de Plano de Ensino para Cursos de Análise e Desenvolvimento de Sistemas a partir de uma Análise Crítica em Instituições de Ensino Superior Privadas de Sorocaba/SP

Sorocaba – SP
2025

RICHARDSON KENNEDY LUZ

Integração de Metodologias Tradicionais e Ágeis no Ensino de Engenharia de Software: Uma Proposta de Modelo de Plano de Ensino para Cursos de Análise e Desenvolvimento de Sistemas a partir de uma Análise Crítica em Instituições de Ensino Superior Privadas de Sorocaba/SP

.

Tese apresentada à Banca Examinadora do Programa de Pós-Graduação em Processos Tecnológicos e Ambientais da Universidade de Sorocaba, como exigência parcial para obtenção do título de Doutor em Processos Tecnológicos e Ambientais.

.

Orientador: Prof. Dr. Daniel Bertoli Gonçalves

Sorocaba – SP

2025

Ficha Catalográfica

Luz, Richardson Kennedy
L994i Integração de metodologias tradicionais e ágeis no ensino de engenharia de software: uma proposta de modelo de plano de ensino para cursos de Análise e Desenvolvimento de Sistemas a partir de uma análise crítica em instituições de ensino superior privadas de Sorocaba/SP / Richardson Kennedy Luz. -- 2025.
164 f. : il.

Orientador: Prof. Dr. Daniel Bertoli Gonçalves.
Tese (Doutorado em Processos Tecnológicos e Ambientais) - Universidade de Sorocaba, Sorocaba, SP, 2025.

1. Engenharia de software. 2. Universidades e faculdades particulares – Sorocaba (SP). 3. Ensino superior – Inovações tecnológicas. 4. Metodologia. I. Gonçalves, Daniel Bertoli, orient. II. Universidade de Sorocaba. III. Título.

Elaborada por Maria Carla Pascotte Freitas Gonçalves – CRB-8 6721

RICHARDSON KENNEDY LUZ

Integração de Metodologias Tradicionais e Ágeis no Ensino de Engenharia de Software: Uma Proposta de Modelo de Plano de Ensino para Cursos de Análise e Desenvolvimento de Sistemas a partir de uma Análise Crítica em Instituições de Ensino Superior Privadas de Sorocaba/SP

Tese apresentada à Banca Examinadora do Programa de Pós-Graduação em Processos Tecnológicos e Ambientais da Universidade de Sorocaba, como exigência parcial para obtenção do título de Doutor(a) em Processos Tecnológicos e Ambientais.

Aprovado em: 09/09/2025

BANCA EXAMINADORA:

Prof. Dr. Daniel Bertoli Gonçalves (Orientador)
Universidade de Sorocaba - UNISO

Prof. Dr. Ricardo José Orsi de Sanctis (Titular)
Faculdade de Tecnologia de Sorocaba - FATEC

Prof. Dr. Bruno Aguilar da Cunha (Titular)
Universidade de Sorocaba - UNISO

Prof. Dr. Marcello Bellodi (Titular)
Universidade Paulista - UNIP

Prof. Dr. José Luiz Antunes de Almeida (Titular)
Universidade de Sorocaba - UNISO

AGRADECIMENTO

Agradeço ao Prof. Dr. Daniel Bertoli Gonçalves, que aceitou o desafio e com sua orientação possibilitou a realização deste trabalho.

Aos meus pais Robinson Luz, (em memória), e Roseli Martins Luz, (em memória), que sempre me apoiaram, de onde estiverem continuam a olhar por mim.

As minhas filhas Pietra e Luna, pois sem o apoio, compreensão, ajuda e paciência delas, este trabalho não seria concluído.

Um obrigado especial aos professores, Profa. Dra. Ednilse Leme, Prof. Dr. Manuel Meirelles, pelas importantes contribuições para este trabalho.

Agradecimento a Profa. Me. Ivani Vecina Abib, responsável pela revisão de língua portuguesa deste trabalho.

Um obrigado aos professores, Prof. Dr. Ricardo José Orsi de Sanctis, Prof. Dr. Bruno Aguilar da Cunha, Prof. Dr. Marcello Bellodi e Dr. José Luiz Antunes de Almeida, pelas importantes contribuições fazendo parte da banca de professores avaliadores.

Aos colegas, diretor, coordenadores, professores e estudantes Universidade Paulista – UNIP e da Universidade de Sorocaba – UNISO, pelos conselhos e apoios.

Agradeço às pessoas que, direta ou indiretamente, ajudaram na elaboração deste trabalho.

"Engenharia de Software é a aplicação de uma abordagem sistemática, disciplinada e quantificável ao desenvolvimento, operação e manutenção de software." PRESMMAM

RESUMO

Esta pesquisa investiga como a Engenharia de Software é abordada em cursos de graduação de 05 (cinco) instituições particulares de ensino superior em Sorocaba/SP. Tem como objetivo propor estratégias pedagógicas que melhor alinhem o ensino às demandas do mercado de trabalho, em constante evolução. O estudo foca na comparação entre os modelos de processos tradicionais e ágeis, examinando como ambas são integradas ao currículo e qual a relevância de cada uma para a formação dos futuros engenheiros de software. A pesquisa se caracteriza como um estudo descritivo e exploratório, de natureza qualitativa, com enfoque documental e bibliográfico com análise comparativa. Os resultados indicam que, embora as metodologias ágeis estejam ganhando maior destaque devido à sua flexibilidade e alinhamento com as necessidades atuais do mercado, os métodos tradicionais ainda desempenham um papel crucial na compreensão completa do ciclo de vida do software. Além disso, a pesquisa explora o impacto das inovações tecnológicas e pedagógicas no ensino, destacando a necessidade de uma abordagem que equilibre tradição e inovação. Conclui-se que um currículo híbrido, que combine práticas ágeis com fundamentos tradicionais, é essencial para a formação de profissionais bem-preparados para enfrentar os desafios do setor de Tecnologia da Informação. As propostas de melhoria incluem a adaptação contínua do currículo e a incorporação de tecnologias educacionais avançadas.

Palavras-chave: Metodologias Ágeis, Engenharia de Software, Educação Superior.

ABSTRACT

This research investigates how Software Engineering is addressed in undergraduate programs at five private higher education institutions in Sorocaba/SP. Its objective is to propose pedagogical strategies that better align teaching with the constantly evolving demands of the labor market. The study focuses on the comparison between traditional and agile process models, examining how both are integrated into the curriculum and the relevance of each to the future software engineers' education. The research is characterized as a descriptive and exploratory study, of a qualitative nature, with a documentary and bibliographic approach and comparative analysis.The results indicate that, although agile methodologies are gaining greater prominence due to their flexibility and alignment with current market needs, traditional methods still play a crucial role in fully understanding the software life cycle. Furthermore, the research explores the impact of technological and pedagogical innovations on teaching, highlighting the need for an approach that balances tradition and innovation. It is concluded that a hybrid curriculum, combining agile practices with traditional foundations, is essential for preparing professionals to face the challenges of the Information Technology sector. The improvement proposals include the continuous adaptation of the curriculum and the incorporation of advanced educational technologies.

key-words: Agile Methodologies, Software Engineering, Higher Education

LISTA DE SIGLAS E ABREVIATURAS

ACM - Associação para Maquinaria da Computação

AVA - Ambientes Virtuais de Aprendizagem

C4 Model - Context, Containers, Components, Code

CI/CD - Conhecimento em Integração e Entrega Contínua

CMMI - Capability Maturity Model Integration

DCN - Diretrizes Curriculares Nacionais

DDD - Domain-Driven Design

DevOps - Development + Operations

DSDM - Dynamic Systems Development Method

FDD - Feature-Driven Development

GIT - Sistema de Controle de Versão

IDE - Integrated Development Environments

ITIL - Information Technology Infrastructure Library

MEC - Ministério da Educação

MPS.Br - Melhoria de Processo do Software Brasileiro

MVC - Model-View-Controller

MVP - Minimum Viable Product

OOPSLA - Object-Oriented Programming, Systems, Languages & Applications

PE_IES – Plano de Ensino _ Instituição de Ensino Superior

PBL - Problem-Based Learning

PIM - Projeto Integrado Multidisciplinar

PPC - Plano Pedagógico do Curso

PSP - Personal Software Process

RUP - Rational Unified Process

TDD - Test Driven Development

TESTES BDD – Test Behavior-Driven Development

TI - Tecnologia da Informação

UML - Unified Modeling Language

XP - Extreme Programming

LISTA DE ILUSTRAÇÕES

FIGURA 1: EVOLUÇÃO DO SOFTWARE 24

FIGURA 2: MODELO DE PROCESSO - CASCATA 29

FIGURA 3: MODELO DE PROTOTIPAÇÃO 32

FIGURA 4: MODELO ESPIRAL DE BOEHM 38

FIGURA 5: MODELO DO PROCESSO SCRUM 47

FIGURA 6: MODELO ÁGIL XP 50

QUADRO 1: PRÁTICAS DAS ENGENHARIAS 55

QUADRO 2: TAXAS DE INSUCESSO 55

QUADRO 3: PRÁTICAS USADAS POR PREVALÊNCIA 56

QUADRO 4: NÍVEL PREDOMINANTE 59

QUADRO 5: ALINHAMENTO ENTRE FORMAÇÃO ACADÊMICA E MERCADO 68

QUADRO 6 A ANÁLISE COMPARATIVA 92

QUADRO 7: INCORPORAÇÕES TECNOLÓGICAS 110

QUADRO 8: INOVAÇÕES PEDAGÓGICAS 112

QUADRO 9: COMPROVAÇÃO DE PRÁTICAS PEDAGÓGICAS DE APRENDIZAGEM ATIVA 115

QUADRO 10: PROPOSTA DE PLANO DE ENSINO DE ENGENHARIA DE SOFTWARE 130

QUADRO 11: COMPARATIVO MEC X PROPOSTA DO PLANO DE ENSINO 133

GRÁFICO 1: INTEGRAÇÃO DOS MODELOS POR PE_IES 63

GRÁFICO 2: COMPARATIVOS ENTRE AS IES E SEUS CONTEÚDOS PROGRAMÁTICOS 88

GRÁFICO 3: COMPARATIVO DE CONTEÚDO TRADICIONAL X ÁGIL 89

GRÁFICO 4: COMPARATIVO CURRICULAR 91

Gráfico 5: Modelos Tradicionais 93
Gráfico 6: Modelos Ágeis 94
Gráfico 7: Ferramentas Modernas 95
Gráfico 8: Modelos Tradicional e Ágil 96
Gráfico 9: Atualização Curricular 96
Gráfico 10: Distribuição de Conteúdo Tradicionais e Ágeis 98
Gráfico 11: Tradicional x Ágil 99
Gráfico 12: Distribuição Curricular: Modelos Tradicionais vs Metodologias Ágeis 101
Gráfico 13: Abordagem de Desenvolvimento de Software nos Documentos Curriculares: 102
Gráfico 14: Disciplinas comuns vs exclusivas 104
Gráfico 15: Comparativo de Aulas Teóricas vs Práticas por Instituição 106
Gráfico 16: Métodos Ágeis 108
Gráfico 17: Incorporação de Tecnologias no Ensino de Engenharia de Software 109
Gráfico 18: Revisão de Planos de Ensino 119
Gráfico 19: Projetos Práticos Colaborativos 121
Gráfico 20: Uso de Ambientes Virtuais 123
Gráfico 21: Parcerias com Empresas 125
Gráfico 22: Habilidade mais negligenciadas pelos egressos 127

SUMÁRIO

1 INTRODUÇÃO 14
2 OBJETIVOS 18
2.1 Objetivo Geral **18**
2.2 Objetivos Específicos **18**
2.3 Hipóteses a serem testada **19**
3 METODOLOGIA DA PESQUISA 20
3.1 Etapas da Pesquisa **20**
4 REFERENCIAL TEÓRICO 23
4.1 Fundamentos da Engenharia de Software 23
4.2 Modelos de Desenvolvimento de Software 26
4.2.1 Modelos Tradicionais de Desenvolvimento de Software 27
4.2.2 Modelo Cascata 27
4.2.3 Modelo de Prototipação 31
4.2.4 Modelo Espiral De Boehm 37
4.3 O Impacto no Ensino de Engenharia de Software 40
4.4 Metodologias Ágeis 42
4.4.1 Modelo Ágil *SCRUM* 45
4.4.2 Modelo Ágil - *Extreme Programing (XP)* 48
4.5 Causas dos problemas dos Softwares só utilizando o Modelo Ágil 53
4.5.1 Quadros de Pesquisa e Dados 54
5 PROPOSTAS DE MELHORIA PARA O ENSINO DE ENGENHARIA DE SOFTWARE 61
5.1 Integração de Metodologias Tradicionais e Ágeis no Ensino 61
5.1.1 Evolução da Engenharia de Software: a proposta de um modelo híbrido 61
5.1.2 Modelo Híbrido 62
5.1.3 O porquê do modelo híbrido 67
5.2 Alinhamento entre Ensino e Mercado de Trabalho 68
5.2.1 Competências Técnicas 69
5.2.2 Competências Interpessoais 70
5.2.3 Competências Analíticas 71

5.2.4 Competências de Aprendizado Contínuo 72
5.3 Desafios na Formação Acadêmica 73
5.4 Propostas de Melhoria para o Plano de Ensino da disciplina de Engenharia de Software 74
6 RESULTADOS E DISCUSSÃO 78
6.1 Análise Qualitativa do Currículo da Disciplina Engenharia de Software 78
6.1.1 PE_IES A 78
6.1.2 PE_IES B 80
6.1.3 PE_IES C 82
6.1.4 PE_IES D 84
6.1.5 PE_IES E 85
6.2 Análise Documental Análise Qualitativa do Currículo da Disciplina Engenharia de Software 90
6.2.1 Análise Documental Comparada 91
6.2.2 Desempenho e Relevância dos Modelos de Processos Tradicionais e Ágeis da Engenharia de Software 98
6.2.3 Disciplinas Comuns vs. Disciplinas Exclusivas 103
6.3 Comparativo de Aulas Teóricas vs Práticas por Plano de Ensino 105
6.3.1 Análise do Ensino de Metodologias Ágeis 107
6.3.2 Impacto das Inovações Tecnológicas no Ensino 109
6.3.3 Inovações Pedagógicas e seus Resultados 112
6.4 Proposição e Validação Teórica de Estratégias para o Plano de Ensino de Engenharia de Software 116
6.5 Proposição de Estratégias 118
6.5.1 Propostas de Melhoria 118
6.5.2 Revisão de Planos de Ensino e Atualização Curricular 118
6.5.3 Inclusão de Módulos Práticos Colaborativos 120
6.5.4 Ampliação do Uso de Ambientes Virtuais e Ferramentas Educacionais 122
6.5.5 Criação de Parcerias com Empresas e Startups Locais 123
6.6 Alinhamento com as Demandas do Mercado de Trabalho 126
6.7 Proposta do Plano de Ensino Futura 129
6.7.1 Plano de Ensino - Engenharia de Software Híbrida 130
6.7.2 Estrutura MEC 133
6.7.3 Sugestão de Acompanhamento e Avaliação Contínua 134

7 CONSIDERAÇÕES FINAIS 137
REFERÊNCIAS 140
ANEXO A – QUADRO DESCRITIVO DAS SIGLAS 145
ANEXO B – PLANOS DE ENSINO - PE_IES A 147
ANEXO C – PLANOS DE ENSINO - PE_IES B 149
ANEXO D – PLANOS DE ENSINO - PE_IES C 153
ANEXO E – PLANOS DE ENSINO - PE_IES D 161
ANEXO F – PLANOS DE ENSINO - PE_IES E 162

1 INTRODUÇÃO

O mundo moderno é impensável sem o software, que é a base da evolução tecnológica e um elemento essencial em todas as revoluções industriais, especialmente na atual Quarta Revolução Industrial. A Engenharia de Software se tornou uma disciplina vasta e complexa, impossível de ser totalmente abordada em um único estudo. Por isso, este texto se concentrará em destacar as principais características de cada modelo de desenvolvimento, reconhecendo que, embora os processos ágeis sejam amplamente adotados atualmente, os métodos tradicionais e clássicos continuam a ser fundamentais para o desenvolvimento de software, desde a especificação inicial até a evolução contínua dos sistemas.

Segundo Sommerville (2011), a Engenharia de Software é definida como um conjunto de práticas sistemáticas e bem estruturadas para o desenvolvimento, operação e manutenção de software. Ele enfatiza a importância dessa engenharia devido à complexidade dos sistemas modernos e à necessidade de garantir a qualidade, confiabilidade e eficiência dos produtos de software.

De maneira geral, a prática da Engenharia de Software envolve uma coleção de conceitos, princípios, métodos e ferramentas que são utilizados diariamente pelos engenheiros de software. Essas práticas preenchem os modelos de processo de software com as abordagens técnicas e gerenciais necessárias para executar o trabalho, desde a definição do escopo do projeto até a implantação e manutenção do software, simplificando o processo, por mais complexo que ele seja.

A Engenharia de Software propõe métodos, processos e ferramentas que visam à construção de softwares de alta qualidade, dentro dos prazos estabelecidos e atendendo ao orçamento do cliente. Entre as metodologias estudadas estão os modelos de processos ágeis como *Extreme Programming* (Beck, 2000), *SCRUM* (Schwaber; Sutherland, 1997), e a integração com modelos de processos tradicionais, como o modelo em Cascata (Royce, 1970), Prototipação (Yourdon, 1990) e o modelo Espiral (Boehm, 1988), permitindo a escolha do melhor modelo de processo de software para garantir a entrega de software que atenda plenamente às necessidades do cliente.

Sommerville (2011) afirma que “um processo de software é um conjunto de atividades relacionadas que levam à produção de um produto de software.” Essas atividades podem incluir o desenvolvimento de sistemas a partir do zero em uma

linguagem padrão de programação. Em outras palavras, são etapas e práticas previamente definidas para desenvolver softwares e os produtos associados, como documentos de projeto, código-fonte, casos de uso e manuais de sistema, com o objetivo de garantir a qualidade e a produtividade dos desenvolvedores.

Na Engenharia de Software, diferentes modelos de desenvolvimento são empregados, comumente classificados em duas categorias principais: modelos tradicionais e modelos ágeis. Enquanto os modelos tradicionais adotam uma abordagem sequencial e estruturada, em que cada fase do desenvolvimento deve ser concluída antes de se iniciar a próxima, os modelos ágeis enfatizam a flexibilidade, a colaboração contínua com o cliente e a entrega incremental de software.

Nos cursos de nível superior da área, o ensino da Engenharia de Software geralmente abrange tanto os modelos tradicionais quanto os ágeis, refletindo a necessidade de preparar os estudantes para um mercado de trabalho dinâmico. Esse mercado valoriza tanto o conhecimento dos modelos tradicionais quanto a proficiência em métodos ágeis. No entanto, a relevância do ensino de metodologias ágeis tem crescido, acompanhando a tendência do setor de Tecnologia da Informação (TI), em que a agilidade, a flexibilidade e a capacidade de resposta rápida às mudanças são cruciais. Consequentemente, cursos de extensão e especialização também têm concentrado esforços em metodologias ágeis, com algumas instituições oferecendo programas específicos para *SCRUM Masters*, *Product Owners* e outras certificações ágeis.

Nesse contexto, este trabalho de pesquisa tem como objetivo propor um modelo de plano de ensino a partir de uma investigação sobre como a disciplina de Engenharia de Software é abordada no curso de graduação de Análise e Desenvolvimento de Sistemas. Para isso, foram analisados os planos de ensino de cinco (05) instituições particulares de ensino superior na cidade de Sorocaba/SP, enquanto recorte territorial representativo da região, já que três (03) delas são redes nacionais de universidades.

Nos últimos anos, diversos autores tem alertado sobre tal problema, como é o caso de Oguz e Oguz (2019) e Haystack Analytics (2023), que afirmam que 65% dos egressos de cursos de Engenharia de Software não dominam metodologias ágeis exigidas pelo mercado, apesar de 89% das empresas priorizarem essa competência. Essa discrepância atesta a distância existente entre a realidade dos cursos e as demandas do mercado, algo que o modelo proposto busca diminuir.

Para isso, inicialmente foi adotada uma revisão bibliográfica aprofundada, com o objetivo de contextualizar a integração de metodologias tradicionais e ágeis no ensino da disciplina de Engenharia de Software. Uma etapa que permitiu identificar lacunas na literatura e fundamentar a pesquisa no cenário acadêmico.

Em seguida, a coleta de dados se deu por meio da análise de documentos institucionais, como planos de ensino, ementas de disciplinas e projetos pedagógicos relacionado à disciplina dos cursos em questão.

Com os dados coletados, foi realizada uma análise comparativa entre os conteúdos abordados e as melhores práticas descritas na literatura, bem como as demandas do mercado de trabalho. Essa análise permitiu avaliar se o modelo atual está alinhado com as expectativas do setor de TI, conforme apontado na literatura.

Com base nos resultados obtidos, foram propostas estratégias concretas para melhorar a abordagem da Engenharia de Software nas instituições estudadas, de modo a incluir a integração de metodologias ágeis em disciplinas tradicionais, o desenvolvimento de novos módulos focados em práticas ágeis, e a inclusão de projetos práticos colaborativos que reflitam situações reais do mercado. Ademais, diante da subutilização de metodologias ágeis nos currículos, esta pesquisa propõe um modelo híbrido, que integra o Modelo Tradicional e o Modelo Ágil, preenchendo dois *gaps* críticos: a falta de flexibilidade para mudanças de requisitos (problema dos modelos tradicionais) e a carência de documentação robusta (problema dos modelos ágeis).

A proposta também inclui um plano de implementação das estratégias nas instituições, considerando a revisão curricular necessária e a criação de parcerias entre as Universidades com empresas, e assim, proporcionar experiências práticas.

A presente pesquisa, focada na investigação de como a disciplina de Engenharia de Software é abordada em cursos de graduação e na proposição de estratégias pedagógicas, também, para um melhor alinhamento com as demandas do mercado de trabalho, está fortemente alinhada com as áreas de concentração "inovação, tecnologia e meio ambiente" e com a linha de pesquisa "desenvolvimento e controle de processos tecnológicos" do programa de pós-graduação ao qual pertence. Esse alinhamento se evidencia, primeiramente, pela inovação na disciplina Engenharia de Software.

A adoção e integração de metodologias ágeis no currículo representam uma mudança significativa, pois essas metodologias, ao contrário dos modelos

tradicionais, promovem uma aprendizagem mais flexível e dinâmica. Elas permitem que os estudantes interajam diretamente com processos de desenvolvimento de software que refletem as práticas reais do mercado, adaptando o ensino às novas demandas tecnológicas e de inovação que emergem no setor de TI.

Diante de um mercado de TI em constante evolução, que exige profissionais com conhecimento técnico e capacidade de adaptação às mudanças, a proposta desenvolvida nesse trabalho busca proporcionar aos graduandos uma melhor preparação para tais exigências. Assim, a formação proposta valoriza não apenas a eficiência tecnológica, mas também a inovação contínua e a capacidade de resposta às novas tendências do setor, reforçando o vínculo entre o projeto de pesquisa e as áreas de concentração e linha de pesquisa do grupo.

Nas sessões seguintes estão apresentados os objetivos deste trabalho, sua metodologia, referencial teórico, resultados e discussões, incluindo o modelo proposto de plano de ensino e suas estratégias de implementação.

2 OBJETIVOS

2.1 Objetivo Geral

Propor um modelo de plano de ensino mais alinhado com as necessidades do mercado de trabalho, a partir de uma análise crítica sobre como a disciplina de Engenharia de Software é abordada nos cursos de graduação de Análise e Desenvolvimento de Sistemas das cinco (05) Instituto Superior de Ensino (IES) particulares da cidade de Sorocaba - SP.

2.2 Objetivos Específicos

1. Realizar uma revisão bibliográfica sobre a integração de metodologias ágeis e tradicionais no ensino de Engenharia de Software, identificando lacunas e melhores práticas descritas na literatura.

2. Analisar documentos institucionais, como planos de ensino, ementas de disciplinas e projetos pedagógicos dos cursos investigados, para identificar como os modelos tradicionais e ágeis de desenvolvimento de software estão sendo abordados.

3. Propor estratégias concretas para melhorar o plano de ensino da disciplina de Engenharia de Software nas instituições estudadas, visando um maior alinhamento com as expectativas do setor de TI.

4. Validar teoricamente as propostas de melhoria utilizando a literatura e os documentos analisados como base, recomendando um plano de implementação nas instituições de ensino.

5. Sugerir um processo de acompanhamento e avaliação contínua das mudanças implementadas, para garantir a eficácia das estratégias em longo prazo.

2.3 Hipótese a ser testada

A abordagem atual presente dos planos de ensino da disciplina de Engenharia de Software nos cursos de graduação apresenta lacunas na formação dos estudantes, em relação à integração de modelos de processos ágeis.

A hipótese será avaliada por meio de:

- Análise documental dos planos de ensino das instituições (identificadas como IES A a E para preservar anonimato);
- Revisão bibliográfica sobre eficácia de metodologias ágeis na educação em Engenharia de Software.

3 METODOLOGIA DA PESQUISA

A presente pesquisa se caracteriza como um estudo descritivo e exploratório, de natureza qualitativa, com enfoque documental e bibliográfico com análise comparativa. A escolha dessa abordagem se justifica pela necessidade de compreender como a Engenharia de Software é abordada em cursos de graduação, analisando documentos institucionais e literatura especializada para identificar práticas e propor melhorias no alinhamento com as demandas do mercado de trabalho.

Foram analisados os planos de ensino de cinco Instituições de Ensino Superior privadas, localizadas no município de Sorocaba/SP, definido como recorte territorial representativo da região de estudo, entre os anos de 2023 e 2025. A seleção dessas IES fundamentou-se em critérios de pertinência geográfica ao programa de Pós-Graduação, viabilidade de acesso e disponibilidade dos documentos institucionais, bem como no potencial de geração de impacto acadêmico e social em escala regional e nacional. Ressalta-se que, dentre as cinco instituições investigadas, três integram redes universitárias com presença em outros municípios e estados, o que amplia a relevância dos resultados obtidos, possibilitando inferências e comparações em contextos mais amplos.

Os Planos de Ensino das Instituição de Ensino Superior estudadas foram denominadas pelo código PE_IES A, B, C, D e E, mantendo assim o anonimato delas. Os planos de Ensino foram repassados pelos coordenadores dos cursos de Análise e Desenvolvimento de Sistemas (ADS) dessas IES, apesar do livre acesso, para qualquer pessoa, ao Plano Pedagógico do Curso (PPC) no site de cada Instituição.

3.1 Etapas da Pesquisa

A abordagem qualitativa permite uma análise aprofundada dos currículos e das práticas educacionais. Entretanto, apresenta limitações inerentes ao método, como a influência da subjetividade do pesquisador na interpretação dos dados documentais e a restrição na compreensão das lacunas pedagógicas quando não se utilizam fontes diretas, como entrevistas com professores ou estudantes (Lakatos; Marconi, 2021; Minayo, 2014; Gil, 2008).

I. **Revisão Bibliográfica.** A primeira etapa consistiu em realizar uma revisão bibliográfica abrangente sobre a integração de metodologias ágeis e tradicionais no plano de ensino da disciplina de Engenharia de Software. Essa revisão busca contextualizar o estudo dentro do panorama acadêmico e identificar lacunas na literatura. Foram consultadas bases de dados acadêmicas, livros e artigos especializados, buscando entender como essas metodologias têm sido abordadas e quais são as tendências e melhores práticas atuais no ensino da Engenharia de Software por meio da triangulação das fontes e revisão por pares.

II. **Análise Documental.** Na segunda etapa, a pesquisa se concentrou na análise documental dos cursos de graduação das cinco instituições particulares de ensino superior de Sorocaba/SP. Foram coletados e examinados documentos institucionais como planos de ensino, ementas de disciplinas e projetos pedagógicos vigentes entre o primeiro semestre de 2023 e primeiro semestre de 2025. A escolha da análise documental se justifica pela sua capacidade de oferecer uma visão detalhada e sistemática das abordagens de ensino adotadas pelas instituições.

III. **Análise Comparativa.** A terceira etapa envolveu uma análise comparativa entre as práticas educacionais identificadas na análise documental e as melhores práticas descritas na literatura, bem como as exigências do mercado de trabalho. Essa comparação permitiu avaliar o grau de alinhamento entre o ensino atual de Engenharia de Software e as necessidades do setor de TI, identificando possíveis discrepâncias e áreas que necessitam de melhorias. A análise comparativa é essencial para fundamentar as propostas de aprimoramento da disciplina de Engenharia de Software.

IV. **Proposição de Estratégias.** Com base nas informações coletadas e analisadas, foram propostas estratégias concretas para melhorar a abordagem do plano de ensino da disciplina de Engenharia de Software nas instituições estudadas. Essas estratégias incluem a revisão de ementas, a inclusão de novos módulos focados em práticas ágeis e a implementação de projetos práticos colaborativos que reflitam situações reais do mercado.

V. **Validação Teórica das Propostas.** As estratégias propostas foram submetidas a uma validação teórica, utilizando como base a literatura revisada e os documentos analisados. Essa validação buscou garantir que as sugestões de melhoria estejam solidamente fundamentadas e sejam viáveis dentro do contexto educacional das instituições.

VI. **Sugestão de Acompanhamento e Avaliação Contínua.** Foi sugerido um processo de acompanhamento e avaliação contínua das mudanças implementadas. Esse processo é necessário para monitorar a eficácia das estratégias adotadas ao longo do tempo, garantindo que elas continuem a atender às necessidades do mercado de trabalho em constante evolução. A avaliação contínua permitirá ajustes e melhorias contínuas, assegurando que a formação oferecida pelos cursos de graduação mantenha-se atualizada e relevante.

4 REFERENCIAL TEÓRICO

Este capítulo tem como finalidade apresentar o referencial teórico que sustenta este trabalho de pesquisa. A Engenharia de Software é um campo dinâmico e em constante evolução, que requer uma base teórica sólida para a compreensão de suas práticas, metodologias e ferramentas. Assim, este referencial visa a delinear as principais abordagens e conceitos que guiam a investigação em curso.

Serão discutidas as metodologias tradicionais e ágeis e de desenvolvimento de software, com ênfase nas suas particularidades, vantagens e desvantagens.

A interseção entre a teoria e a prática será o foco central, uma vez que a aplicação eficaz dos conceitos teóricos é essencial para a melhoria dos processos de desenvolvimento.

Ao final deste capítulo, espera-se que o referencial teórico não apenas contextualize a pesquisa, mas também possibilite a identificação de lacunas e oportunidades para futuras investigações, promovendo um diálogo crítico com a literatura existente.

4.1 Fundamentos da Engenharia de Software

A Engenharia de Software existe há cinco décadas, e hoje ela já é reconhecida como uma disciplina legítima no mundo do desenvolvimento de software. A disciplina está entre as mais importantes no século XXI, pois ela tem o objetivo de transformar e controlar o bem mais importante dessa era: a informação (Paula filho, 2012).

A crise do software teve seu início na década de 1970, quando o rápido crescimento e a complexidade dos sistemas começaram a evidenciar sérias dificuldades nos desenvolvimentos.

De acordo com Boehm (1981), o aumento da demanda por software de qualidade e a complexidade dos projetos revelaram a necessidade urgente de estabelecer processos e técnicas eficazes para o planejamento e desenvolvimento de sistemas. Foi nesse contexto que o termo “Crise do Software” foi proposto por Edsger Dijkstra (1972) que, em sua palestra de 1972 na Associação para Maquinaria da Computação, fundada em 1947, na cidade de Nova York (ACM), destacou as falhas e a fragilidade da indústria de software da época.

A crise do software foi um período marcado por dificuldades no desenvolvimento de software, devido a vários fatores, como:

- Aumento da demanda por software,
- Complexidade dos problemas a serem resolvidos,
- Inexistência de técnicas estabelecidas para o desenvolvimento de sistemas,
- Imaturidade da Engenharia de Software como profissão.

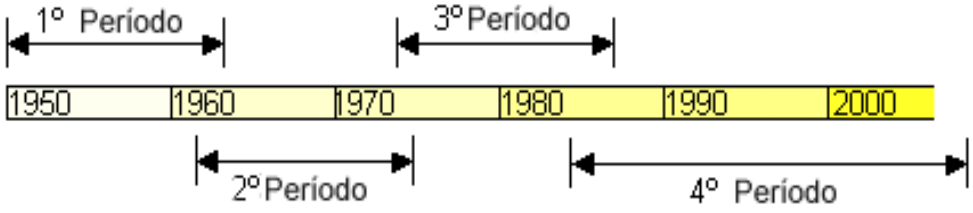
A crise do software se manifestava de várias formas, como:

- Projetos que ultrapassavam o orçamento,
- Projetos que ultrapassavam o prazo,
- Software de baixa qualidade,
- Software que não satisfazia os requisitos,
- Projetos integráveis,
- Código difícil de manter.

Para Lobo (2008), a disciplina de Engenharia de Software está evoluindo cada vez mais, como se pode ver na Figura 1, a qual detalha:

- 1º período corresponde à orientação e distribuição limitada,
- 2º período envolve as etapas de multiusuário, a entrada de banco de dados, a informação em “tempo real” e a produção do software.
- 3º período entra em sistemas distribuídos, com baixos custos de hardware e
- 4º período, que no presente momento são os poderosos sistemas de mesas, tecnologias orientadas a objeto, redes neurais e a busca de excelentes sistemas especialistas, os quais terão alto poder de decisão na Era da Indústria 4.0.

Figura 1: Evolução do software



Fonte: Lobo, 2008

A imaturidade da sociedade na criação de software resultou em projetos de baixa qualidade, com códigos de difícil manutenção, além de orçamentos superados e prazos não cumpridos. Como destacou Pressman (2014), esse cenário levou à necessidade de uma abordagem mais estruturada e disciplinada para o desenvolvimento de software, o que culminou no surgimento da Engenharia de Software como uma profissão e área de conhecimento especializada.

> Desenvolver um software sem nenhuma sistematização é um verdadeiro caos, mas utilizar processos pesados também se torna uma carga insustentável para as empresas, em custo e tempo. O meio termo está na metodologia ágil de desenvolvimento de software, pois ela define regras e rotinas, que não “pesam” no desenvolvimento do software, mas torna o ambiente mais controlado e rápido. [...] A documentação da metodologia ágil é leve, permitindo assim dinamismo e flexibilidade às mudanças no projeto. Para que a documentação de um software seja útil, ela deve ser fácil de ser criada e entendida pela equipe de software. (Lobo, 2008, pág. 42).

Segundo Sommerville (2011), Engenharia de Software é uma disciplina de engenharia relacionada com todos os aspectos da produção de software, desde os estágios iniciais de especificação do sistema até sua manutenção, depois que esse entrar em operação. Nessa definição, há duas etapas importantes que são discutidas em sala de aula e que devem ser focadas nas empresas.

I. **Disciplina de Engenharia**. É dito uma disciplina, porque os engenheiros estão sempre aplicando teorias, métodos e ferramentas sempre onde elas forem mais apropriadas e usando-as de modo seletivo; além disso, os engenheiros estão sempre estudando novas maneiras e métodos de solucionar os problemas.

II. **Todos os aspectos da produção de software**. Significa dizer que a Engenharia não está relacionada apenas com os processos técnicos de desenvolvimento de software, mas também com atividades como o gerenciamento e desenvolvimento de ferramentas, métodos e teorias que apoiem a produção de software.

Alguns dos principais objetivos dessa disciplina incluem capacitar os estudantes a produzirem softwares de qualidade, respeitando prazo e custo previstos, além de proporcionar uma visão ampla do processo de desenvolvimento, desde a definição de requisitos até a entrega da aplicação final.

A disciplina também busca demonstrar a importância de compreender os principais modelos de processo de desenvolvimento de software, identificando boas práticas recorrentes em diferentes abordagens. Mesmo com a adoção de modelos mais atuais e alinhados às tendências do mercado, a experiência prática revela que não existe um modelo único capaz de atender a todas as demandas de um projeto. A escolha do processo mais adequado deve considerar os requisitos identificados, as características do projeto e as necessidades específicas do cliente, podendo inclusive envolver a combinação de abordagens tradicionais e ágeis para garantir entregas eficazes e adaptáveis.

Para Lessa (2019), a Engenharia de Software compreende um conjunto de etapas que fazem a ligação de forma organizada de processos. O conjunto de etapas compreende os modelos de Engenharia de Software. Cada paradigma compõe o Ciclo de Vida do software. Um paradigma pode ser escolhido dependendo da natureza do projeto e da aplicação, métodos, ferramentas a serem usados, os controles e os produtos que precisam ser entregues.

Um processo de software pode ser diferente dependendo da organização ou do projeto, ser adaptável às necessidades. Esse processo conta com a ajuda de toda a equipe de desenvolvimento, equipe de testes, gerentes, entre outros. Além também dos próprios solicitantes do software que devem colaborar com sua definição, construção e teste.

4.2 Modelos de Desenvolvimento de Software

Segundo Sommeville (2011), os modelos de processos de software tradicionais são caracterizados por metodologias estruturadas e bem definidas, criadas para guiar o desenvolvimento de software de forma sistemática e previsível. De modo geral, seguem uma abordagem sequencial, na qual as fases como levantamento de requisitos, projeto, implementação e testes são realizadas em ordem específica e com baixa flexibilidade.

Para Pressman (2016), os modelos tradicionais apresentam valor significativo em virtude de sua estrutura clara e rigorosa, que pode auxiliar na gestão de projetos de grande porte e alta complexidade. Contudo, tendem a ser menos adaptáveis diante de mudanças frequentes, exigindo planejamento e controle mais rígidos. Por essa

razão, diversos autores ressaltam a necessidade de adotar modelos híbridos ou ajustar práticas para equilibrar previsibilidade e flexibilidade

Já para Beck et. al (2001), modelos de processos de software ágil são metodologias que enfatizam a flexibilidade, a colaboração e a capacidade de resposta rápida às mudanças. Eles contrastam com os modelos tradicionais, que tendem a ser mais rígidos e sequenciais. Os processos ágeis são projetados para lidar com a complexidade e a incerteza do desenvolvimento de software, permitindo uma adaptação contínua às necessidades dos clientes e às mudanças no mercado.

Schwaber; Beedle (2002), os modelos ágeis são projetados para permitir uma abordagem mais flexível e adaptativa ao desenvolvimento de software. Eles valorizam a colaboração, a entrega contínua de valor e a capacidade de responder rapidamente às mudanças. Cada modelo ágil tem suas próprias características e práticas, e a escolha do modelo mais apropriado pode depender das necessidades específicas do projeto, da equipe e da organização. Em muitos casos, as organizações adotam uma combinação de práticas ágeis para criar um processo que se adapte melhor às suas necessidades.

4.2.1 Modelos Tradicionais de Desenvolvimento de Software

Royce (1970), afirma que os modelos tradicionais de desenvolvimento de software representam as primeiras tentativas sistemáticas de organizar e estruturar o processo de construção de sistemas computacionais. Esses modelos surgiram em um contexto em que a previsibilidade, o planejamento detalhado e a documentação extensiva eram consideradas essenciais para garantir a qualidade do produto. Dentre os mais conhecidos, destacam-se o modelo Cascata *(Waterfall),* o modelo Incremental, o modelo Espiral, modelo de Prototipação e o RUP *(Rational Unified Process).*

A seguir serão apresentadas mais as características e aplicações sobre os modelos tradicionais, suas vantagens e contextos em que são mais adequados.

4.2.2 Modelo Cascata

O primeiro modelo do processo de desenvolvimento de software a ser publicado foi derivado de processos mais gerais da engenharia de sistemas criados

na década de 70 (Royce, 1970). O modelo só ganhou visibilidade cerca de 10 anos mais tarde e pode ser considerado mais rígido e menos administrativo que outros que têm o mesmo objetivo.

Segundo Royce (1970), todo o processo de desenvolvimento ocorre de forma linear, ou seja, as atividades são agrupadas em tarefas e executadas sequencialmente de maneira que uma só poderá ter início quando a sua anterior for finalizada.

Segundo Lawrence (2004), o modelo Cascata é considerado como de alto nível, pois depende da compreensão dos envolvidos e a capacidade de prever situações na fase de desenvolvimento que possam colocar riscos ao projeto com prazos, tecnologias, entre outros. Além disso, o ciclo de vida clássico gera documentações detalhadas em cada fase do sistema a ser construído, tornando assim, o sistema mais transparente e possibilita um melhor controle por parte dos gestores.

Segundo Sommerville (2011), o modelo Cascata possui cinco fases, chamadas também de estágios: definição de requisitos, projeto de sistema e software, implementação e teste unitário, integração e teste de sistema, e operação e manutenção.

I. **Definição de requisitos.** Os serviços, restrições e metas do sistema são estabelecidos por meio de consulta aos usuários.

II. **Projeto de sistema e software.** O processo de projeto de sistemas aloca os requisitos tanto para sistemas de hardware como para sistemas de software, por meio da definição de uma arquitetura geral do sistema.

III. **Implementação e teste unitário.** Durante esse estágio, o projeto do software é desenvolvido como um conjunto de programas ou unidades de programa. O teste unitário envolve a verificação de que cada unidade atenda a sua especificação.

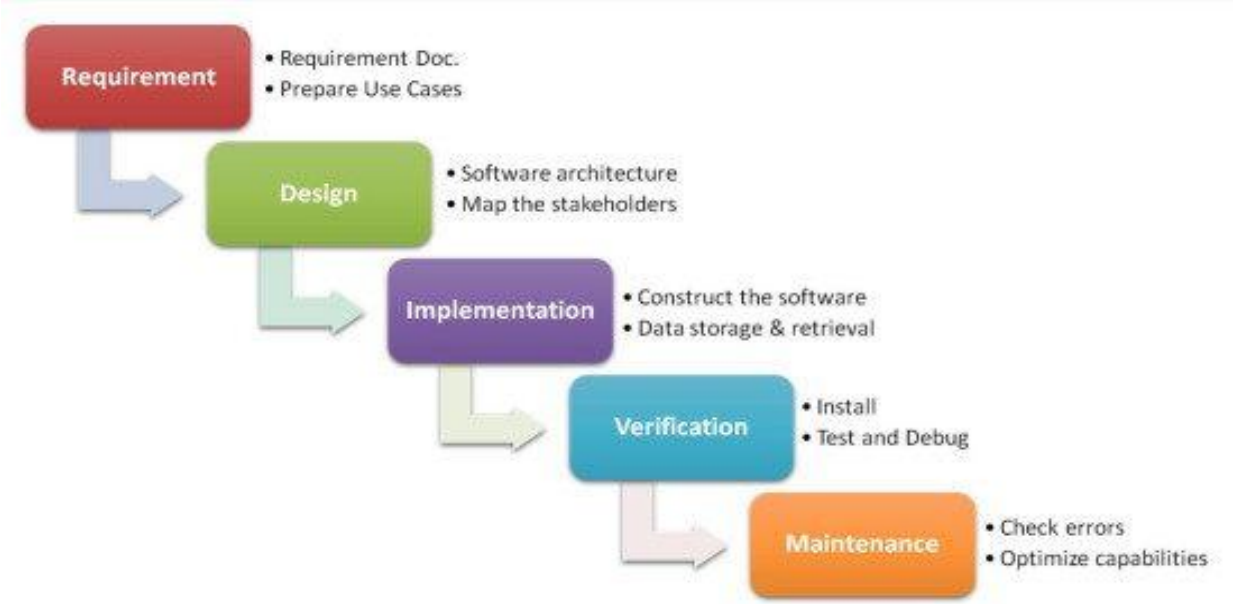
IV. **Integração e teste do sistema.** As unidades individuais do programa são integradas e testadas como um sistema completo para assegurar que os

requisitos do software tenham sido atendidos. Após o teste, o sistema de software é entregue ao cliente.

V. **Operação e manutenção.** A manutenção envolve a correção de erros que não foram descobertos em estágios iniciais do ciclo de vida, com melhoria da implementação das unidades do sistema e ampliação de seus serviços em resposta às descobertas de novos requisitos.

Na Figura 2, é apresentado o modelo de processo em Cascata.

Figura 2: Modelo de Processo - Cascata



Fonte: Sommerville (2011)

Segundo o autor, o modelo Cascata inicia de uma abordagem orientada a projetos para a Engenharia de Software. O projeto é considerado uma tarefa claramente delineada para a qual os resultados desejados podem ser determinados completamente e sem ambiguidade.

Segundo Sommerville (2011), podem-se destacar as seguintes vantagens do modelo Cascata:

I. **Estrutura clara e disciplinada**. O modelo em Cascata oferece uma sequência bem definida de fases, o que torna o processo de desenvolvimento fácil de

entender e seguir. Isso é especialmente útil em projetos menores ou quando os requisitos são bem definidos e estáveis desde o início.

II. **Fácil de gerenciar**. Como as etapas são realizadas de forma sequencial, o progresso do projeto pode ser facilmente monitorado. Cada fase tem um ponto de verificação claro, o que facilita a gestão e o acompanhamento do projeto.

III. **Documentação detalhada**. O modelo Cascata enfatiza a criação de documentação completa em cada fase, o que resulta em um registro detalhado dos requisitos, *design* e outras especificações do projeto. Isso é importante para a manutenção do sistema e para equipes que possam precisar dar continuidade ao trabalho após o término do desenvolvimento.

IV. **Adequado para projetos com requisitos bem definidos**. O modelo em Cascata é eficaz quando os requisitos do sistema são bem compreendidos e pouco sujeitos a mudanças ao longo do projeto. Isso é comum em sistemas de pequeno porte ou em projetos com objetivos bem claros e escopo fechado.

V. **Facilidade de implementação**. Como as etapas são rigorosamente separadas, os desenvolvedores podem se concentrar em uma fase de cada vez, o que pode resultar em uma implementação mais focada e menos propensa a erros.

Embora o modelo em Cascata tenha suas vantagens, também possui limitações, especialmente quando se trata de projetos complexos ou com requisitos sujeitos a mudanças frequentes. Para tais casos, abordagens mais ágeis ou iterativas, como o modelo ágil, são frequentemente preferidas. O modelo Cascata, embora seja um dos primeiros métodos estruturados de desenvolvimento de software, ainda é considerado útil em determinadas situações, principalmente em projetos com requisitos bem definidos e pouca necessidade de mudanças ao longo do processo. Sua principal vantagem reside na clareza e rigidez das etapas, o que facilita a gestão, a documentação e o acompanhamento do progresso do projeto. No entanto, sua aplicação é limitada em cenários mais dinâmicos e complexos, em que mudanças nos requisitos podem ocorrer de forma contínua.

A falta de flexibilidade e a dificuldade de adaptação a novos requisitos tornam o modelo Cascata menos adequado para ambientes ágeis e em constante evolução. Dessa forma, enquanto o modelo Cascata pode ser eficaz em contextos específicos, sua utilização deve ser cuidadosamente avaliada, especialmente à luz de metodologias mais modernas, como as abordagens ágeis, que oferecem maior flexibilidade e adaptabilidade aos desafios do desenvolvimento de software contemporâneo.

4.2.3 Modelo de Prototipação

De acordo com Pressman (2014), o modelo de prototipação envolve a criação de uma versão preliminar do software, que é usada para coletar *feedback* dos usuários e refinar os requisitos do sistema. A cada ciclo de *feedback*, o protótipo é aprimorado, até que o sistema atenda às expectativas dos usuários.

Para Yourdon (1990), Prototipação é gradual e evolutiva por parte do usuário e do desenvolvedor. Assim, obtém-se um conjunto inicial de necessidades e as implementam, rapidamente, com a intenção de refiná-las de acordo com o aumento do conhecimento do sistema por parte do desenvolvedor e do usuário.

É um processo interativo de geração de modelos de software que faz parte da análise do ciclo de vida dos sistemas. É a atividade de desenvolvimento de uma versão inicial do sistema baseada no atendimento dos requisitos ainda pouco definidos.

O modelo de Prototipação compreende várias técnicas visando à validação do cliente, que podem ser feitas repetidas vezes, de todos os aspectos ligados ao software: do *layout* às funcionalidades. A ideia é sempre identificar os eventuais problemas e as oportunidades de melhoria, a fim de compilar os raciocínios e otimizar o desenvolvimento.

Prototipação é o desenvolvimento rápido de um sistema. No passado, protótipo tinha a finalidade exclusiva de avaliar os requisitos, assim o desenvolvimento tradicional era necessário. Atualmente, os limites entre a prototipação e o desenvolvimento normal do sistema, muitas vezes, são indefinidos e muitos sistemas são desenvolvidos usando uma abordagem evolucionária.

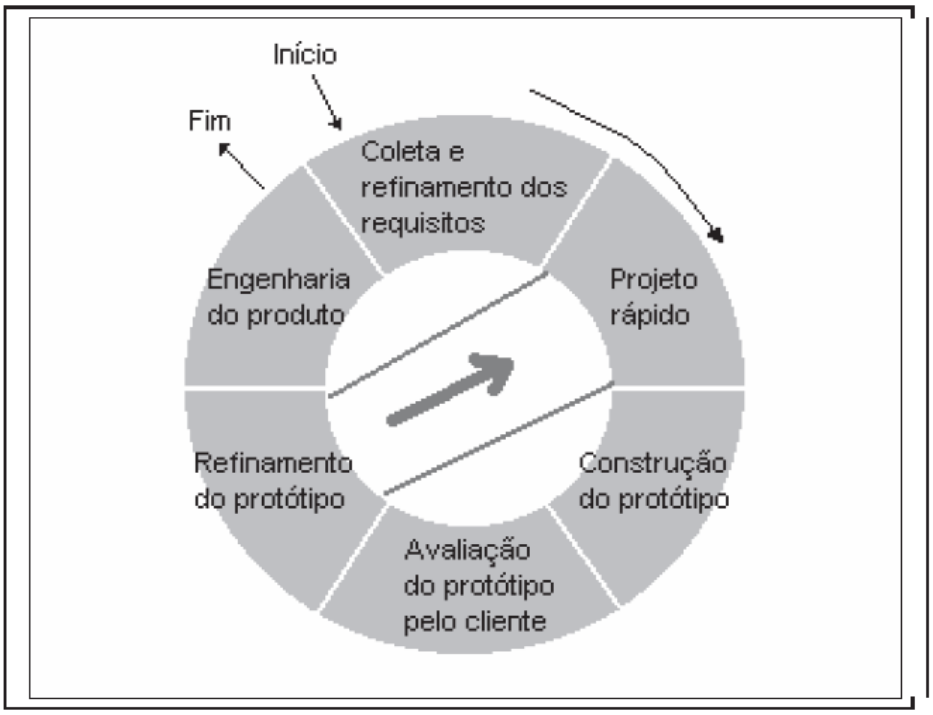
Uma distinção deve ser feita entre o protótipo e um sistema real. Sistemas reais têm a intenção de uso operacional e devem seguir determinados padrões no que diz

respeito a sua qualidade, segurança, desempenho, capacidade, robustez e facilidade de manutenção.

Em contraste, protótipos visam a clareza na visualização de determinados aspectos de um sistema sobre os quais há incerteza. Protótipos são usados para verificar a precisão das hipóteses feitas sobre esses aspectos.

Na figura 3 é apresentado o modelo de processo Prototipação, com os seguintes destaques:

Figura 3: Modelo de Prototipação



Fonte: CENTRAL DA ENGENHARIA. Paradigmas: Prototipação. Central da Engenharia, 09 fev. 2011. Disponível em: https://centraldaengenharia.wordpress.com/2011/02/09/paradigmas-prototipacao/ . Acessado em 20/04/2024

I. **Coleta dos requisitos.** Nessa fase, são identificados os requisitos básicos do sistema, sem entrar em detalhes excessivos. O objetivo é entender as funcionalidades mais importantes para os usuários e definir as principais necessidades. A coleta de requisitos pode ser feita por meio de reuniões com os *stakeholders*, entrevistas e análise de documentos existentes. O desenvolvedor e cliente definem os objetivos gerais do software, identificam quais requisitos são conhecidos e as áreas que necessitam de definições adicionais.

II. **Projeto rápido**. Com base nos requisitos iniciais, é criado um protótipo inicial do sistema. Esse protótipo não precisa ser funcional em sua totalidade, mas deve refletir as funcionalidades principais e servir como uma representação visual das ideias e conceitos do sistema. O protótipo pode ser feito de forma rápida, utilizando ferramentas simples e não precisa ser totalmente refinado, representação dos aspectos do software que são visíveis ao usuário (abordagens de entrada e formatos de saída), protótipos do sistema permitem aos usuários verem quão bem o sistema dá suporte a seu trabalho.

III. **Construção do protótipo.** É uma fase crítica dentro do processo de prototipação, pois é quando a primeira versão tangível do sistema é criada, permitindo que os usuários vejam e interajam com uma representação das funcionalidades do software. Essa etapa visa desenvolver rapidamente um modelo funcional, mas simplificado, que pode ser alterado e melhorado com base no *feedback* dos usuários. O principal objetivo da construção do protótipo é transformar requisitos iniciais e ideias em uma versão preliminar do sistema, que servirá como uma base para avaliações e iterações subsequentes. O protótipo não precisa ser uma versão completamente funcional ou final, mas deve ser suficientemente interativo e representativo das funcionalidades essenciais do sistema.

Características da construção do protótipo:

a. **Desenvolvimento rápido e simplificado.** O protótipo é criado de forma rápida, utilizando ferramentas de desenvolvimento ágil ou mesmo tecnologias mais simples, como protótipos em papel, *wireframes*, ou versões iniciais com funcionalidades básicas. A ideia é criar algo funcional, mas sem investir muito tempo em detalhes ou complexidade de implementação. O objetivo é mostrar as funcionalidades chave e permitir que os usuários possam interagir com elas.

b. **Foco nas funcionalidades principais.** Durante essa etapa, a equipe de desenvolvimento foca em implementar as funcionalidades mais importantes do sistema, aquelas que são essenciais para a operação do

software e que os usuários necessitam testar e avaliar. O protótipo não deve incluir todas as funcionalidades planejadas para o sistema final, mas sim um conjunto representativo das funcionalidades cruciais.

c. **Interatividade e *feedback* imediato.** O protótipo deve ser interativo, ou seja, o usuário deve ser capaz de realizar algumas ações no sistema, simulando a experiência final. A interatividade é fundamental para que os usuários possam fornecer *feedback* eficaz sobre a interface e a experiência do usuário. Isso ajuda a identificar problemas de usabilidade e a ajustar os requisitos de maneira mais precisa.

d. **Versões iterativas.** Em muitos casos, o protótipo é desenvolvido em versões sucessivas e iterativas. Após a construção inicial, o protótipo é apresentado aos usuários para coleta de *feedback.* Com base nas sugestões e críticas recebidas, a equipe de desenvolvimento faz ajustes e refinamentos no protótipo, resultando em uma nova versão que é novamente testada pelos usuários.

Importância da construção do protótipo**:**

a. **Visualização das ideias.** Os usuários conseguem visualizar e experimentar o sistema em sua forma inicial, o que facilita a compreensão do produto e ajuda a identificar falhas, problemas e melhorias.

b. **Ajuste contínuo.** O processo de construção e refinamento iterativo do protótipo possibilita uma constante adaptação às necessidades reais dos usuários, levando a um sistema mais alinhado com suas expectativas.

c. **Redução de riscos.** Ao identificar problemas ou requisitos mal definidos logo no início do desenvolvimento, a etapa de construção de protótipo reduz o risco de erros custosos e mudanças significativas nas fases posteriores do projeto.

IV. **Avaliação do protótipo pelo cliente.** Após a construção do protótipo inicial, ele é apresentado aos usuários ou *stakeholders* para coleta de *feedback*. Os usuários avaliam a interface, a usabilidade e as funcionalidades oferecidas e fornecem suas sugestões de melhorias ou ajustes. Essa etapa é crucial para identificar problemas e validar se o protótipo atende às necessidades reais dos usuários, validação de requisitos. O protótipo pode revelar erros e omissões nos requisitos, cliente e desenvolvedor avaliam o protótipo.

V. **Refinamento do protótipo.** Com base no *feedback* recebido, o protótipo é ajustado e refinado. As funcionalidades são aprimoradas, erros corrigidos e novas funcionalidades podem ser adicionadas, conforme a evolução do entendimento dos requisitos. Esse processo de refinamento pode ser repetido várias vezes, criando versões cada vez mais próximas do produto. Algumas partes dos requisitos (p. ex. funções críticas com relação à segurança) são difíceis de aparecer em protótipos, assim acabam não configurando na especificação.

VI. **Engenharia do produto.** Após várias iterações de prototipagem e refinamento, o sistema final é desenvolvido com base nas especificações definidas ao longo do processo. O protótipo se torna-se o sistema funcional, atendendo às expectativas dos usuários. Durante essa fase, o foco é garantir que o software seja robusto, escalável e de alta qualidade, para que possa ser colocado em produção. São identificados os requisitos, o protótipo deve ser descartado e a versão de produção deve ser construída considerando os critérios de qualidade.

Vantagens do modelo de prototipação:

I. ***Feedback* contínuo dos usuários**. A principal vantagem desse modelo é a possibilidade de obter *feedback* constante dos usuários, o que garante que o sistema seja desenvolvido de acordo com suas necessidades reais. Isso ajuda a evitar a implementação de funcionalidades desnecessárias e a garantir a satisfação do cliente.

II. **Redução de riscos**. Como o modelo permite o teste contínuo de funcionalidades, problemas e falhas podem ser identificados mais cedo no processo, reduzindo o risco de erros no produto.

III. **Maior envolvimento do cliente**. O cliente ou usuário tem a oportunidade de interagir com o protótipo desde as fases iniciais, o que aumenta o envolvimento e a confiança no processo de desenvolvimento.

IV. **Flexibilidade para mudanças nos requisitos**. O modelo de prototipação é altamente flexível e pode ser ajustado com facilidade à medida que os requisitos evoluem. Isso é particularmente útil em projetos em que os requisitos não estão completamente definidos ou podem mudar ao longo do tempo.

V. **Melhora na compreensão dos requisitos**. Ao interagir com o protótipo, os usuários conseguem visualizar de maneira mais clara como o sistema funcionará, o que facilita a compreensão e aprimoramento dos requisitos, que podem ser ajustados de acordo com as necessidades reais.

Apesar de suas vantagens, o modelo de prototipação pode apresentar desafios, como o risco de criar protótipos de baixa qualidade que não representem com precisão o sistema final, ou ainda, de gerar expectativas irrealistas nos usuários. Portanto, é essencial garantir que o protótipo evolua de forma planejada e que o *feedback* seja bem gerido.

O processo de prototipação é fundamental para compreender o propósito do software a ser desenvolvido. Após identificar os tipos de protótipos que melhor atendem às necessidades do projeto, é essencial manter um alinhamento contínuo com a equipe de desenvolvimento.

Isso porque detalhes mal definidos ou representados incorretamente no protótipo podem gerar retrabalho, elevar os custos e provocar atrasos significativos na entrega do produto.

4.2.4 Modelo Espiral De Boehm

O famoso modelo Espiral foi proposto por Boehm (1988), é um modelo de processo de software evolucionário que também é interativo como a prototipação, porém com aspectos sistemáticos e controlados do modelo Cascata. O modelo Espiral fornece um grande potencial para que permita o rápido desenvolvimento de versões cada vez mais completas. É um processo revolucionário, ou seja, adequado para softwares que precisam passar por inúmeras evoluções na medida em que o desenvolvimento acontece.

Sommerville (2011), relata que o modelo em Espiral combina prevenção e tolerância a mudanças, assume que mudanças são um resultado de riscos de projeto e inclui atividades explícitas de gerenciamento de riscos para sua redução.

É uma abordagem mais realística para o desenvolvimento de sistemas e de software em grande escala. A cada ciclo desenvolvido é feita a análise de riscos e, dependendo do resultado da análise, pode-se optar por não desenvolver algo no produto ou mesmo interromper o desenvolvimento. O modelo ajuda a envolver o cliente na avaliação e apresentação de sugestões durante o projeto, o cliente, também se envolve-se com o projeto e ajuda a minimizar os riscos no desenvolvimento por meio da prototipação.

Neste modelo, apenas o início é definido. A evolução e amadurecimento dos requisitos demandam tempo ajustável, assim como o custo. Isto torna o sistema difícil de ser vender ao cliente e exige um alto nível de gerenciamento em todo o processo.

É importante ressaltar que, a cada ciclo, o planejamento deve ser revisto com base no *feedback* do cliente, ajustando, inclusive, o número de iterações planejadas. De fato, este é o maior problema do ciclo de vida Espiral, ou de maneira geral, dos modelos evolucionários: a gerência de projetos.

Na Figura 4, é apresentado o modelo de processo em Espiral, com os seguintes destaques:

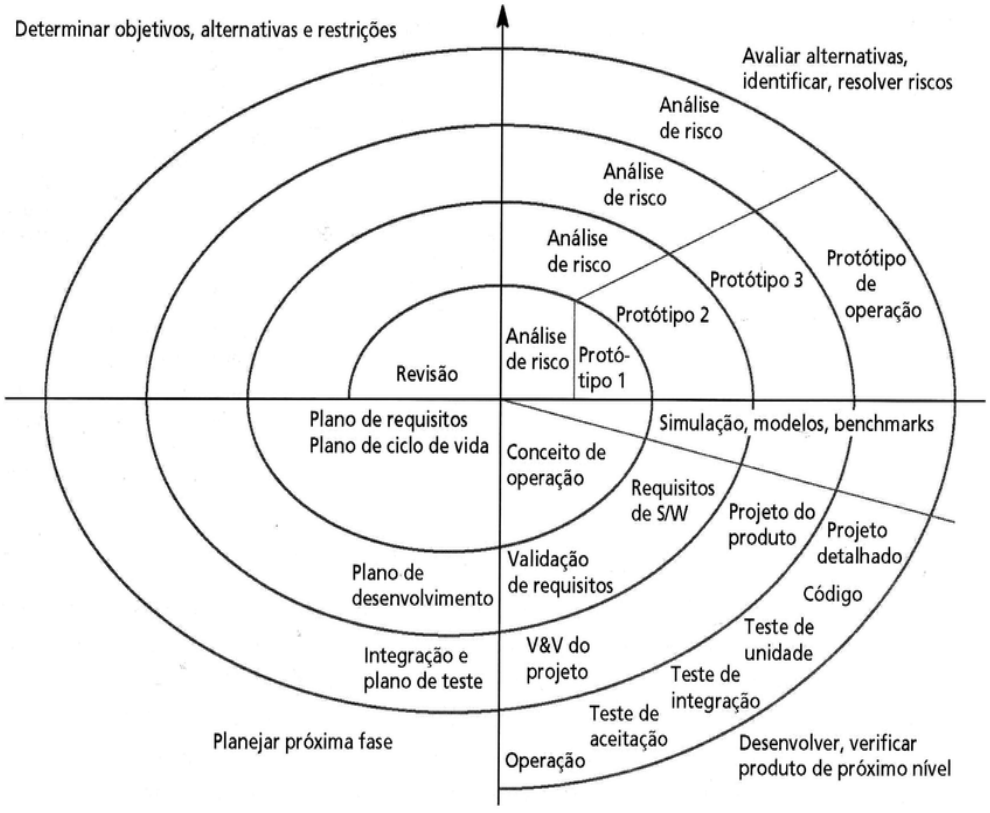
I. **Definição de objetivos.** Objetivos específicos para essa fase do projeto são definidos, restrições ao processo e ao produto são identificadas, e um plano de gerenciamento detalhado é elaborado, sendo os riscos do projeto são identificados. Podem ser planejadas estratégias alternativas em função desses riscos.

II. **Avaliação e redução de riscos.** Para cada um dos riscos identificados do projeto é feita uma análise detalhada. Medidas para redução do risco são tomadas. Por exemplo, se houver risco de os requisitos serem inadequados, um protótipo de sistema pode ser desenvolvido.

III. **Desenvolvimento e validação.** Após a avaliação dos riscos, é selecionado um modelo de desenvolvimento para o sistema. Por exemplo, a prototipação descartável pode ser a melhor abordagem de desenvolvimento de interface de usuário se os riscos forem dominantes.

IV. **Planejamento.** O projeto é revisado e uma decisão é tomada a respeito da continuidade do modelo com mais uma volta da Espiral. Caso se decida pela continuidade, planos são elaborados para a próxima fase do projeto.

Figura 4: Modelo Espiral de Boehm



Fonte: Boehm, B. (s.d.). Figura 20 – Modelo de processo de desenvolvimento em Espiral proposto por Boehm . https://www.researchgate.net/figure/Figura-20-Modelo-de-processo-de-desenvolvimento-em-Espiral-proposto-por-Boehm_fig15_317087042

Acessado em 30/04/2024

Vantagens do modelo Espiral.

I. **Redução de riscos:** O modelo Espiral coloca uma ênfase significativa na identificação e mitigação de riscos logo nas fases iniciais do projeto. Ao abordar os riscos em ciclos repetidos, o modelo permite que problemas sejam detectados precocemente e corrigidos antes que se tornem mais graves.

II. **Flexibilidade e adaptação a mudanças.** Devido à natureza iterativa do modelo, o processo de desenvolvimento pode ser adaptado conforme os requisitos mudam. O *feedback* contínuo dos *stakeholders* permite que ajustes sejam feitos a cada ciclo, o que o torna altamente flexível, especialmente em projetos com requisitos volúveis.

III. **Entrega Incremental de funcionalidades.** O modelo Espiral permite a entrega Incremental do software, com cada ciclo resultando em uma versão funcional do sistema. Isso facilita a avaliação do progresso e a validação das funcionalidades, além de permitir que o software seja disponibilizado aos usuários mais cedo.

IV. **Envolvimento constante dos *stakeholders*.** O modelo incentiva a comunicação constante com os *stakeholders*, o que assegura que o produto esteja alinhado com as expectativas dos usuários e às necessidades do negócio.

V. **Adequado para projetos grandes e complexos.** O modelo Espiral é particularmente útil em projetos de grande porte e complexidade, nos quais os riscos são mais elevados. Sua abordagem iterativa permite que o projeto seja tratado em partes menores, o que facilita o gerenciamento e o controle.

O modelo usa uma abordagem “evolucionária” à Engenharia de Software, capacitando o desenvolvedor e o cliente a entenderem e reagirem, aos riscos em cada fase evolutiva. O modelo Espiral usa a prototipação como um mecanismo de redução de riscos, mas o que é mais importante, possibilita que o desenvolvedor aplique a abordagem de prototipação em qualquer etapa da evolução do produto.

Embora o modelo Espiral tenha muitas vantagens, ele também pode ser mais complexo de gerenciar em comparação com outros modelos, devido à necessidade de uma constante avaliação de riscos e ao foco na prototipação. Além disso, a necessidade de recursos significativos em termos de tempo e esforço para realizar as avaliações de risco podem ser um desafio em projetos com orçamento limitado.

4.3 O Impacto no Ensino de Engenharia de Software

Boehm (1988) descreve os modelos tradicionais e como o modelo em Cascata, a Prototipação e o modelo Espiral, continuam a ser incorporados ao currículo de Engenharia de Software, porém em uma perspectiva mais integrada e adaptativa, servindo como base conceitual para compreender a evolução das práticas de desenvolvimento

Para Pressman e Maxim (2014), as universidades particulares desempenham um papel relevante na formação de engenheiros de software, uma vez que refletem as necessidades do mercado e acompanham as tendências educacionais. Nesses contextos, o plano de ensino da disciplina tende a ser marcado por maior flexibilidade curricular, favorecendo a adaptação às novas demandas do setor de tecnologia

Para os autores, Boehm (1988) e Royce (1970), a incorporação de modelos tradicionais no ensino de Engenharia de Software é uma prática que enriquece a formação dos estudantes, permitindo que desenvolvam uma compreensão sólida dos processos de desenvolvimento de software, entre eles podem ser destacados:

- **Fundamentos Teóricos**. Os modelos tradicionais fornecem uma base teórica sólida sobre o ciclo de vida do desenvolvimento de software. O ensino desses modelos permite que os estudantes compreendam as etapas do desenvolvimento, desde a concepção até a manutenção (Sommerville, 2011).

- **Análise Crítica**. Ao estudar os modelos tradicionais, os estudantes aprendem a analisar suas limitações e pontos fortes. Isso é fundamental para desenvolver uma mentalidade crítica e a capacidade de escolher o modelo mais adequado para diferentes projetos. (Royce, 1970).

- **Integração com Práticas Ágeis.** Muitos currículos têm adotado uma abordagem híbrida, que combina práticas tradicionais com metodologias ágeis. Essa integração ajuda os estudantes a entenderem como os modelos clássicos podem ser adaptados a contextos dinâmicos e de rápida mudança.

- **Trabalho em Grupo e Projetos Práticos**. Os cursos frequentemente incluem projetos em equipe em que os estudantes aplicam modelos tradicionais, permitindo a vivência de um ciclo de vida de desenvolvimento real. Isso ajuda a consolidar o aprendizado teórico na prática.

- **Tecnologia e Ferramentas**. A incorporação de ferramentas modernas que suportam os modelos tradicionais (como ferramentas de gestão de projetos e versionamento) é comum. Isso garante que os estudantes não apenas entendam a teoria, mas também saibam como aplicá-la usando tecnologias atuais.

Para o MEC (2016), o impacto do ensino de Engenharia de Software nas universidades particulares é evidenciado pela incorporação de metodologias pedagógicas inovadoras, pela ênfase no desenvolvimento de competências diversificadas e pela intensificação da colaboração com o mercado de trabalho e a integração de laboratórios e ambientes de simulação que possibilitam a aplicação prática dos conhecimentos teóricos. Universidades particulares frequentemente investem em parcerias com empresas de tecnologia, proporcionando estágios e experiências reais que enriquecem o aprendizado. Podem-se citar entre alguns fatores:

- **Preparação para o Mercado.** O conhecimento em modelos tradicionais prepara os estudantes para as exigências do mercado de trabalho. Como afirmam Parnas e Clements (2001), os empregadores frequentemente valorizam profissionais que têm uma compreensão abrangente dos métodos de desenvolvimento, pois isso se traduz em melhores práticas de trabalho.

- **Desenvolvimento de Habilidades de Pensamento Crítico.** O ensino dos modelos tradicionais estimula o pensamento crítico. Segundo Hazzan e Dubinsky (2008), a capacidade de questionar e avaliar diferentes abordagens é essencial para a formação de um engenheiro de software inovador.

- **Inovação no Campo.** Compreender as teorias clássicas permite que os estudantes inovem. Boehm (2002) argumenta que a inovação no desenvolvimento de software muitas vezes surge da combinação de práticas tradicionais com novas abordagens e tecnologias.

A crescente demanda por profissionais qualificados em Engenharia de Software tem provocado mudanças significativas nas abordagens pedagógicas adotadas pelas universidades particulares. Este segmento educacional, por sua flexibilidade e capacidade de inovação, busca alinhar seus currículos às necessidades do mercado e às novas tendências tecnológicas.

4.4 Metodologias Ágeis

As metodologias ágeis, ganharam força após 2001, por um grupo de desenvolvedores de software, estabeleceu uma nova abordagem para o desenvolvimento de software, priorizando flexibilidade, colaboração e entrega contínua de valor que propõe uma abordagem mais dinâmica, que valoriza a colaboração e a adaptação às mudanças. Surgiu como uma resposta às limitações dos métodos tradicionais de desenvolvimento de software, como o modelo em Cascata.

Para Beck et. al (2001) os princípios fundamentais das metodologias ágeis são essenciais para entender como esses valores se traduzem em práticas concretas em que foram desenvolvidos os doze princípios que fundamentam essa filosofia.

O Manifesto Ágil (Manifesto para desenvolvimento ágil de software, 2001) é estruturado em torno de quatro valores centrais:

1. **Indivíduos e interações mais que processos e ferramentas.** A comunicação e a colaboração entre membros da equipe são mais importantes do que seguir rigidamente processos ou utilizar ferramentas específicas.

2. **Software em funcionamento mais que documentação abrangente**. O foco está em entregar software funcional, em vez de se perder em documentação extensa que pode não ser útil.

3. **Colaboração com o cliente mais que negociação de contratos**. A parceria e o diálogo contínuo com o cliente são essenciais para garantir que o produto atenda às suas necessidades.

4. **Responder a mudanças mais que seguir um plano**. A capacidade de adaptação é fundamental, permitindo que a equipe reaja a novas informações e requisitos à medida que surgem.

Highsmith (2002) relata que após a elaboração dos quatro valores do Manifesto Ágil, o respectivo grupo trabalhou coletivamente para definir os 12 princípios, que refletem suas visões sobre melhores práticas. Os princípios foram publicados junto com o Manifesto Ágil em janeiro de 2001, em uma página *web* criada pelo próprio grupo. Essa estrutura clara e concisa rapidamente ganhou aceitação e se tornou a base para diversas metodologias ágeis que surgiram posteriormente.

Os doze princípios dos Manifestos Ágeis são descritos como se segue:

1. **Satisfação do Cliente**. O objetivo primário é satisfazer o cliente por meio da entrega contínua de software valioso.
2. **Mudanças Bem-vindas.** As equipes devem ser flexíveis e dispostas a aceitar mudanças nos requisitos, mesmo em estágios avançados do desenvolvimento.
3. **Entregas Frequentes**. O software deve ser entregue de forma frequente, com entregas regulares que vão de semanas a meses.
4. **Colaboração Diária**. Deve haver uma colaboração constante entre as partes interessadas e a equipe de desenvolvimento, com comunicação clara e frequente.
5. **Ambientes de Trabalho Sustentáveis**. O ritmo de trabalho deve ser sustentável, permitindo que as equipes mantenham um equilíbrio saudável entre trabalho e vida pessoal.

6. **Excelência Técnica e Design**. A qualidade do design e das práticas técnicas é fundamental para melhorar a agilidade e a capacidade de resposta.
7. **Simplicidade**. O foco deve estar em maximizar o trabalho não realizado, concentrando-se nas funcionalidades que realmente trazem valor.
8. **Reflexão e Ajuste**. As equipes devem refletir regularmente sobre como se tornam mais eficazes e ajustar seu comportamento e processos conforme necessário.
9. **Desenvolvimento Iterativo**. O desenvolvimento deve ser feito em ciclos curtos, permitindo iterações e melhorias contínuas com base no *feedback*.
10. **Autonomia da Equipe**. As equipes devem ter a liberdade de auto-organizar-se e tomar decisão sobre como melhor realizar seu trabalho.
11. **Valor de Negócio**. A priorização das funcionalidades deve ser baseada no valor que elas trazem ao negócio, permitindo decisões mais informadas sobre o que desenvolver a seguir.
12. **Engajamento e Motivação**. O engajamento da equipe é crucial. Criar um ambiente que incentive a motivação e a responsabilidade é fundamental para o sucesso do projeto.

Os 12 princípios do Manifesto Ágil representam uma mudança cultural no desenvolvimento de software, promovendo uma abordagem mais colaborativa e responsiva. Eles continuam a ser uma referência fundamental para equipes que buscam implementar práticas ágeis eficazes em seus projetos.

O Manifesto Ágil é fundamental em um ambiente de negócios dinâmico, em que a velocidade de mudança é alta. Permite que as organizações respondam rapidamente às necessidades dos clientes e do mercado, promovendo um ciclo contínuo de *feedback* e adaptação. Isso resulta em produtos que não apenas atendem, mas superam as expectativas dos clientes. Os princípios do Manifesto Ágil formam a base para diversas metodologias ágeis, como *SCRUM* e *Extreme Programming* (XP). A adoção dessas práticas pode levar a melhorias significativas na eficiência, na qualidade do produto e na satisfação do cliente.

A seguir serão apresentadas as características e aplicações sobre os modelos ágeis, suas vantagens, limitações e contextos em que são mais adequados.

4.4.1 Modelo Ágil *SCRUM*

Esse é um termo já bem popular no mundo da tecnologia, sendo largamente utilizado por times de desenvolvimento de software. O *SCRUM*, apesar do nome estranho, traz uma abordagem à gestão de projetos e foi desenvolvida por Ken Schwaber e Jeff Sutherland, em 1995, na conferência OOPSLA (Schwaber; Sutherland, 1997).

SCRUM é um *framework* que permite que pessoas, equipes e organizações gerenciem trabalho complexo, com foco em entregar valor de maneira Incremental e iterativa. (SCRUM GUIDE, 2020)

Para Schwaber & Sutherland, (2017), o *SCRUM* é um modelo de desenvolvimento ágil de software que fornece métodos para se definir o planejamento, os principais papéis de pessoas e a forma de trabalho do time. O nome *SCRUM* é derivado de uma jogada de rúgbi, em que todos do mesmo time avançam como apenas uma unidade, trabalhando com os mesmos jogadores e em conjunto, passando sempre a bola para um e para outro.

Ainda para Schwaber & Sutherland, (2017), a ideia do *SCRUM* é justamente definir papéis bem específicos para as pessoas envolvidas no projeto e como cada pessoa vai jogar. Dessa forma cada uma vai ter que fazer com que o time siga em frente no jogo: no caso é o próprio desenvolvimento do software. Tem como base as atividades estruturadas como o levantamento de requisitos, a análise do projeto, o projeto em si, a busca da evolução do software e a entrega final para o cliente.

As equipes de projetos geridos com a metodologia *SCRUM* são compostas basicamente por três papéis: *Product Owner, SCRUM Master* e Time de Desenvolvimento.

Em 2011, Cohn, um dos maiores especialistas em metodologias ágeis, especialmente *SCRUM*, apresentou uma visão detalhada sobre o *framework SCRUM* no contexto do desenvolvimento ágil.

A definição das etapas do *SCRUM*, conforme explicada por Mike Cohn, enfatiza as práticas essenciais e os papéis envolvidos para alcançar uma entrega bem-sucedida.

Conforme será visto na Figura 5, o processo *SCRUM* é composto pelas etapas definidas como: papéis, eventos e artefatos principais, com as funções de:

1. **Papéis**

- *SCRUM* Master: Facilita a implementação do *SCRUM* e remove obstáculos para a equipe.
- *Product Owner:* Responsável por gerenciar o *backlog* do produto e priorizar as funcionalidades a serem desenvolvidas.
- Equipe de Desenvolvimento: Profissionais que trabalham para entregar o produto no final de cada *sprint.*

2. **Eventos**

- *Sprint:* Ciclo de trabalho, com duração fixa (normalmente 2-4 semanas).
- Reunião de Planejamento da *Sprint*: Planejamento das atividades para a Sprint.
- *Daily SCRUM*: Reunião diária para alinhamento de tarefas.
- Revisão da *Sprint:* Avaliação dos resultados ao final de cada *Sprint*.
- Retrospectiva da *Sprint:* Reflexão sobre o processo e melhoria contínua.

3. **Artefatos**

- *Product Backlog*: Lista priorizada de tudo o que precisa ser feito no produto.
- *Sprint Backlog:* Lista das tarefas que serão realizadas durante a *Sprint.*
- Incremento: Resultado da *Sprint*, que deve ser um produto utilizável e potencialmente entregável.

Está descrito o significado de cada etapa do *SCRUM* a seguir:

I. ***Sprints*.** É o nome dado para os ciclos de cada projeto. Em geral são ciclos mensais e são determinados para que as tarefas sejam realizadas.

II. ***Product Backlog*.** É o nome dado para o conjunto de objetivos de um projeto. No caso de um projeto de desenvolvimento de software, é o nome dado ao pacote de funcionalidades a serem desenvolvidas em um projeto.

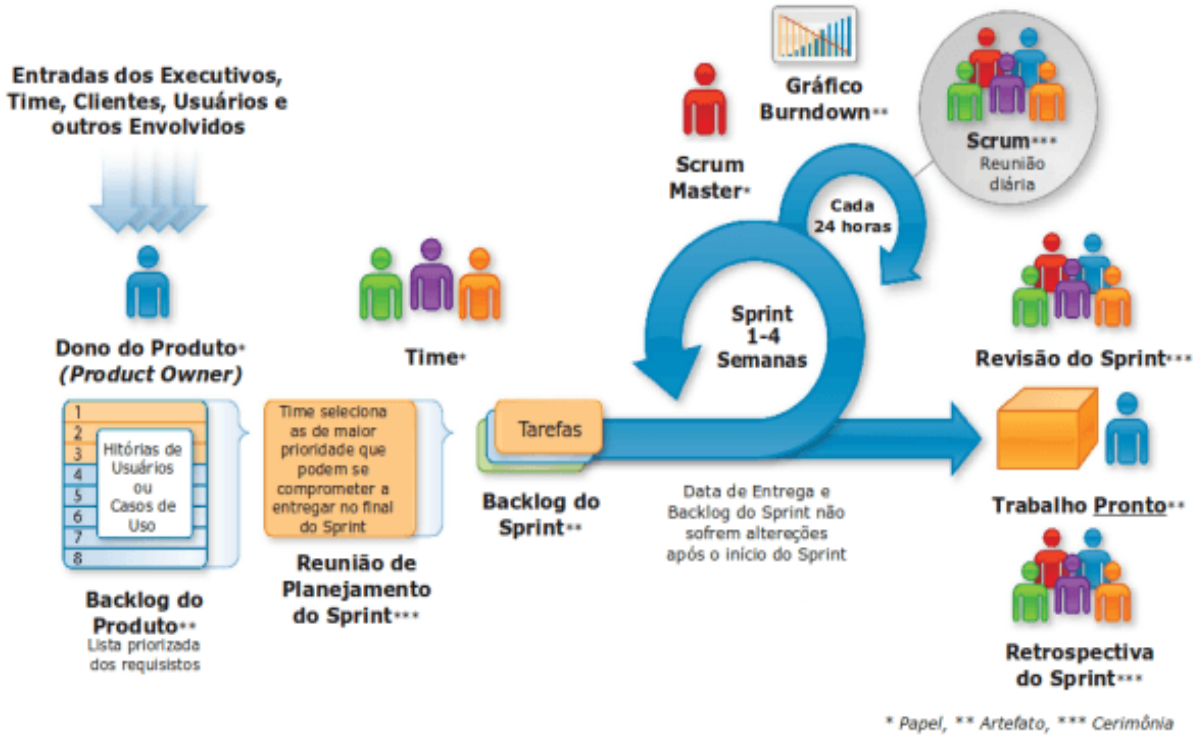
III. ***Sprint Planning Meeting*.** São reuniões periódicas que acontecem no início de cada *sprint*, ou ciclo, para planejar e priorizar os itens do *Product Backlog* que serão desenvolvidos naquele período.

IV. ***Sprint Backlog*.** É como são chamadas as tarefas específicas que serão realizadas e desenvolvidas em cada ciclo, ou *sprint*.

V. ***Daily SCRUM*.** Reunião diária para acompanhamento do projeto. A ideia é que toda a equipe se reúna diariamente para discutir as atividades desenvolvidas, disseminar conhecimento, identificar impedimentos e priorizar o trabalho daquele dia.

VI. ***Sprint Review Meeting*.** Reunião que acontece ao final de cada *sprint* para que a equipe apresente o que foi realizado e os resultados do trabalho daquele ciclo.

Figura 5: Modelo do processo *SCRUM*



Fonte: iMasters. (2016). SCRUM como Metodologia Ágil na produção de jogos digitais. https://imasters.com.br/agile/scrum-como-metodologia-agil-na-producao-de-jogos-digitais Acessado em 30/04/2024

Segundo Cohn (2011), "durante a *Sprint Review,* o objetivo é reunir as partes interessadas para entender o progresso, compartilhar a visão e ajustar as expectativas conforme necessário."

> Para CONH, destaca que a qualidade do planejamento inicial é crucial para o sucesso da Sprint. Ele explica: "Um bom planejamento da Sprint ajuda a garantir que o trabalho que será realizado tenha um valor claro e significativo para o cliente, com um entendimento compartilhado do que precisa ser feito. (Cohn, Agile Estimating and Planning, 2011).

4.4.2 Modelo Ágil - *Extreme Programing (XP)*

Segundo Sommerville (2011), gerenciamento ágil de projeto é um conjunto de princípios de desenvolvimento, os quais tendem a facilitar a criação de sistemas.

Nos dias de hoje, as empresas operam em um ambiente global, com mudanças rápidas. Softwares fazem parte de quase todas as operações de negócios, assim, novos softwares são desenvolvidos rapidamente para obterem proveito de novas oportunidades e responder às pressões competitivas.

Os métodos ágeis surgiram de profissionais que vieram da orientação a objetos como Kent Beck e Ward Cunningham, duas grandes personalidades no que tange o tema de métodos ágeis. *Extreme Programming* (XP) é talvez o mais conhecido e mais utilizado dos métodos ágeis. Criado por Kent Beck nos anos 1990, o XP busca melhorar a qualidade do software e a capacidade de resposta às mudanças nas necessidades dos clientes.

O XP funciona baseado em voltas de *feedback,* ou seja, retornos sobre cada entrega. Para conseguir se adaptar às mudanças, o XP preconiza ciclos curtos que dão previsibilidade e redução de incertezas ou riscos, simplificam e geram melhorias constantes de código para facilitar a mudança a partir de testes automatizados e integração contínua, aumentando a confiança.

Quando o novo código é integrado ao sistema, todos os testes devem ser executados com sucesso. Há um curto intervalo entre os *releases* do sistema.

Existem 3 fatores determinantes para a adoção de metodologias ágeis em projetos de Tecnologia que são: a redução *time to Market*, a transformação digital e o aumento da produtividade.

Os métodos ágeis são métodos de desenvolvimento Incremental em que os incrementos são pequenos e, normalmente, as novas versões do sistema são criadas

e disponibilizadas aos clientes a cada duas ou três semanas. Elas envolvem os clientes no processo de desenvolvimento para obter *feedback* rápido sobre a evolução dos requisitos. Assim, minimiza-se a documentação, pois se utiliza mais a comunicação informal do que reuniões formais com documentos escritos.

Supostamente, a documentação formal deve descrever o sistema e, assim, tornar sua compreensão mais fácil para as pessoas que fazem as mudanças. Porém, na prática, a documentação formal nem sempre é atualizada, e, portanto, não reflete exatamente o código do programa.

Mais importante do que descrever as etapas a seguir é mostrar os 5 conceitos que fazem dar certo o Modelo Ágil. A programação extrema inicialmente reconheceu 4 valores em 1999 e, na segunda edição foi incorporado mais um valor (Respeito):

1. **Comunicação.** Enfatiza que se deve sempre estarem comunicando, seja com desenvolvedores ou com os clientes;

2. **Simplicidade.** Fazer o mais simples possível e, caso seja necessário, deixar algo mais complexo para depois;

3. ***Feedback*.** É o valor primordial dentro do desenvolvimento ágil. O *feedback* deve ser logo no início, para se saber se está sendo feita a coisa correta, precisa ser concreto perguntando diretamente ao código e precisa ser constante por meio de iterações curtas, incrementos e releases;

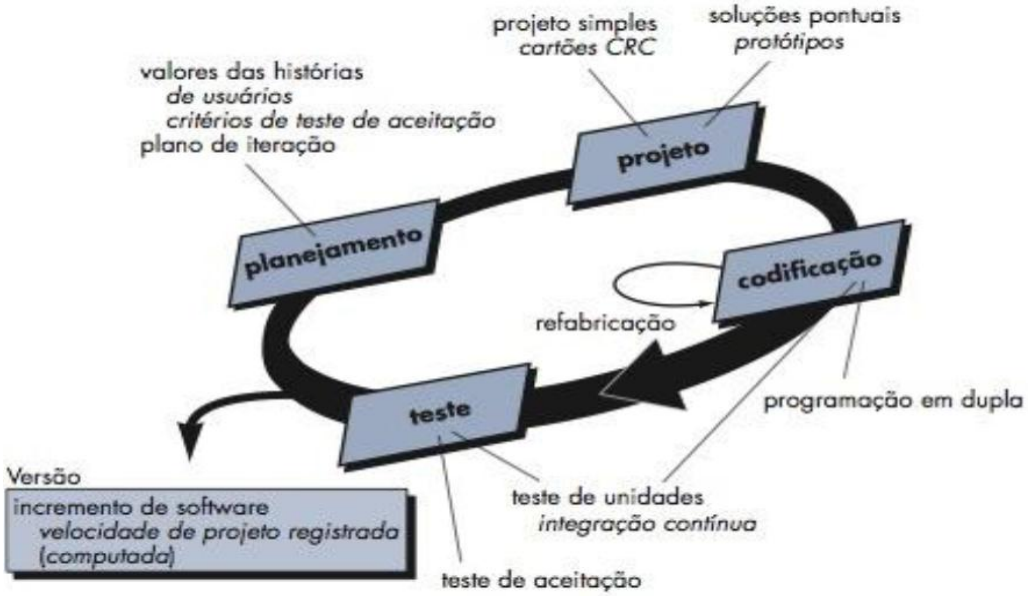
4. **Coragem (Disciplina).** XP diz que se deve ter coragem de sempre colocar o cliente a par do que está acontecendo, como por exemplo, acreditar na capacidade de reagir a mudanças; trocar de paradigma; aprender com os erros; jogar fora código ruim; entre outros;

5. **Respeito.** Responsável por garantir a aderência a esses valores nas práticas.

Conforme a figura 6, o XP tem algumas particularidades interessantes. É confirmado pelo Beck (2004), que descreve que um desses benefícios é a agilidade no Planejamento (*Planning Games*) de não definir uma especificação completa e formal dos requisitos, ao contrário das metodologias tradicionais. Outro é a produção

de sistemas simples que atendam aos atuais requisitos, não tentando antecipar o futuro e permitindo atualizações frequentes em ciclos bastante curtos.

Figura 6: Modelo Ágil XP



Fonte: MACHLIZADEVI. Extreme Programming: Pilihan Metodologi Pengembangan Sistem. Blog BINUSIAN, 17 abr. 2014. Disponível em: https://machlizadevi.blog.binusian.org/2014/04/17/extreme-programming-pilihan-metodologi-pengembangan-sistem/. Acessado em 05/05/2024

As características principais do *Extreme Programming* (XP):

I. **Desenvolvimento Iterativo e Incremental.** Como outras metodologias ágeis, o XP trabalha em ciclos curtos de desenvolvimento, em que o software é entregue de forma Incremental, permitindo ajustes rápidos com base no *feedback* contínuo.

II. **Desenvolvimento Colaborativo**. A interação constante entre os membros da equipe é um princípio central do XP. Ele promove o trabalho conjunto entre desenvolvedores, clientes e *stakeholders.*

III. **Simplicidade.** O XP busca criar soluções simples, que atendam exatamente às necessidades do cliente e evitem funcionalidades desnecessárias.

IV. ***Feedback* Contínuo**. A metodologia foca em um ciclo constante de testes e revisões, com *feedback* rápido, o que facilita a identificação de problemas de forma antecipada.

V. **Comunicação constante**. A comunicação entre os membros da equipe é essencial. Isso inclui reuniões diárias (ou similares) e outros métodos para garantir que todos estejam alinhados.

VI. **Testes Automatizados**. O XP utiliza testes automatizados em todo o ciclo de desenvolvimento, garantindo que o código se mantenha funcional mesmo após modificações.

VII. **Programação em Par**. A técnica de programação em par, em que dois desenvolvedores trabalham juntos em uma única estação de trabalho, é um pilar do XP, promovendo a colaboração e a revisão contínua do código.

As vantagens do modelo *Extreme Programming* (XP) entre os mais utilizados no dia a dia são:

- **Alta Qualidade do Código**. O XP utiliza testes automatizados e revisões contínuas para garantir que o código seja de alta qualidade. Segundo Kent Beck:

> A prática de programação em par permite que dois desenvolvedores compartilhem conhecimentos, reduzindo erros e melhorando a qualidade do código. Além disso, os testes automatizados garantem que o código seja testado de forma contínua. (Beck, Extreme Programming Explained, 2004)

- **Capacidade de Responder Rápido às Mudanças**. Uma das principais vantagens do XP é sua flexibilidade em se adaptar rapidamente às mudanças nas necessidades do cliente. Como os ciclos são curtos e o *feedback* é constante, mudanças no projeto podem ser implementadas rapidamente.

A capacidade de responder rapidamente às mudanças no código e no *design* é o que torna o *Extreme Programming* tão eficaz em projetos de desenvolvimento dinâmicos. (Beck, *Extreme Programming Explained*, 2004)

- **Melhor Colaboração e Comunicação entre Equipe e Cliente**. O XP promove uma colaboração constante com os clientes, o que garante que as expectativas sejam sempre alinhadas. As reuniões diárias e o *feedback* contínuo garantem que a equipe de desenvolvimento e o cliente estejam sempre em sintonia sobre o progresso do projeto.

A comunicação entre clientes e desenvolvedores é central no XP, pois ajuda a esclarecer as necessidades reais e a ajustar o desenvolvimento com base nas mudanças frequentes de requisitos. (Cunningham, *The XP Series*, 2001)

- **Desenvolvimento de Software Mais Rápido e Eficiente**. O XP ajuda a reduzir a sobrecarga do desenvolvimento, eliminando funcionalidades desnecessárias e mantendo o foco no que realmente é necessário para atender ao cliente. O uso de programação simples e refatoração constante permite que a equipe entregue valor de forma rápida e eficiente.

A simplicidade no design e a refatoração contínua permitem que as equipes agilizem o processo de desenvolvimento sem comprometer a qualidade do produto. (Beck, *Extreme Programming Explained*, 2004)

- **Aumento da Moral e Satisfação da Equipe.** A prática de programação em par e o *feedback* constante ajudam a aumentar a moral da equipe, promovendo um ambiente de trabalho colaborativo e engajado. A troca de ideias e a parceria no desenvolvimento criam um sentimento de pertencimento e propósito entre os membros da equipe.

A prática de programação em par não apenas melhora a qualidade do código, mas também aumenta o engajamento dos desenvolvedores, pois eles aprendem uns com os outros e se ajudam mutuamente." (Beck, *Extreme Programming Explained*, 2004)

- **Entrega Contínua de Valor ao Cliente**. Como o desenvolvimento é feito de forma iterativa e Incremental, os clientes podem começar a usar e avaliar o

software em suas versões mais simples logo no início do projeto, o que garante que o valor seja entregue de forma contínua e constante.

O *feedback* rápido e contínuo durante o ciclo de desenvolvimento assegura que o cliente possa ver o progresso de forma tangível, com entregas constantes de valor. (Cunningham, *The XP Series*, 2001)

O *Extreme Programming* (XP) é uma abordagem ágil que coloca ênfase na qualidade, simplicidade, colaboração e flexibilidade, garantindo que as equipes possam se adaptar rapidamente às mudanças e entregar produtos de alto valor ao cliente. Suas práticas, como a programação em par, testes contínuos e *feedback* constante, tornam o processo de desenvolvimento mais eficiente e colaborativo, gerando resultados mais rápidos e satisfatórios para todos os envolvidos.

4.5 Causas dos problemas dos Softwares só utilizando o Modelo Ágil

Sobre a pesquisa publicada em dezembro de 2023 conduzida pelo Dr. Ali e JL Partners para Haystack Analytics (Ali, 2023), pode-se citar:

I. 70% dos projetos de software não são entregues no prazo, de acordo com pesquisas científicas, apesar de 83% dos desenvolvedores dizerem que o desempenho da entrega é importante.

II. 90% dos líderes empresariais dos EUA relatam a entrega no prazo como o fator-chave na avaliação do desempenho da Engenharia de Software e 89% estão preocupados com a entrega pontual de projetos de software em suas organizações.

A mesma pesquisa também descobriu que 77% dos tomadores de decisão de negócios no Reino Unido e 90% nos EUA relatam a entrega no prazo como a principal forma de medir o desempenho das equipes de Engenharia de Software. Apesar disso, 81% dos tomadores de decisão de negócios no Reino Unido e 89% nos EUA estão preocupados com a entrega pontual de projetos de software em suas organizações.

> A maioria dos projetos de software não cumpre seus prazos, paralisando o crescimento e corroendo a confiança. Haystack aborda esse problema generalizado. Nossa plataforma Software Delivery Ops transforma dados em

> ações decisivas, virando a maré na taxa de atraso de 70% do projeto. Com o Haystack, as equipes cumprem prazos de forma consistente, impulsionam o sucesso dos negócios e redefinem os padrões de entrega de software. (Rahul Singh, fundador e CEO da Haystack, citado por Ali, 2023)

A pesquisa também revelou que 65% dos projetos de software que adotam práticas de engenharia de requisitos ágeis não são entregues no prazo e dentro do orçamento, com um alto padrão de qualidade. Por outro lado, os projetos que adotam uma nova abordagem de Engenharia, falharam apenas 10% das vezes (Ali, 2023).

A significância estatística dos dados do estudo dos projetos mostra que o uso das práticas de uma nova Engenharia de Software está sendo denominada de modelo híbrido.

Três das quatro práticas listadas no Manifesto Ágil são "Software funcional sobre documentação abrangente", "Colaboração com o cliente sobre negociação de contratos" e "Responder à mudança sobre seguir um plano".

No entanto, descobriu que os projetos que tinham uma especificação ou requisitos documentados antes do início do desenvolvimento tinham 50% mais chances de sucesso do que aqueles que não tinham.

Já os projetos que tinham requisitos claros antes de iniciar o desenvolvimento tinham 97% mais chances de sucesso e os projetos que não exigiam mudanças significativas nos requisitos no final do processo de desenvolvimento tinham 7% mais chances de sucesso e projetos em que os requisitos eram baseados em um problema do mundo real tinham 54% mais chances de sucesso do que aqueles que não o faziam. Essa etapa do desenvolvimento, o levantamento de requisitos, vem do modelo tradicional, principalmente o modelo Cascata.

4.5.1 Quadros de Pesquisa e Dados

No período de 3 a 7 de maio 2024, a J.L. Partners solicitou respostas de 481 engenheiros de software que encontraram um projeto bem-sucedido pela última vez e 119 que encontraram um projeto fracassado pela última vez, para permitir uma análise suficiente de ambos os grupos.

Conforme quadro 1, as seguintes práticas de Engenharia de Software tiveram os seguintes aumentos nas taxas de sucesso em comparação com quando não são usadas:

Quadro 1: Práticas das Engenharias

Prática de engenharia	Aumento do sucesso ao usar versus não usar a prática (%)
Os requisitos do projeto eram claros antes do início do processo de desenvolvimento de software	97%
Ser capaz de discutir e resolver problemas rapidamente (Ou seja, "segurança psicológica")	87%
Os requisitos do projeto foram baseados com precisão no problema do mundo real	54%
O projeto tinha um documento completo de especificação ou requisitos antes do início do desenvolvimento	50%
Nenhuma alteração significativa foi feita nos requisitos no final do processo de desenvolvimento	7%

Fonte: USEHAYSTACK. 83% of developers suffer from burnout, Haystack Analytics study finds. Haystack Blog, 10 ago. 2021. Disponível em: https://www.statista.com/statistics/318818/numbers-of-programmers-and-software-development-professionals-in-the-uk/ .Acessado em 18/08/2025

Pode-se verificar no quadro 2, que há altas taxas de insucesso por metodologia ágil conforme o estudo demostrado, principalmente no item engenharia de requisitos ágeis com 65%, pois infelizmente não houve um foco nesse item.

Quadro 2: Taxas de Insucesso

Metodologia de Engenharia	Taxa de falha (%)	Aumento de falhas ao usar vs não usar metodologia (%)
Engenharia de Requisitos Ágeis (O desenvolvimento começa antes de requisitos claros, sem especificação completa, mudanças significativas no final do desenvolvimento.)	65%	268%
Desenvolvimento de software enxuto (trabalhando apenas em um projeto de cada vez.)	21%	7%
Engenharia de Impacto (Uso de todas as práticas de engenharia estudadas que aumentam as taxas de sucesso.)	10%	-56%

Fonte: HAYSTACK. 83% of developers suffer from burnout, Haystack Analytics study finds. Haystack Blog, 10 ago. 2021. Fonte: USEHAYSTACK. 83% of developers suffer from burnout, Haystack Analytics study finds. Haystack Blog, 10 ago. 2021. Disponível em: https://www.statista.com/statistics/318818/numbers-of-programmers-and-software-development-professionals-in-the-uk/ .Acessado em 18/08/2025

Os dados do quadro 3 sobre as práticas usadas por prevalência refletem diferentes aspectos da gestão de projetos de desenvolvimento de software, especificamente sobre a eficiência da comunicação, a clareza dos requisitos e a flexibilidade do processo. Essas práticas são muito usadas por prevalência no Reino Unido e nos EUA.

Um estudo desenvolvido por Oguz e Oguz (2019), analisa a lacuna entre a academia e a indústria de software. O artigo apresenta as razões para as diferenças, que incluem currículos desatualizados e a falta de integração prática na formação dos estudantes.

Quadro 3: Práticas usadas por prevalência

Prática	**Prevalência no Reino Unido (%)**	**Prevalência nos EUA (%)**
Ser capaz de discutir e resolver problemas rapidamente	79%	90%
O engenheiro de software relatou não ter que trabalhar em mais de um projeto ao mesmo tempo	61%	71%
Mudanças significativas foram feitas nos requisitos no final do processo de desenvolvimento	65%	59%
O projeto tinha um documento completo de especificação ou requisitos antes do início do desenvolvimento	88%	83%
Os requisitos do projeto eram claros antes do início do processo de desenvolvimento de software	92%	88%
Os requisitos do projeto foram baseados com precisão no problema do mundo real	83%	86%

Fonte: HAYSTACK. 83% of developers suffer from burnout, Haystack Analytics study finds. Haystack Blog, 10 ago. 2021. Disponível em: Fonte: USEHAYSTACK. 83% of developers suffer from burnout, Haystack Analytics study finds. Haystack Blog, 10 ago. 2021. Disponível em: https://www.statista.com/statistics/318818/numbers-of-programmers-and-software-development-professionals-in-the-uk/ .Acessado em 18/08/2025

O estudo sobre a lacuna entre academia e empresas no ensino de Engenharia de Software aborda questões fundamentais para compreender os desafios enfrentados pelas universidades na preparação de profissionais alinhados às

demandas do mercado de trabalho. O artigo do Oguz e Oguz (2019) trata os seguintes itens:

- **Identificação de Entidades.** O qual define os principais atores envolvidos, como estudantes, instituições acadêmicas e a indústria, além de suas relações e contribuições.

- **Habilidades Necessárias.** Destacando a importância de desenvolver habilidades técnicas *(hard skills)* e comportamentais *(soft skills),* incluindo comunicação e trabalho em equipe.

- **Metodologia.** Baseia-se em questionários aplicados a estudantes e recém-formados, além de análises de anúncios de emprego para identificar as competências demandadas.

- **Propostas de Solução.** Recomenda que todas as partes envolvidas (academia, estudantes e indústria) estejam atualizadas com as mudanças tecnológicas e se adaptem para minimizar essa lacuna.

- **Estudos e Pesquisas Anteriores**. Revisão de estudos que tenham investigado o alinhamento (ou desalinhamento) entre o ensino de Engenharia de Software e as necessidades do mercado de trabalho.

Para Oguz e Oguz (2019), o estudo ainda fala sobre a distância entre a indústria e a academia, por não existir uma fonte única do problema, que a profissão de Engenharia de Software é capaz de reagir rapidamente a novas plataformas e tendências que exigem a aquisição de novas habilidades, fato que não ocorre na academia.

A academia não é rápida o suficiente para incorporar as mudanças na profissão em seu currículo, tornando-se um desafio criar experiências realistas no ensino de engenharia por falta de experiências realistas, torna-se mais desafiador, ainda, para os estudantes adquirirem habilidades sociais necessárias para a colaboração no desenvolvimento de projetos de software em grande escala.

Como o mercado de tecnologia está em constante evolução e as habilidades exigidas para os profissionais de software também acompanham esse ritmo, a pesquisa realizada pelos autores, além de ter o objetivo de entender melhor as competências e conhecimentos que as empresas de software valorizam, também, com base nesses dados, apresentaram métodos práticos que podem ajudar tanto estudantes quanto engenheiros de software a preencher a lacuna entre a formação acadêmica e as exigências do mercado de trabalho.

O intuito é oferecer *insights* valiosos que podem contribuir para a literatura sobre a afirmação da distância persistente, como:

- analisar de várias perspectivas diferentes que envolvem todas as entidades relacionadas,
- usar questionários e realizar entrevistas com estudantes, recém-formados e acadêmicos para obter comentários e opiniões sobre o problema,
- compilar uma lista de habilidades que são procuradas pelas empresas de software, composta por uma inspeção cuidadosa de cinquenta anúncios de cargos de Engenharia de Software,
- apresentar conclusões e discutir métodos propostos que ajudarão estudantes e engenheiros de software a preencher a lacuna entre as partes.

Um aspecto relevante no ensino dos cursos de Engenharia de Software é que eles são conectados e construídos uns sobre os outros, mas os estudantes não conseguem ver sua conexão. Também é comum que os acadêmicos se concentrem nos tópicos relacionados ao curso e negligenciem outros tópicos.

Para Craig (2018), os estudantes se tornam proficientes em cada tópico, mas nunca desenvolvem aplicativos completos em que todos os tópicos são usados ao mesmo tempo. Se o curso for de estruturas de dados, a avaliação do projeto pode ignorar a interface gráfica do usuário, porém se for sobre interação humano-computador, a avaliação pode ignorar a representação de dados ou os princípios orientados a objetos. Os estudantes se tornam-se engenheiros de software e após a formatura e são os que mais sentem a distância entre a academia e as empresas.

Nesse mesmo estudo de Oguz e Oguz (2019), os levantamentos dos possíveis equívocos na Área da Engenharia de Software e seus níveis como descreve o quadro 4.

Sobre os níveis do artigo do Ogus Oguz (2019) refere-se:

Quadro 4: Nível Predominante

Área analisada	Nível predominante	Respostas	Interpretação
Análise de requisitos	2 – Pouco desafiador	21	A maioria percebe desafios iniciais, mas não considera extremamente complexa.
Design de Software	4 – Muito desafiador	21	Etapa considerada crítica, com alto nível de dificuldade técnica.
Desenvolvimento de Software	2 – Pouco desafiador	26	Atividade vista como prática e relativamente dominada pelos respondentes.
Testes de Software	2 – Pouco desafiador	18	Dificuldade moderada; distribuição equilibrada entre os níveis.
Gerência de Projetos	3 – Moderadamente desafiador	18	Exige atenção e esforço; parte dos respondentes indicou não ter experiência.
Gestão do Tempo Pessoal	2 – Pouco desafiador	21	Reconhecida como um desafio de organização, mas não extremo.
Comunicação com membros da equipe	1 – Nada desafiador	30	Habilidade bem consolidada, vista como de fácil manejo.
Uso de ferramentas de projeto (ex.: Git)	1 – Nada desafiador	20	Competência já incorporada à prática dos participantes.

Fonte: Oguz e Oguz (2019)

De outro ponto de vista, o maior desafio foram os testes de software, com 12 respostas. Entretanto, o menor desafio foi a comunicação com os membros da equipe, com 30 respostas.

Os resultados acabam sendo interessantes, pois a comunicação está surpreendentemente em um nível de desafio baixo, enquanto o design de software é tão desafiador quanto se espera que seja na área de Engenharia de Software.

O artigo de Oguz e Oguz (2019) destaca que, do ponto de vista das empresas, os recém-formados frequentemente não possuem competências suficientes para lidar de maneira eficaz com projetos do mundo real. Eles carecem de familiaridade com as ferramentas mais atuais, apresentam habilidades sociais limitadas e enfrentam dificuldades nos estágios iniciais de suas carreiras. Por essa razão, o mercado tende a priorizar estudantes com alto desempenho acadêmico, que podem ser treinados mais rapidamente, ou aqueles que já possuem experiência prática, adquirida por meio de participação em projetos comerciais ou de código aberto.

Sob a ótica acadêmica, diversos desafios também se apresentam, sendo o mais evidente o equilíbrio entre currículos que promovam tanto o desenvolvimento técnico quanto o aprimoramento das habilidades interpessoais dos estudantes. A academia tem reconhecido essa lacuna desde o início e tem buscado estratégias para reduzi-la ou eliminá-la. Apesar de alguns avanços, o rápido progresso das tecnologias e práticas profissionais tem mantido essa diferença persistente ao longo das últimas três décadas.

A indústria, consciente da lacuna existente, precisa intensificar sua colaboração com a academia, compartilhando informações sobre as metodologias e tecnologias mais recentes. As tendências de mercado desempenham um papel crucial na definição das tecnologias que ganham destaque, e a indústria pode beneficiar a si mesma e à academia ao adotar uma postura mais proativa nesse contexto.

Por sua vez, a academia pode estreitar os laços com o setor produtivo por meio da colaboração em projetos conjuntos, proporcionando aos estudantes oportunidades de vivenciar experiências do mundo real. Essa parceria ajudará a equilibrar a delicada relação entre os fundamentos teóricos e os aspectos práticos da profissão, promovendo um desenvolvimento mais alinhado às demandas do mercado.

5 PROPOSTAS DE MELHORIA PARA O ENSINO DE ENGENHARIA DE SOFTWARE

O plano de ensino da disciplina de Engenharia de Software desempenha um papel fundamental na formação de profissionais qualificados para atender às demandas do mercado de tecnologia da informação, cada vez mais dinâmico e competitivo. Contudo, é evidente que desafios persistem no alinhamento entre os conteúdos acadêmicos e as necessidades práticas do setor.

Diversos fatores, como a rápida evolução das tecnologias, a adoção de metodologias ágeis e o foco crescente em habilidades comportamentais exigem a constante revisão das abordagens pedagógicas e currículos. A busca é promover um aprendizado mais interativo, relevante e adaptado às exigências do mercado, contribuindo para a formação de profissionais mais preparados e competentes.

5.1 Integração de Metodologias Tradicionais e Ágeis no Ensino

A evolução das metodologias de desenvolvimento de software ao longo das últimas décadas tem exigido uma reavaliação constante das práticas pedagógicas adotadas no ensino superior. Com o advento das metodologias ágeis, como *SCRUM, XP, Kanban e Lean* Software *Development*, tornou-se evidente a necessidade de promover a flexibilidade, a colaboração e a entrega contínua de valor, aspectos essenciais no atual cenário de desenvolvimento dinâmico e centrado no usuário. No contexto educacional, isso implica integrar tais abordagens ao currículo, de modo que os estudantes desenvolvam competências técnicas e comportamentais em ambientes simulados que reflitam o cotidiano das equipes ágeis reais.

A ideia é criar uma abordagem mista**,** explorando como as metodologias tradicionais e ágeis podem ser combinadas ou integradas no ensino de Engenharia de Software.

5.1.1 Evolução da Engenharia de Software: a proposta de um modelo híbrido

Sommerville e Pressman (2020) discutem as tendências emergentes na Engenharia de Software, incluindo novas tecnologias, práticas e metodologias que

estão moldando o futuro da profissão. Esse estudo buscar trazer uma nova abordagem na metodologia de ensino no desenvolvimento do ciclo de vida do software por meio do processo híbrido para ser utilizado nas universidades e ser incorporado ao mercado de trabalho por meio dos dados nos próximos capítulos.

Os modelos tradicionais observam que, embora esses modelos proporcionem uma abordagem estruturada, podem ser inflexíveis para lidar com mudanças nos requisitos. Já os modelos ágeis enfatizam que eles oferecem maior flexibilidade e capacidade de resposta a mudanças, o que é particularmente útil em ambientes dinâmicos e projetos onde os requisitos podem evoluir rapidamente.

Sommerville (2020) explora técnicas de estimativa, gerenciamento de riscos e controle de mudanças. Com essa atividade, o ciclo de Vida do Software por meio da fase de levantamento de requisitos em que a engenharia é fundamental para o sucesso de um projeto de software. A fase explora técnicas para elicitação, análise e documentação de requisitos e discute como lidar com requisitos conflitantes e mudanças.

Combinar modelos de processos de software tradicionais e ágeis pode ser uma abordagem eficaz para aproveitar o melhor dos dois mundos. Pode-se chamar essas estratégias para integrar práticas tradicionais e ágeis em um modelo híbrido

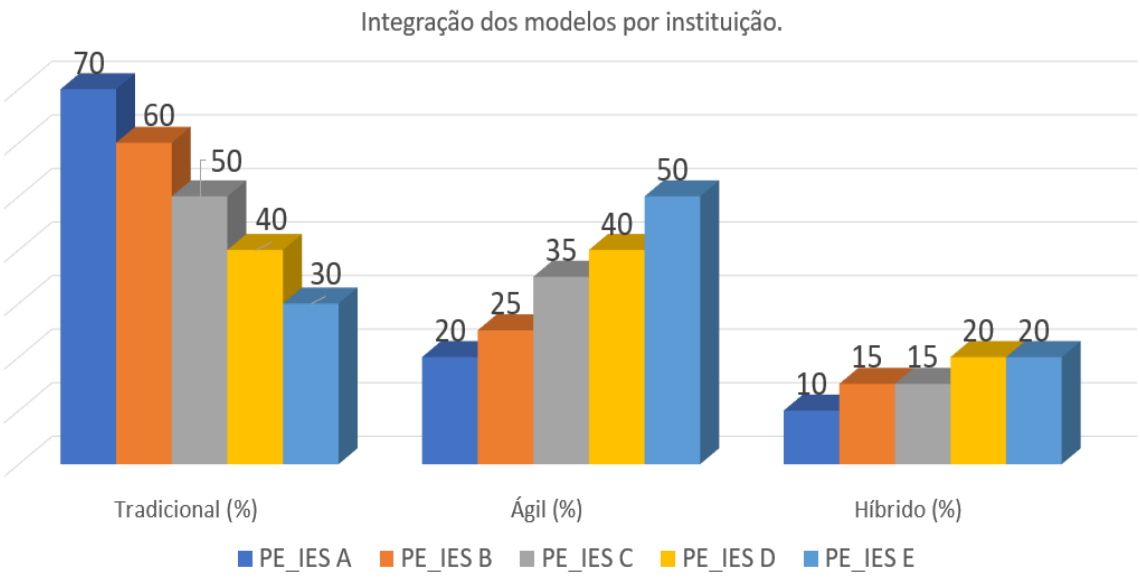
5.1.2 Modelo Híbrido

A escolha entre modelos tracionais e ágeis depende muito das características específicas do projeto, da equipe e da organização. Portanto, não há um modelo universal ideal que se encaixe perfeitamente em todos os projetos.

O modelo híbrido, por sua vez, combina práticas de ambos os tipos de abordagem, permitindo flexibilidade e controle contínuo do fluxo de trabalho. Esse modelo é ideal para organizações que precisam personalizar seus processos para atender a diferentes tipos de projetos e requisitos. Ele também pode ser vantajoso para equipes que desejam a estrutura e a previsibilidade dos modelos tradicionais, que ao mesmo tempo, precisam da flexibilidade e adaptabilidade dos métodos ágeis.

O gráfico 1, demonstra como apresenta os modelos de processos nos planos de ensino das instituições de ensino superior de Sorocaba, em termos de percentual e por isso a ideia se dispor de um modelo híbrido, para buscar nivelar o uso dos modelos de processos.

Gráfico 1: Integração dos modelos por PE_IES



Fonte: Elaboração própria

No entanto, a adoção de um modelo híbrido exige cuidados especiais, pois pode ser necessário ajustar e adaptar constantemente as práticas de ambos os modelos para encontrar o equilíbrio ideal. Implementá-lo de maneira eficaz pode ser complexo e requer um entendimento profundo das necessidades do projeto e das capacidades da equipe. Por isso, é importante analisar cuidadosamente qual combinação de práticas irá garantir o melhor desempenho e sucesso para o projeto em questão. Conforme alguns dos autores citados como Sommerville, Pressman, Royce, para a escolha do modelo mais adequado é necessário considerar:

- **Natureza do Projeto.** Requisitos bem definidos e estáveis podem se beneficiar de modelos tradicionais, enquanto projetos com alta incerteza e necessidade de adaptação são mais adequados para modelos ágeis.

- **Tamanho e Complexidade da Equipe.** Modelos ágeis geralmente funcionam melhor com equipes menores e mais colaborativas.

- **Expectativas de Mudança.** Projetos com alta probabilidade de mudanças nos requisitos tendem a se adaptar melhor a práticas ágeis.

- **Cultura da Organização.** A disposição da equipe e da organização para adotar práticas ágeis ou tradicionais pode influenciar a escolha do modelo.

Combinar práticas de diferentes modelos também pode ser uma solução eficaz para adaptar o processo às necessidades específicas do projeto e da equipe. Umas das posposta desse modelo é inserir nos estudos e nas empresas:

- **Documentação rigorosa.** Garantir que todas as etapas do projeto sejam bem documentadas para clareza e referência futura.

- **Engenharia de requisitos.** Definir claramente os requisitos desde o início para evitar mudanças frequentes e manter o foco, principalmente o requisito de negócio.

- **Segurança psicológica.** Criar um ambiente onde os desenvolvedores possam discutir problemas sem medo de represálias, promovendo a resolução rápida de questões.

- **Monitoramento contínuo.** Usar ferramentas avançadas para monitorar o progresso do projeto e fazer ajustes necessários em tempo hábil.

I. Benefícios do Modelo

- **Redução nas taxas de falha.** A metodologia demonstrou uma redução significativa nas taxas de falha dos projetos.

- **Entrega pontual.** Com requisitos bem definidos e documentados, os projetos têm mais chances de serem entregues no prazo.

- **Qualidade elevada.** A documentação e o monitoramento contínuos garantem a qualidade do software desenvolvido.

- **Prevenção do esgotamento.** Um ambiente de trabalho seguro e colaborativo ajuda a prevenir o esgotamento dos desenvolvedores.

II. Indefinição e fases do Modelo Híbrido

- **Fases Tradicionais**. Usar fases tradicionais para planejamento e design em alto nível, garantindo uma base sólida.

- **Iterações Ágeis**. Realizar iterações ágeis para desenvolvimento e ajustes contínuos, permitindo a adaptação às mudanças e *feedbacks*.

III. Planejamento e Gestão de Projetos

- **Planejamento Inicial**. Usar técnicas tradicionais para um planejamento detalhado no início do projeto.

- **Gestão Ágil**. Empregar práticas ágeis para gestão diária e adaptação, com reuniões diárias, revisões de *sprint* e retrospectivas.

IV. Documentação

- **Documentação Inicial**. Preparar documentação detalhada para requisitos e design, conforme práticas tradicionais.

- **Documentação Contínua**. Utilizar documentação leve e atualizada continuamente para refletir mudanças ágeis durante o desenvolvimento.

V. Controle de Qualidade

- **Testes Tradicionais**. Implementar testes formais em fases críticas para garantir conformidade com requisitos.

- **Testes Ágeis**. Realizar testes contínuos e *feedback* rápido durante o ciclo de desenvolvimento para detectar e corrigir problemas rapidamente.

VI. Gestão de Risco

- **Análise de Risco Inicial.** Realizar uma análise abrangente de riscos no início do projeto.

- **Gerenciamento de Risco Ágil**. Monitorar e ajuste a abordagem de gerenciamento de risco conforme o projeto avança e novos riscos emergem.

VII. Integração e Entregas

- **Integração Contínua**. Combinar práticas de integração contínua com entregas incrementais para assegurar que o produto seja testado e validado frequentemente.

- **Entregas Planejadas**. Planejar entregas maiores e marcos importantes seguindo práticas tradicionais, com entregas menores e mais frequentes ao longo do desenvolvimento.

VIII. Comunicação e Colaboração

- **Estruturas Formais.** Utilizar estruturas formais para comunicação e aprovação em momentos chave.

- **Colaboração Ágil**. Promover uma comunicação aberta e colaborativa entre as equipes, facilitada por reuniões rápidas e colaboração contínua.

IX. Ferramentas e Tecnologia

- **Ferramentas Tradicionais.** Utilizar ferramentas tradicionais para planejamento, rastreamento e gestão de projetos.

- **Ferramentas Ágeis**. Empregar ferramentas ágeis para suporte à comunicação e colaboração em tempo real, como quadros *Kanban* e sistemas de rastreamento de tarefas.

X. Cultura e Treinamento

- **Treinamento Formal**. Oferecer treinamento formal em práticas tradicionais e ágeis para a equipe.

- **Adaptação Cultural**. Fomentar uma cultura de flexibilidade e abertura à mudança, promovendo uma mentalidade que valorize tanto a previsibilidade quanto a adaptabilidade.

Integrar práticas tradicionais e ágeis permite que se mantenha a estrutura e previsibilidade necessárias para grandes projetos enquanto aproveita a flexibilidade e a capacidade de resposta ágil para mudanças e melhorias contínuas. O sucesso dessa abordagem híbrida depende de uma aplicação cuidadosa e adaptada às necessidades específicas do projeto e da organização.

5.1.3 O porquê do modelo híbrido

Com as informações já analisadas neste estudo e com a preocupação das empresas em entregar um Software com qualidade, não dá para seguir com um único modelo de processo de software. Os dados a seguir mostram a preocupação de alguns países em seguir somente com o modelo ágil, visto que é o mais popular hoje em dia.

No período de 3 a 7 de maio 2024, a J.L. Partners solicitou respostas de 481 engenheiros de software que encontraram um projeto bem-sucedido pela última vez e 119 que encontraram um projeto fracassado pela última vez, para permitir uma análise suficiente de ambos os grupos conforme citado no quadro 2.

O estudo enfatiza a necessidade de processos bem documentados e a importância de um ambiente seguro para discussões de problemas, além de medidas para evitar o esgotamento dos desenvolvedores. A execução de projetos de software é um campo em constante evolução, com metodologias ágeis se tornando uma abordagem popular devido à sua flexibilidade e foco em entregas incrementais. No entanto, como revelado pelo estudo mencionado, essa abordagem também tem suas desvantagens significativas.

As metodologias ágeis, concentram-se na entrega Incremental de software, permitindo ajustes rápidos com base no *feedback* contínuo do cliente. Esse processo é interativo e colaborativo, com equipes auto-organizadas que trabalham nas entregas (*sprints*) com entregas em tempo menores e funcionalidades prontas para uso. Alguns dos equívocos analisados do processo são:

- **Falta de documentação completa.** A ênfase em software funcionando pode levar à documentação inadequada, dificultando a manutenção e o entendimento do projeto a longo prazo.

- **Mudanças constantes nos requisitos.** A flexibilidade para mudanças pode resultar em escopo indefinido e constante alteração nos objetivos, causando atrasos.

- **Esgotamento dos desenvolvedores.** O ritmo intenso e a pressão para entregas rápidas podem levar ao esgotamento dos membros da equipe.

5.2 Alinhamento entre Ensino e Mercado de Trabalho

As competências exigidas pelo mercado de trabalho para profissionais de Engenharia de Software buscam a formação de profissionais que deve estar alinhada rapidamente em função das inovações tecnológicas. A seguir no quadro 5, são apresentadas as competências importantes para as empresas.

Quadro 5: Alinhamento entre formação acadêmica e mercado

Técnicas	Interpessoais	Analíticas	Aprendizado Contínuo
Programação, Testes, DevOps	Comunicação, Trabalho em equipe, Resolução de conflitos	Pensamento crítico, Análise de dados, Requisitos	Adaptabilidade, Iniciativa, Mentalidade de crescimento

Fonte: Elaboração própria

5.2.1 Competências Técnicas

As competências técnicas em Engenharia de Software referem-se ao conjunto de habilidades e conhecimentos que permitem aos profissionais projetarem, desenvolverem, testarem e manterem sistemas de software de forma eficaz e eficiente.

Com base na definição de competências técnicas em Engenharia de Software apresentada por Sommerville (2011), que destaca a importância de habilidades específicas para a execução de tarefas complexas no desenvolvimento de software, podem ser exploradas algumas das principais competências técnicas essenciais para profissionais da área, como programação, metodologias ágeis, testes de software e práticas de *DevOps*, que são fundamentais não apenas para o sucesso no ambiente atual de desenvolvimento, mas também para acompanhar as rápidas mudanças tecnológicas e as demandas do mercado. Algumas mudanças a serem feitas são:

- **Programação**. O domínio de linguagens como *Java, Python e JavaScript* é fundamental. Segundo McKinsey (2021), "as habilidades de programação são essenciais para quase todas as funções em tecnologia, sendo uma das competências mais procuradas pelos empregadores."

- **Desenvolvimento Ágil**. A familiaridade com metodologias ágeis, como *SCRUM* e *Kanban,* é altamente valorizada. Como afirmado por Cohn (2005), "as equipes que utilizam práticas ágeis conseguem entregar valor mais rapidamente e responder melhor às mudanças nas necessidades dos clientes."

- **Testes de Software**. A capacidade de implementar práticas como *Test Driven Development* (TDD) é crucial. Segundo Beck (2003), TDD não só melhora a qualidade do código, mas também ajuda aos desenvolvedores a entenderem melhor os requisitos do software.

- **DevOps**. O conhecimento em integração e entrega contínua (CI/CD) é vital. De acordo com Jez Humble e David Farley (2010), *DevOps* transforma a cultura organizacional, permitindo uma colaboração mais eficaz entre desenvolvimento e operações.

5.2.2 Competências Interpessoais

As competências interpessoais, também conhecidas como *soft skills*, são habilidades essenciais que permitem a interação eficaz e colaborativa entre profissionais em ambientes de trabalho. Na Engenharia de Software, essas competências são tão importantes quanto às habilidades técnicas, pois a maioria dos projetos de software envolve trabalho em equipe, comunicação com partes interessadas e gestão de conflitos. Como afirmam Pressman e Maxim (2014), o sucesso em projetos de software não depende apenas do domínio técnico, mas também da habilidade de trabalhar bem com os outros e gerenciar relacionamentos de forma eficaz.

Considerando a importância das competências interpessoais destacadas por Pressman e Maxim (2014), que ressaltam a necessidade de habilidades para interagir e colaborar de forma eficaz em projetos de software, é possível observar como competências específicas, como trabalho em equipe, comunicação e resolução de problemas, desempenham um papel fundamental na garantia do sucesso dos projetos. Como afirmam Katzenbach e Smith (1993), Berkun (2010) e Polya (1957), essas habilidades são essenciais para criar soluções criativas, garantir a fluidez no compartilhamento de ideias e lidar com desafios complexos de forma colaborativa. Além disso, o domínio dessas *soft skills* contribui para um ambiente de trabalho harmonioso, onde a confiança e a cooperação entre os membros da equipe são fortalecidas, o que é crucial para a entrega de soluções de software de alta qualidade.

- **Trabalho em Equipe**. A habilidade de colaborar em grupos multidisciplinares é essencial. Segundo Katzenbach e Smith (1993), equipes de alto desempenho são aquelas que combinam diferentes habilidades e experiências, resultando em soluções mais criativas.

- **Comunicação**. A comunicação eficaz é fundamental para a colaboração. Como ressaltado por Berkun (2010), grandes ideias não valem nada se não puderem ser comunicadas claramente aos outros.

- **Resolução de Problemas**. A habilidade em identificar e resolver problemas complexos é necessária. Segundo Polya (1957), resolver problemas é uma

habilidade que pode ser ensinada e aprendida e, é essencial em qualquer campo, especialmente na engenharia.

5.2.3 Competências Analíticas

As competências analíticas são habilidades cruciais que permitem aos profissionais de Engenharia de Software avaliar problemas complexos, interpretar dados e tomar decisões informadas. Essas competências são fundamentais para garantir a qualidade e a eficácia das soluções de software desenvolvidas, uma vez que envolvem a capacidade de decompor problemas em partes menores e analisar cada componente de forma crítica. Segundo Boehm (1988), a análise eficaz é a base para a tomada de decisões acertadas e a resolução de problemas em projetos de software.

Considerando a importância das competências analíticas destacadas por Boehm (1988), que são essenciais para a avaliação e a resolução de problemas complexos no desenvolvimento de software, podem ser identificado habilidades específicas, como pensamento crítico, análise de dados, análise de requisitos, raciocínio lógico e avaliação de dados, que são fundamentais para garantir a precisão e a qualidade das decisões e soluções adotadas ao longo do processo de desenvolvimento. As empresas que utilizam dados para tomar decisões superam significativamente aquelas que não o fazem, o que evidencia a relevância da análise crítica e da interpretação de dados em todas as fases do ciclo de vida do software. Podem-se observar mais detalhes sobre as habilidades citadas pelo Boehm:

- **Pensamento Crítico**. A capacidade de avaliar informações de forma crítica é crucial. De acordo com Facione (2011), o pensamento crítico é a habilidade de pensar de maneira clara e racional, compreendendo a conexão entre ideias.

- **Análise de Dados**. Com o crescimento da big data, a análise de dados se torna cada vez mais importante. Segundo Davenport e Harris (2007), as empresas que utilizam dados para tomar decisões superam significativamente aquelas que não o fazem.

- **Análise de Requisitos**. Habilidade para coletar, entender e documentar as necessidades dos usuários, garantindo que os requisitos sejam claros e completos.

- **Raciocínio Lógico**. Capacidade de aplicar princípios lógicos para resolver problemas e desenvolver algoritmos, o que é essencial na programação e na criação de soluções eficientes.

- **Avaliação de Dados**. Aptidão para interpretar dados quantitativos e qualitativos, utilizando ferramentas e técnicas de análise para apoiar decisões de design e implementação.

5.2.4 Competências de Aprendizado Contínuo

As competências de aprendizado contínuo referem-se à capacidade dos profissionais de Engenharia de Software de se adaptarem e se atualizarem em um ambiente tecnológico em constante mudança. Essa habilidade é fundamental, uma vez que novas tecnologias, metodologias e ferramentas emergem regularmente, e a capacidade de aprender de forma autônoma e contínua é crucial para a eficácia e a relevância no campo. Como destaca Smith (2012), no contexto da Engenharia de Software, o aprendizado contínuo é não apenas uma vantagem competitiva, mas uma necessidade para a sobrevivência profissional.

Considerando a importância do aprendizado contínuo destacada por Smith (2012), que enfatiza a necessidade de adaptação e atualização constantes no campo da Engenharia de Software, é possível observar competências como adaptabilidade e iniciativa, mencionadas por Senge (1990), são fundamentais para que os profissionais se mantenham relevantes e eficazes frente às rápidas mudanças tecnológicas e exigências do mercado. Além disso, o alinhamento entre o aprendizado contínuo e o desenvolvimento dessas habilidades não só garante a evolução profissional, mas também facilita a integração do indivíduo nas organizações, contribuindo para o sucesso coletivo no enfrentamento de novos desafios.

- **Adaptabilidade**. A capacidade de se adaptar rapidamente a novas tecnologias é essencial. Como destacado por Senge (1990), as organizações que

aprendem são aquelas que se adaptam rapidamente às mudanças no ambiente.

- **Iniciativa**. A proatividade em buscar aprendizado é valorizada. De acordo com Dweck (2006), uma mentalidade de crescimento leva as pessoas a se esforçarem mais e a se dedicarem ao desenvolvimento contínuo.

As competências exigidas pelo mercado para profissionais de Engenharia de Software incluem um conjunto abrangente de habilidades técnicas, interpessoais, analíticas e de aprendizado contínuo. O alinhamento entre a formação acadêmica e as expectativas do mercado é essencial para garantir que os graduados estejam prontos para os desafios das organizações.

5.3 Desafios na Formação Acadêmica

Para o MEC (2016), os desafios enfrentados pelas instituições de ensino na tentativa de alinhar o currículo às demandas do mercado acontecem, pois, as universidades particulares têm um papel importante na formação de engenheiros de software, refletindo as necessidades do mercado e as tendências educacionais. O ensino de Engenharia de Software nessas instituições geralmente é marcado pela flexibilidade curricular, que permite a rápida adaptação às novas demandas do setor.

Alguns desafios nas Academias estudantis:

- **Proporcionar currículos flexíveis e adaptáveis.** Pois geram a capacidade de ajustar rapidamente seus currículos, integrando novas tecnologias e metodologias. Isso permite que os cursos de Engenharia de Software incluam não apenas modelos tradicionais, mas também práticas contemporâneas, como metodologias ágeis.

- **Criar ênfase em práticas e projetos.** Os estudantes são incentivados a trabalhar em projetos reais ou simulados. Isso proporciona uma experiência valiosa que complementa a teoria, essencial para a formação de profissionais qualificados.

- **Desenvolver parcerias com a indústria.** Permite a inclusão de estágios e projetos colaborativos. Essa interação com o mercado fornece aos estudantes uma visão prática das necessidades e desafios enfrentados pelas empresas.

  Fazer análise crítica e contextualização. Considerando suas vantagens e desvantagens em diferentes contextos. Isso os ajuda a desenvolver habilidades de tomada de decisão e adaptação, fundamentais no mercado de trabalho.

- **Uso de Ferramentas e Tecnologias Modernas.** As universidades particulares frequentemente adotam ferramentas de software modernas que suportam os modelos tradicionais. Isso ajuda os estudantes a aplicarem teorias em um ambiente de trabalho realista, preparando-os para as tecnologias que encontrarão na indústria.

5.4 Propostas de Melhoria para o Plano de Ensino da disciplina de Engenharia de Software

O ensino de Engenharia de Software enfrenta desafios relacionados à necessidade de preparar profissionais para um mercado de trabalho dinâmico e tecnologicamente avançado. Nesse contexto, a implementação de inovações pedagógicas torna-se essencial para garantir uma formação que alinhe competências técnicas, práticas e sociais.

Uma abordagem destacada é a aprendizagem baseada em projetos, que incentiva os estudantes a trabalharem em problemas reais ou simulados, promovendo a aplicação prática dos conceitos teóricos.

Segundo Kolmos et al. (2020), esse método não apenas facilita a retenção do conhecimento, mas também desenvolve habilidades colaborativas, de resolução de problemas e gestão de projetos, essenciais na área de Engenharia de Software.

Outro modelo promissor é o ensino híbrido, que combina aulas presenciais e recursos digitais. Conforme descrito por Graham (2019), essa abordagem permite flexibilidade no aprendizado, oferecendo ao aluno a possibilidade de acessar conteúdos teóricos online e utilizar os encontros presenciais para discussões práticas e atividades em grupo. Essa metodologia é especialmente relevante em disciplinas como Engenharia de Software, que requerem um equilíbrio entre teoria e prática.

Além disso, a criação de laboratórios ágeis no ambiente acadêmico tem se mostrado uma solução eficaz. Esses laboratórios simulam o funcionamento de equipes ágeis utilizadas no mercado, como *SCRUM* e *Kanban*, proporcionando aos estudantes uma experiência imersiva.

De acordo com Beck e Van Rensburg (2021), a utilização de laboratórios ágeis fomenta a adaptação a cenários reais, promovendo a experimentação, o *feedback* contínuo e a melhoria Incremental, características fundamentais no desenvolvimento de software.

Portanto, a combinação de métodos inovadores pode transformar o estudo da Engenharia de Software, preparando os estudantes para enfrentar os desafios da profissão de forma mais eficiente e adaptável. Essas propostas não apenas aumentam o engajamento dos estudantes, mas também alinham a formação acadêmica às necessidades reais do mercado de tecnologia.

Salienta-se, também, a Integração de experiências práticas, com análise da importância de projetos práticos, estágios, e parcerias com empresas na formação dos estudantes em que é fundamental acompanhar as constantes mudanças no mercado tecnológico, garantindo que os estudantes adquiram tanto conhecimentos teóricos quanto habilidades práticas.

De acordo com Sommerville (2020), esses projetos permitem aos estudantes aplicarem conceitos teóricos em situações reais ou simuladas, desenvolvendo competências como análise de requisitos, design de sistemas e colaboração em equipe. Além disso, projetos interdisciplinares que envolvam outras áreas da computação ajudam a preparar os estudantes para os desafios da indústria, que frequentemente exige a integração de diferentes especialidades.

Os estágios supervisionados desempenham um papel igualmente importante na formação profissional. Segundo Lethbridge et al. (2019), a experiência em ambientes corporativos permite que os estudantes entendam o ciclo de vida do software em contextos reais, além de oferecer a oportunidade de vivenciar metodologias ágeis, ferramentas de gestão e práticas colaborativas. Estágios também contribuem para o desenvolvimento de *soft skills*, como comunicação e resolução de problemas, habilidades cruciais em equipes de desenvolvimento.

Segundo Pressman e Maxim (2020), a colaboração com empresas permite que os estudantes tenham acesso a desafios reais, ferramentas avançadas e mentoria de profissionais experientes. Além disso, programas como *hackathons,* oficinas e

projetos coorientados estimulam a inovação e ajudam os estudantes a desenvolverem uma visão prática e estratégica. As parcerias entre universidades e empresas têm se mostrado uma estratégia eficaz para aproximar os estudantes das demandas do mercado.

A integração de experiências práticas no ensino da disciplina de Engenharia de Software é essencial para formar profissionais mais completos e preparados para os desafios do mercado de trabalho. Projetos práticos, estágios e parcerias com empresas oferecem aos estudantes a oportunidade de alinhar teoria e prática, consolidando o aprendizado e promovendo a empregabilidade.

As Universidades devem propor um currículo flexível e adaptativo e conforme o que o MEC pode permitir por meio de suas diretrizes, que essa criação de currículos possa se adaptar rapidamente às mudanças nas demandas do mercado, incorporando *feedback* contínuo de profissionais e empresas.

Esse modelo curricular deve garantir uma formação sólida e abrangente, mas também permite que o curso se ajuste rapidamente às mudanças nas necessidades da indústria, incorporando *feedback* contínuo de profissionais e empresas. Um currículo flexível tem a capacidade de integrar as tendências tecnológicas emergentes e adaptá-las ao longo do tempo, sem comprometer os fundamentos da disciplina.

De acordo com Garcia et al. (2018), currículos adaptativos permitem a inclusão de novas tecnologias e metodologias de desenvolvimento de software, como inteligência artificial, *blockchain* e computação em nuvem, que rapidamente se tornam relevantes no mercado. Isso cria um ambiente de aprendizado onde os estudantes não apenas aprendem as práticas consolidadas, mas também são preparados para lidar com as inovações que surgem na área. Além disso, a criação de currículos flexíveis requer um mecanismo de *feedback* contínuo entre as universidades e o mercado de trabalho.

Segundo Shulman (2019), um processo de avaliação em tempo real, baseado em contribuições de empresas e profissionais da indústria, pode ajudar as instituições a ajustarem rapidamente o conteúdo curricular e os métodos pedagógicos. Isso pode ser feito por meio de consultorias acadêmicas, programas de estágio, parcerias com empresas de tecnologia e até mesmo por meio de plataformas de aprendizagem online que forneçam dados sobre as lacunas de habilidades e as novas exigências do mercado.

A incorporação de *feedback* contínuo também pode ocorrer por meio de fóruns, *hackathons* e projetos colaborativos com empresas, proporcionando uma interface direta entre estudantes e profissionais da área. Para Moravec (2021), essas interações permitem que o currículo se mantenha relevante, respondendo a necessidades emergentes de forma mais ágil e eficaz. Portanto um currículo flexível e adaptativo, que consiga se moldar conforme as transformações do mercado e incorpore o *feedback* constante de profissionais e empresas, é uma proposta essencial para a área de Engenharia de Software.

A melhoria do ensino da disciplina de Engenharia de Software requer a integração de abordagens inovadoras, experiências práticas e um currículo flexível. Esses elementos garantem que os estudantes desenvolvam habilidades técnicas e interpessoais, além de estarem alinhados às necessidades de um mercado em constante evolução. Com essas propostas, o ensino de Engenharia de Software se torna mais eficiente e adaptado às exigências do século XXI.

6 RESULTADOS E DISCUSSÃO

Esta pesquisa foi conduzida a partir de uma coleta de dados de cursos de Análise e Desenvolvimento de Sistemas em cinco instituições de ensino superior particulares da cidade de Sorocaba. Os Planos de Ensino dessas instituições estão denominados como PE_IES A, PE_IES B, PE_IES C, PE_IES D e PE_IES E.

A primeira parte do estudo envolveu a leitura e interpretação dos planos de ensino da disciplina de Engenharia de Software de cada IES, que foram repassados pelos coordenadores de cada curso/instituição.

6.1 Análise Qualitativa do Currículo da Disciplina Engenharia de Software

Nesse subcapítulo, será analisada uma visão geral das instituições apresentando uma descrição das instituições de ensino analisadas, incluindo informações sobre a estrutura dos cursos de Engenharia de Software, o currículo adotado, e as abordagens metodológicas utilizadas. Foram feitas análise pelo Plano de Ensino, e será apresentado no Anexo A, os planos das cinco instituições particulares de Sorocaba sem sua identificação.

6.1.1 PE_IES A

Esta análise buscou examinar, sob uma abordagem qualitativa, o plano de ensino da disciplina Engenharia de Software do curso de Análise e Desenvolvimento de Sistemas da instituição A.

O foco recai sobre a estrutura curricular, os conteúdos programáticos, os objetivos de aprendizagem e os métodos avaliativos, com o intuito de identificar a forma como os modelos tradicionais e modernos de desenvolvimento de software são tratados no âmbito formativo.

A análise, de natureza documental, reconhece limitações quanto à interpretação prática e real da implementação didático-pedagógica.

a) Potencialidades da Abordagem Qualitativa

O PE_IES A, apresenta uma estrutura clara e objetiva, organizada em torno de tópicos fundamentais da Engenharia de Software. Entre os conteúdos contemplados, destacam-se:

- Métodos de análise e projeto de software;
- Métricas de software e estimativas;
- Projeto e implementação;
- Testes de software;
- Qualidade e gerenciamento de configuração de software.

A proposta também explicita a intenção de apresentar o processo unificado, princípios de gerenciamento de projetos, arquiteturas de software e padrões de qualidade, o que sugere uma abordagem sólida do ponto de vista dos modelos clássicos de desenvolvimento.

A bibliografia básica é composta por autores consagrados na área Pressman, Sommerville, o que garante respaldo teórico às temáticas abordadas. Há ainda referência a obras específicas sobre *UML*, o que reforça o foco na modelagem orientada a objetos como prática recorrente no ensino.

Quanto às estratégias pedagógicas, são mencionadas aulas expositivas, resolução de exercícios e uso de laboratório com linguagem de programação instalada, sugerindo a tentativa de integração entre teoria e prática. A presença de avaliações objetivas e atividades extraclasse também indica uma preocupação com a verificação do aprendizado de forma contínua.

b) Limitações Inerentes à Abordagem Qualitativa

Apesar da estrutura coerente e da abordagem teórica sólida, a análise qualitativa desse plano de ensino revela algumas limitações importantes, principalmente relacionadas à ausência de conteúdos explícitos sobre metodologias ágeis, como *SCRUM, XP, Kanban* ou *Lean Software Development.* Esse dado sugere um enfoque predominantemente tradicional, com possível defasagem frente às práticas contemporâneas de desenvolvimento ágil adotadas amplamente no mercado.

Além disso, o plano não menciona o uso de ferramentas de apoio ao desenvolvimento, como versionamento (*Git*), prototipação, integração contínua, *DevOps* ou testes automatizados, o que enfraquece a conexão entre o conteúdo didático e a realidade prática vivenciada por profissionais da área.

Não há indicação de avaliação por projetos integradores ou aplicação em contextos reais, o que reduz o potencial de desenvolvimento de competências aplicadas, essenciais à formação profissional no campo da Engenharia de Software.

O PE_IES A revela um currículo coerente com os fundamentos clássicos da Engenharia de Software, com foco em modelagem, métricas, qualidade e processos estruturados.

A ausência de práticas ágeis, ferramentas modernas e metodologias ativas de ensino configura uma limitação significativa em relação às exigências do mercado atual.

6.1.2 PE_IES B

Esta seção apresenta uma análise qualitativa dos planos de ensino das disciplinas Engenharia de Software I e Engenharia de Software II, pertencentes ao curso de Tecnologia em Análise e Desenvolvimento de Sistemas.

A análise baseia-se na leitura interpretativa dos planos, com o objetivo de identificar como os modelos tradicionais e ágeis de desenvolvimento de software são integrados à formação discente, assim como suas abordagens didáticas e limitações curriculares.

A utilização da abordagem qualitativa permite uma leitura aprofundada dos elementos pedagógicos presentes nos documentos analisados. Ambos os planos demonstram um esforço notável de abrangência e atualização conceitual, incorporando tanto fundamentos clássicos quanto práticas modernas do desenvolvimento de software.

A disciplina Engenharia de Software I dedica-se à formação conceitual e processual, introduzindo modelos tradicionais como Waterfall, Incremental, *RAD*, Prototipação, *RUP, Cleanroom* e Processo Unificado, bem como uma seção robusta dedicada aos modelos ágeis como *SCRUM*, *XP*, *FDD*, *DSDM* e *Crystal*. Essa abrangência indica uma tentativa clara de aproximar o discente das diferentes abordagens do ciclo de vida de software.

Na Engenharia de Software II, há uma ênfase nos processos de verificação, validação, testes e manutenção de software, com referências a normas internacionais como *ISO/IEC 9126*, *ISO 25000*, *ISO 12207*, *CMMI* e MPS.Br. Isso demonstra um alinhamento com padrões de qualidade globalmente aceitos, além de promover uma formação voltada à qualidade e à sustentabilidade do software no tempo.

Outro ponto positivo é a inserção explícita de estratégias pedagógicas ativas, como seminários, debates, dinâmicas de grupo, estudo de casos e Projeto Integrado Multidisciplinar (PIM), o que favorece a aprendizagem significativa e o desenvolvimento de competências práticas.

a) Limitações Inerentes à Abordagem Qualitativa

A abordagem qualitativa também impõe limitações que precisam ser consideradas. A análise baseia-se exclusivamente no plano de ensino e, portanto, não permite inferir com segurança como os conteúdos são efetivamente aplicados na prática pedagógica. Não há, por exemplo, menção à utilização de ferramentas de desenvolvimento reais, à participação em projetos com clientes externos, ou ao uso de metodologias ágeis em situações reais.

Os planos mencionam modelos ágeis e tradicionais, porém não está claro se tais modelos são apenas abordados conceitualmente ou se fazem parte de atividades práticas. A ausência de informações sobre o uso de ambientes de versionamento integração contínua, metodologias híbridas ou ferramentas de automação de testes também limita a compreensão do grau de atualização técnica da formação oferecida.

Outro ponto a se destacar é que, apesar de os conteúdos serem distribuídos em duas disciplinas, as práticas *DevOps*, *mobile-first*, segurança no ciclo de vida e computação em nuvem ainda não aparecem de maneira explícita, o que indica um possível descompasso com tendências emergentes do setor.

A análise qualitativa do PE_IES B, revela um currículo equilibrado entre fundamentos teóricos e práticas modernas, com destaque para a inserção de modelos ágeis e padrões de qualidade. No entanto, a limitação da abordagem metodológica impede uma avaliação aprofundada da implementação real desse conteúdo.

6.1.3 PE_IES C

Este trecho do trabalho apresenta uma análise qualitativa do plano de aulas da disciplina Engenharia de Software da Instituição C, parte do curso de Análise e Desenvolvimento de Sistemas. O objetivo é compreender como os modelos de desenvolvimento de software, especialmente os métodos tradicionais e ágeis, estão presentes na estrutura curricular e quais são as estratégias pedagógicas utilizadas na formação discente.

Esta análise documental permite um mapeamento interpretativo do conteúdo programático, embora apresente limitações inerentes à ausência de dados práticos sobre sua implementação real.

a) Potencialidades da Abordagem Qualitativa

O PE_IES C destaca-se pela abrangência temática, integração entre teoria e prática e uso de metodologias ativas de ensino. A carga horária equilibrada entre teoria e prática (40h cada) revela uma intenção clara de articular o conteúdo com experiências laboratoriais e atividades aplicadas.

O conteúdo programático contempla tópicos fundamentais como:

- Ciclo de Vida do Software (*Waterfall,* Incremental, Espiral, Prototipação);
- Engenharia de Requisitos (funcionais, não funcionais, *MVP*, testes A/B);
- Modelagem (*UML, C4 Model*, Engenharia Reversa);
- Arquitetura de Software (*MVC*, Microsserviços, DDD);
- Versionamento com *Git*;
- Metodologias Ágeis (*SCRUM, Kanban, XP, SCRUMBan*);
- Testes de Software (unitários, funcionais, BDD, *smoke tests*);
- Refatoração, SOLID e Métricas de Código.

A presença de ferramentas práticas como *GitHub, Jira, Trello, VSCode, IntelliJ, Draw.io, Spring MVC e Cucumber* reforça o compromisso com uma formação atualizada e alinhada ao mercado. Além disso, o uso do Canvas como ambiente virtual, a presença de dinâmicas em grupo, avaliações formativas contínuas e projetos

finais em equipe refletem uma proposta pedagógica que valoriza o aprendizado ativo, colaborativo e centrado no estudante.

b) Limitações Inerentes à Abordagem Qualitativa

A análise qualitativa, por sua própria natureza, está limitada à interpretação dos dados disponíveis no documento institucional, não permitindo verificar a execução real da proposta pedagógica. Ainda que o plano seja extenso e detalhado, não é possível afirmar com precisão se todas as ferramentas, metodologias e atividades listadas são efetivamente aplicadas em sala de aula, ou se há variação significativa na prática docente.

Outra limitação refere-se à falta de detalhamento sobre o acompanhamento individual do desempenho discente, sobre o uso real de *feedbacks* estruturados e sobre a integração com práticas externas, como parcerias com empresas ou desafios reais. Não há também menção explícita à adoção de metodologias híbridas (ágil-tradicional) nem à abordagem *DevOps*, que atualmente são muito comuns na indústria.

Ainda que temas como arquitetura moderna, testes e refatoração estejam contemplados, conceitos avançados de segurança, escalabilidade em nuvem, CI/CD e cultura *DevSecOps* não aparecem explicitamente, o que pode indicar uma lacuna frente às exigências contemporâneas da Engenharia de Software em ambientes produtivos.

A análise do PE_IES C revela um currículo altamente estruturado, abrangente e moderno, que contempla tanto fundamentos teóricos quanto práticas aplicadas, com forte presença de metodologias ágeis e uso de ferramentas relevantes do mercado. A organização das aulas, os critérios de avaliação distribuídos ao longo do semestre, e os conteúdos atualizados refletem uma preocupação com a formação prática e a empregabilidade dos egressos.

A limitação da abordagem qualitativa, baseada exclusivamente na documentação curricular, impede a verificação empírica da efetividade das práticas pedagógicas propostas.

6.1.4 PE_IES D

A análise qualitativa tem como objetivo examinar o plano de ensino da disciplina Engenharia de Software da Instituição D, inserida no curso de Desenvolvimento de Sistemas. A abordagem adotada permite identificar, a partir do conteúdo programático e da bibliografia proposta, as intenções formativas, os enfoques metodológicos e os modelos de desenvolvimento de software trabalhados na formação discente. Contudo, como em toda pesquisa qualitativa baseada em documentos institucionais, a análise apresenta limitações decorrentes da ausência de evidências empíricas sobre a implementação pedagógica efetiva.

a) Potencialidades da Abordagem Qualitativa

O PE_IES D contempla uma estrutura sólida e abrangente, baseada em fundamentos clássicos da Engenharia de Software, com destaque para os seguintes eixos:

- Modelagem com *UML* (casos de uso, classes, atividades, interação, transição de estados);
- Engenharia de Requisitos (definições, análise, especificações);
- Planejamento e Métricas (estimativas, planos de projeto);
- Codificação (linguagens e tendências);
- Testes de Software (casos de teste);
- Manutenção (correções, melhorias e adaptações);
- Implantação e transição com referência ao *ITIL (Information Technology Infrastructure Library).*

O currículo apresenta ainda uma divisão clara das etapas do ciclo de vida de software, indo da análise de requisitos até a implantação, o que denota uma abordagem estruturada e sequencial, alinhada com os modelos tradicionais de desenvolvimento (como o modelo em Cascata). A bibliografia principal inclui autores consagrados como Pressman, Sommerville e Paula Filho, o que demonstra o compromisso da instituição com uma formação técnica robusta.

Outro ponto positivo é a inserção de conteúdos ligados a tendências e *frameworks* de TI (como ITIL), além de ferramentas de modelagem com *UML*, o que pode oferecer uma base útil para certificações e práticas no mercado corporativo.

b) Limitações Inerentes à Abordagem Qualitativa

A análise documental apresenta limitações importantes. Primeiramente, não há menção explícita a metodologias ágeis no conteúdo programático. Termos como *SCRUM, XP, Kanban* ou mesmo Manifesto Ágil não aparecem no documento, o que pode sugerir um descompasso com as práticas mais recentes adotadas pelo mercado de software contemporâneo, especialmente no contexto de *startups,* desenvolvimento Incremental e integração contínua.

Não são indicadas ferramentas práticas, ambientes de desenvolvimento, nem metodologias ativas (como projetos integradores, aprendizagem baseada em problemas ou uso de repositórios *Github*). A ausência dessas indicações impossibilita avaliar se o ensino é baseado em experiências reais de desenvolvimento ou se permanece centrado apenas em teoria e modelagem.

A bibliografia complementar, embora extensa, é composta majoritariamente por obras publicadas entre as décadas de 1990 e 2000, com poucas referências contemporâneas. Isso pode indicar um currículo com baixa atualização frente aos avanços da Engenharia de Software nos últimos dez anos, especialmente no que tange a temas como *DevOps, cloud computing,* segurança no ciclo de vida do software, testes automatizados e *frameworks* ágeis híbridos.

O PE_IES D apresenta um currículo estruturado, com foco em fundamentos teóricos sólidos e nas fases clássicas do desenvolvimento de software. A proposta pedagógica valoriza a modelagem e documentação, e oferece uma formação alinhada com práticas tradicionais do setor.

A ausência de tópicos relacionados às metodologias ágeis, às ferramentas modernas de desenvolvimento e à cultura *DevOps* representa uma limitação significativa no contexto atual da Engenharia de Software. Do ponto de vista da análise qualitativa, a falta de indicadores sobre a prática docente impede uma avaliação aprofundada da efetividade da aprendizagem.

6.1.5 PE_IES E

A análise qualitativa examina a estrutura curricular da disciplina Engenharia de Software ofertada pela Instituição E. O estudo busca compreender como os conteúdos, competências e práticas relacionados ao desenvolvimento de software,

tanto em abordagens tradicionais quanto ágeis são organizados e propostos no plano didático.

A análise está fundamentada na leitura interpretativa de documentos institucionais, assumindo como limitação principal a ausência de dados empíricos sobre a aplicação prática das estratégias de ensino.

a) Potencialidades da Abordagem Qualitativa

O plano analisado demonstra uma estrutura fortemente segmentada e bem distribuída em quatro unidades temáticas, o que favorece a organização do conteúdo e o acompanhamento da aprendizagem por parte dos discentes.

i. **Fundamentos de Engenharia de Software**:
   a. Trata de princípios, mitos, ciclos de vida e tipos de software.
   b. Introduz e compara metodologias ágeis (*SCRUM*, XP) com modelos tradicionais.
   c. Aborda o uso de controle de versões, incluindo repositórios e práticas de *release*, o que é altamente relevante para a prática contemporânea.

ii. **Qualidade de Software**:
   a. Inclui tópicos como métricas, normas ISO (9001, 9126), *CMMI,* MPS.BR, e *PSP.*
   b. Evidencia uma preocupação com padrões internacionais e nacionais, capacitando o aluno a compreender e aplicar critérios formais de qualidade de produto e processo.

iii. **Testes de Software**:
   a. Trabalha com tipos diversos de testes (funcionais, estruturais, orientados a objetos, web/móveis).
   b. Introduz *TDD* e ferramentas *CASE*.
   c. Essa abordagem demonstra integração entre teoria e aplicações de mercado, como testes automatizados e gestão de *bugs*.

iv. **Auditoria de Sistemas**:

a. Apresenta fundamentos e práticas de auditoria, manutenção e evolução de sistemas.
b. Destaca engenharia reversa, reengenharia e ferramentas de manutenção, aspectos importantes para a sustentação de sistemas legados.

Outro ponto positivo é a presença explícita de resultados de aprendizagem esperados, que definem competências técnicas e atitudinais de forma clara. Isso facilita o alinhamento entre conteúdo, avaliação e mercado de trabalho.

b) Limitações Inerentes à Abordagem Qualitativa

Embora o plano de ensino seja detalhado e atual em diversos aspectos, a análise qualitativa apresenta limitações importantes. A principal diz respeito à ausência de evidências sobre como o conteúdo é operacionalizado na prática, ou seja, não se sabe como são conduzidas as aulas, quais metodologias são de fato aplicadas e se há integração com o mercado ou projetos reais.

Não há referência direta a ferramentas específicas de desenvolvimento, como ambientes de versionamento, plataformas de integração contínua, *IDE*, ou ferramentas de prototipação. Embora o plano mencione “ferramentas *CASE*”, ele não especifica quais são utilizadas ou recomendadas.

Ainda que as metodologias ágeis sejam abordadas, a ênfase parece ser teórica, sem menção a dinâmicas como *sprints,* planning poker, *stand-up meetings* ou uso de ferramentas como *Trello, Jira, Azure DevOps* etc., o que pode limitar o desenvolvimento de habilidades práticas fundamentais.

Não há sinalização de metodologias ativas de ensino, como aprendizagem baseada em projetos (PBLj), aprendizagem por pares ou desenvolvimento colaborativo de software. A ausência desses elementos impede inferências sobre a dimensão prática da formação.

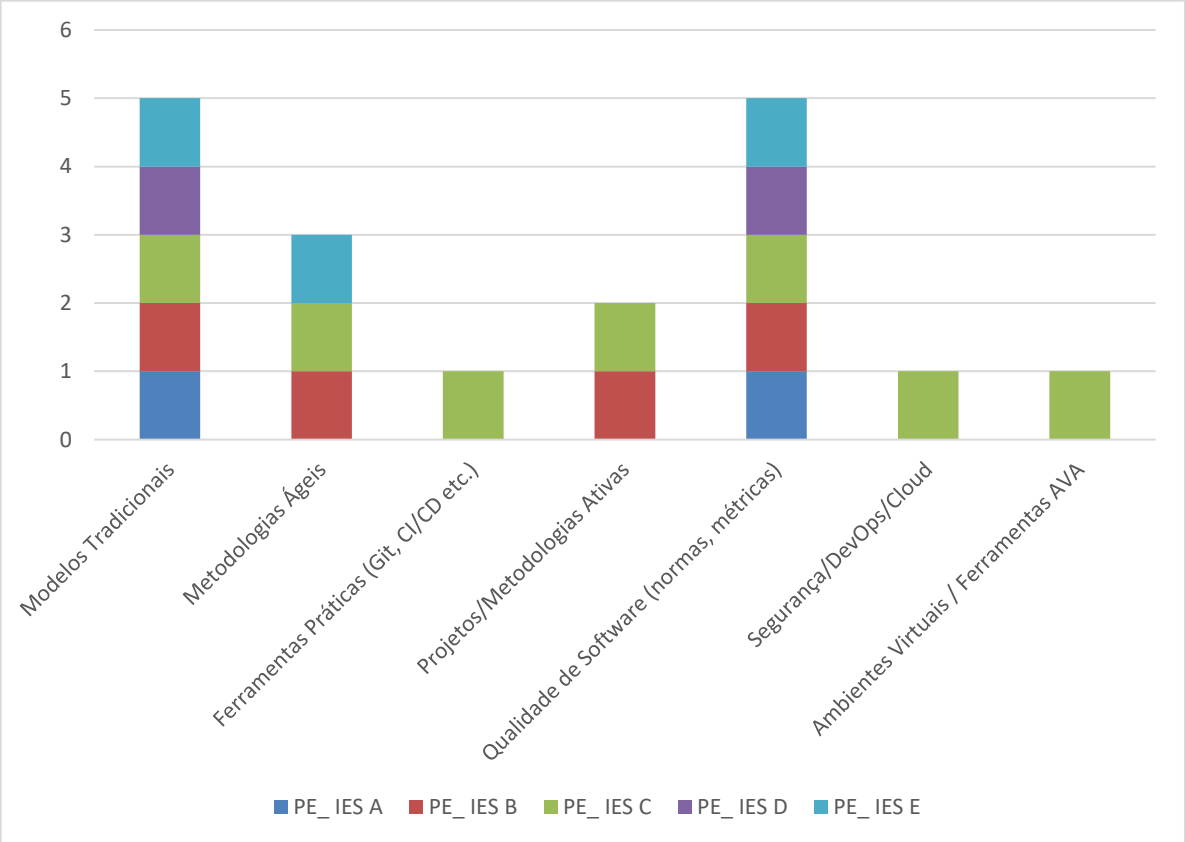
O plano apresenta um currículo abrangente, segmentado por eixos temáticos claros e coerentes, que incluem tanto abordagens tradicionais quanto práticas modernas. O conteúdo contempla aspectos importantes da Engenharia de Software, como qualidade, testes e auditoria e introduz temas relevantes como metodologias ágeis e *TDD*.

Entretanto, a análise qualitativa, limitada ao exame do plano de ensino, não permite verificar o nível de aderência prática, o uso de ferramentas reais nem a efetividade das estratégias de ensino-aprendizagem propostas.

A análise comparativa dos planos de ensino das cinco instituições particulares da cidade de Sorocaba evidencia diferenças significativas na abordagem da disciplina de Engenharia de Software, especialmente no que se refere à integração entre teoria e prática, à atualização tecnológica e ao uso de metodologias ativas.

O gráfico 2, comparativo entre todas os PE_IES (A, B, C, D e E), contempla modelos tradicionais de desenvolvimento de software, como o modelo Cascata, Incremental, Espiral e Prototipação, o que demonstra a presença consolidada desses paradigmas clássicos no ensino da área. No entanto, quando se trata da inclusão de metodologias ágeis, observa-se que apenas os PE_IES B, C e E fazem menção explícita a práticas como *SCRUM, XP* ou *Kanban*.

Gráfico 2: Comparativos entre as IES e seus conteúdos programáticos

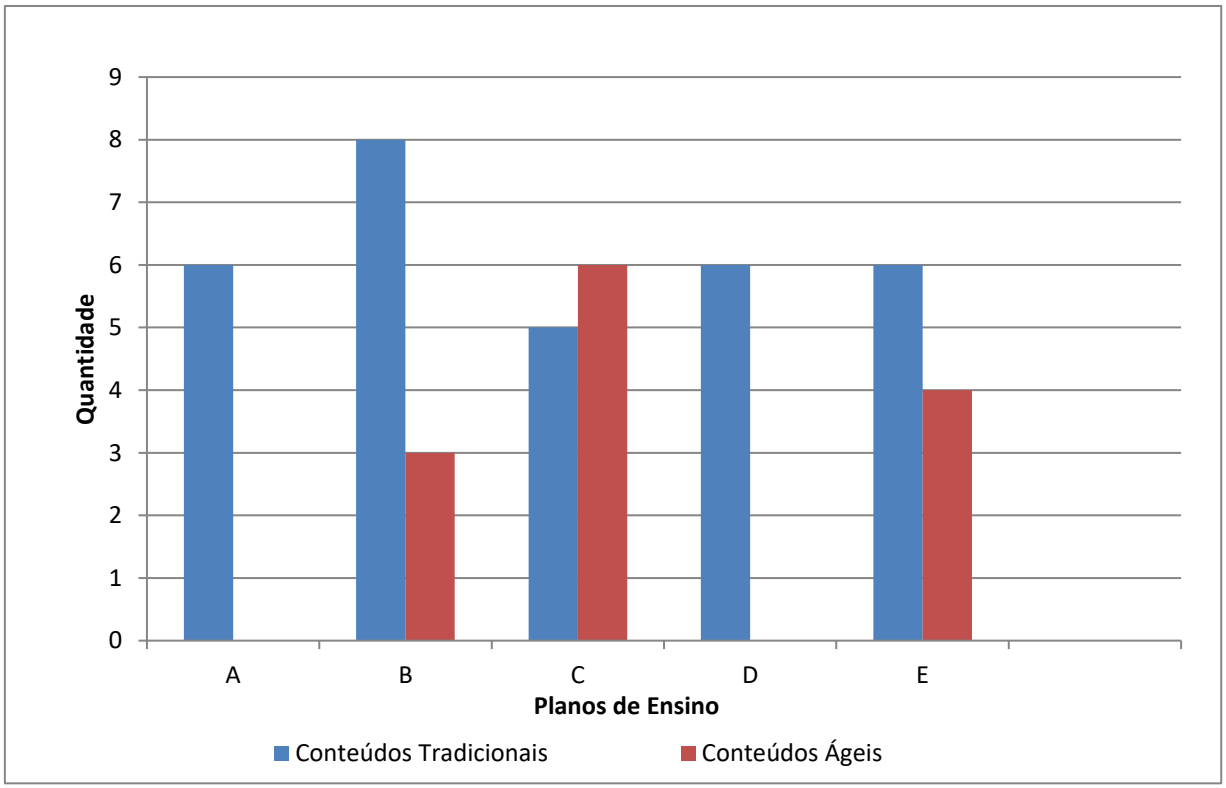


Fonte: Elaboração própria

O PE_IES C destaca-se de forma contundente ao apresentar não apenas conteúdos atualizados sobre metodologias ágeis, mas também ao integrar ferramentas práticas utilizadas pelo mercado, como *Git, C4 Model, Cucumber e Jira*. Essa instituição também é a única que contempla ambientes virtuais de aprendizagem (AVA), evidenciando uma proposta pedagógica mais alinhada com as demandas contemporâneas da educação em tecnologia.

No gráfico 3, relata-se a comparação entre as cinco instituições sobre conteúdos de modelo de processos tradicionais em relação ao modelo de processo ágil, que tem hoje nos planos de ensino, O eixo y (quantidade) refere-se à quantidade de itens de cada modelo que consta em cada plano de ensino.

Gráfico 3: Comparativo de conteúdo Tradicional x Ágil



Fonte: Elaboração própria

Por outro lado, os PE_IES A e D apresentam uma proposta curricular mais tradicional, sem evidências de adoção de metodologias ágeis, ferramentas de apoio ao desenvolvimento, ou estratégias ativas de ensino. Embora ambas apresentem bibliografia consistente e conteúdos teóricos estruturados, suas propostas carecem

de elementos que promovam a prática aplicada e o desenvolvimento de competências atualizadas.

A análise revela também que todas as instituições analisadas abordam conteúdos ligados à qualidade de software, incluindo métricas, normas ISO e processos de verificação e validação. Esse aspecto pode estar relacionado à presença histórica desses tópicos nas bibliografias clássicas (ex.: Pressman, Sommerville), frequentemente adotadas nos cursos.

No que diz respeito a temas emergentes, como segurança no ciclo de vida do software, *DevOps* e computação em nuvem, apenas o PE_IES C demonstra alguma abordagem inicial, ainda que parcial. A ausência desses conteúdos nas demais instituições reforça a necessidade de atualização curricular frente às transformações do setor.

A adoção de projetos práticos e metodologias ativas também é limitada: somente os PE_IES B e C apresentam iniciativas nessa direção, com destaque para projetos integradores, dinâmicas de grupo e uso de tecnologias colaborativas.

6.2 Análise Documental Análise Qualitativa do Currículo da Disciplina Engenharia de Software

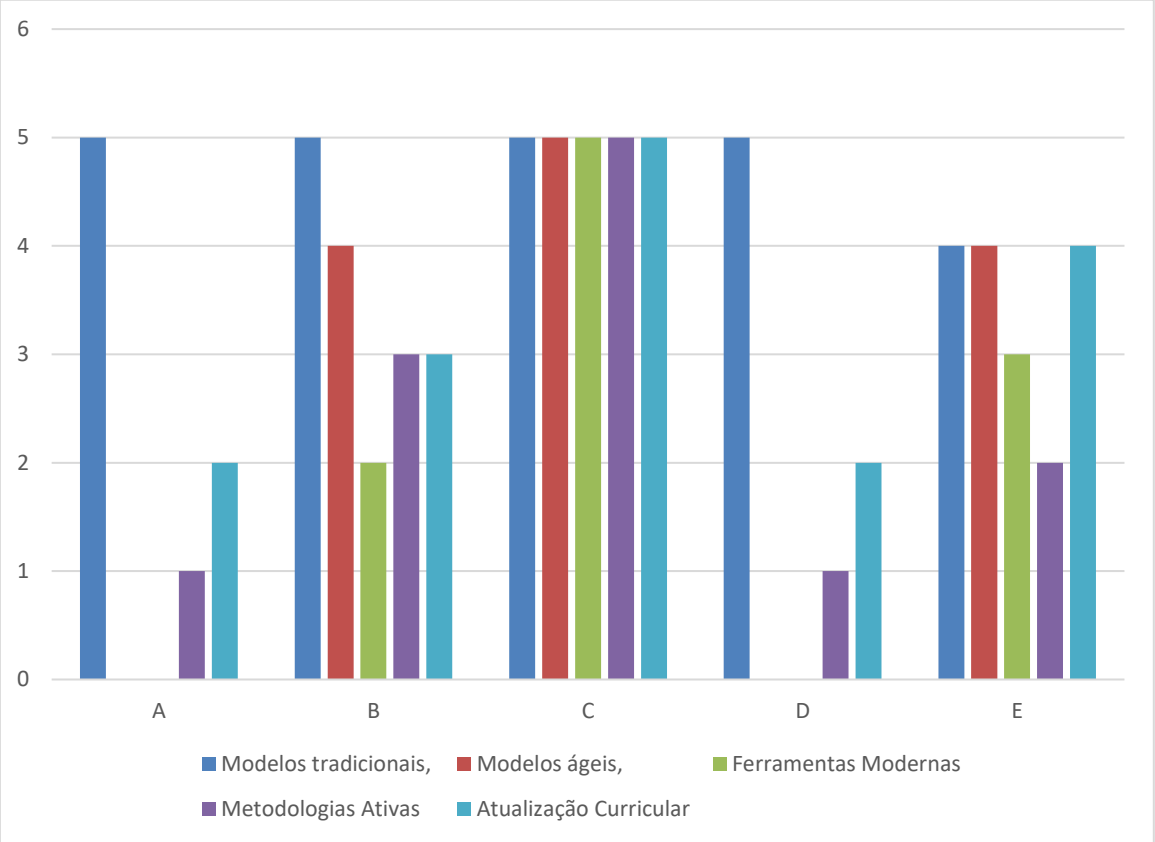
Nesta parte do estudo, serão demonstrados, de maneira organizada os resultados de forma comparativa, destacando as diferenças e semelhanças no modo como os modelos tradicionais e ágeis são ensinados com gráficos entre as 5 instituições. Uma análise qualitativa de natureza documental realizada a partir dos planos de ensino da disciplina Engenharia de Software, lembrando com codinomes, como PE_IES A, B, C, D e E.

O objetivo é demonstrar como os modelos tradicionais e ágeis de desenvolvimento de software estão integrados no currículo de cada instituição, permitindo um mapeamento detalhado e sistemático das práticas educacionais adotadas.

O gráfico 4 apresenta o comparativo entre os PE_IES, no que se refere-se à Modelos tradicionais, Modelos ágeis, ferramentas modernas em aula, metodologia ativa e atualização curricular. A escala utilizada refere-se à quantidade de ocorrências ou presença de cada item curricular analisado. A escala vai de 0 a 6 e cada barra indica quantas vezes aquela dimensão (ex.: modelos tradicionais, ágeis, ferramentas

modernas, metodologias ativas, atualização curricular) aparece ou é aplicada no currículo da universidade correspondente.

Gráfico 4: Comparativo Curricular



Fonte: Elaboração própria

A análise se baseia-se na leitura interpretativa dos documentos institucionais disponibilizados pelas instituições. A abordagem qualitativa permite observar de forma aprofundada os objetivos formativos, os conteúdos programáticos, as referências bibliográficas e os métodos de ensino e avaliação previstos.

A análise apresenta limitações por não incluir dados empíricos sobre a aplicação prática das estratégias curriculares.

6.2.1 Análise Documental Comparada

O PE_IES A possui forte base nos modelos tradicionais da Engenharia de Software, com ênfase em análise, projeto, testes, métricas, gerenciamento de

configuração e qualidade. A bibliografia é clássica e centrada em autores como Pressman e Sommerville. Embora o plano seja estruturado, não há menção direta a metodologias ágeis, ferramentas de versionamento ou práticas de ensino aplicadas.

O PE_IES B está distribuído em duas disciplinas (Engenharia de Software I e II). A primeira aborda diversos modelos, incluindo tradicionais (Cascata, Espiral, *RUP*) e ágeis (*XP*, *SCRUM, FDD, DSDM*). A segunda disciplina enfoca qualidade, testes e manutenção, com referência a normas ISO e modelos como *CMMI* e MPS.Br.

O PE_IES C apresenta o currículo mais detalhado e atualizado entre as instituições analisadas. Integra modelos tradicionais e ágeis, práticas como versionamento com *Git, TDD*, uso de ferramentas como *Jira, Trello, Draw.io*, práticas de arquitetura moderna (*DDD,* microsserviços), testes automatizados e refatoração. A presença de metodologias ativas, ferramentas reais de mercado e atividades práticas contínuas refletem uma abordagem alinhada com as demandas atuais da indústria.

A proposta do PE_IES D é centrada em fundamentos tradicionais, com foco em análise, projeto, testes e manutenção. Não há referências a metodologias ágeis nem uso de ferramentas específicas. A bibliografia é predominantemente clássica, sem menção a práticas contemporâneas como *DevOps*, CI/CD ou desenvolvimento ágil. Trata-se de um currículo com forte embasamento teórico, porém com baixa aderência às práticas emergentes.

No quadro 6, demonstra os pontos fortes e fracos de cada PE_IES

Quadro 6 A análise comparativa

PE_IES	Pontos Fortes	Pontos Fracos
A	• Fundamentação teórica sólida • Ênfase em modelos tradicionais	• Ausência de metodologias ágeis • Pouca prática • Sem ferramentas modernas
B	• Início da adoção de metodologias ágeis • Uso moderado de metodologias ativas	• Ferramentas de mercado pouco exploradas • Atualização curricular parcial
C	• Metodologias ágeis bem aplicadas • Projetos práticos • Ferramentas modernas • AVA estruturado	• Carga de teoria ainda relevante • Necessidade de expansão de parcerias
D	• Conteúdo clássico e estruturado	• Currículo conservador • Sem aplicação prática • Não utiliza AVA • Sem tecnologias de mercado
E	• Adota metodologias híbridas • Algum uso de ferramentas	• Metodologias ativas superficiais • Falta integração com mercado

Fonte: Elaboração própria

O PE_IES E se estrutura em quatro grandes unidades: fundamentos, qualidade, testes e auditoria de sistemas. Há inserção clara de metodologias ágeis (*SCRUM*, XP), controle de versões, *TDD* e ferramentas *CASE*. A abordagem é organizada e abrange temas relevantes, mas sem detalhamento de metodologias ativas ou projetos reais. Ainda assim, destaca-se por abordar modelos ágeis e padrões de qualidade com clareza.

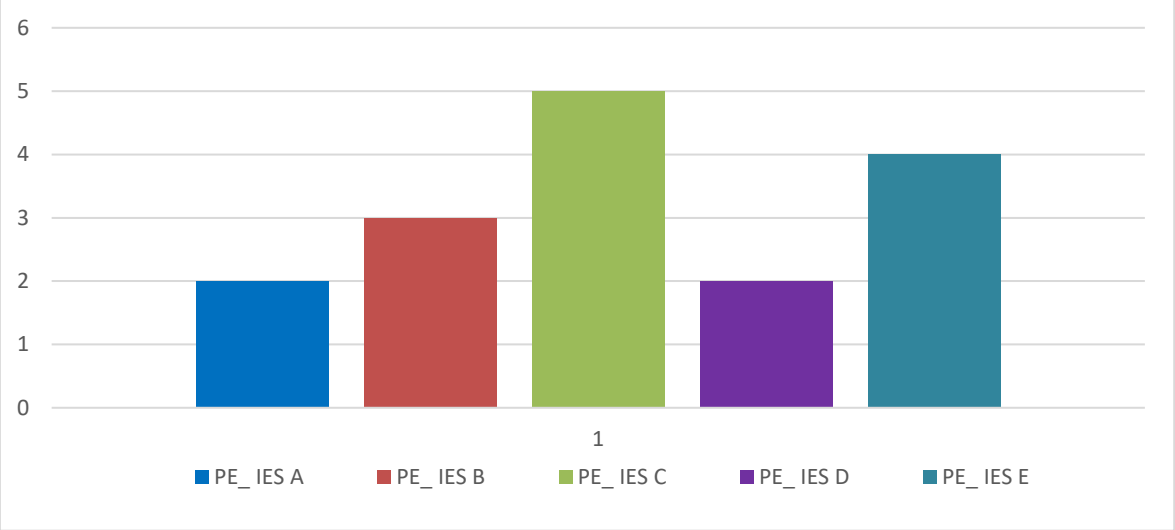
A análise documental das cinco instituições permite concluir que há níveis variados de integração entre os modelos tradicionais e ágeis de desenvolvimento de software nos currículos analisados. Enquanto o PE_IES A e D mantêm um foco tradicional, PE_IES B e E demonstram um equilíbrio maior entre abordagens clássicas e contemporâneas. A IES C, por sua vez, apresenta o currículo mais aderente às práticas modernas da Engenharia de Software, incorporando metodologias ágeis, ferramentas de mercado e estratégias pedagógicas ativas.

Os gráficos de barras 5, 6, 7, 8 e 9 demonstram, de forma separada, o desempenho das IES nos cinco critérios analisados conforme seguem:

Modelos Tradicionais

O gráfico 5, revela que todos os planos analisados mantêm forte presença dos modelos tradicionais de desenvolvimento de software, como o modelo Cascata, Prototipação, Espiral, Incremento e *RUP.* O gráfico informa que apenas no PE_IES E não consta o conteúdo do modelo tradicional básico que são 5 itens.

Gráfico 5: Modelos Tradicionais



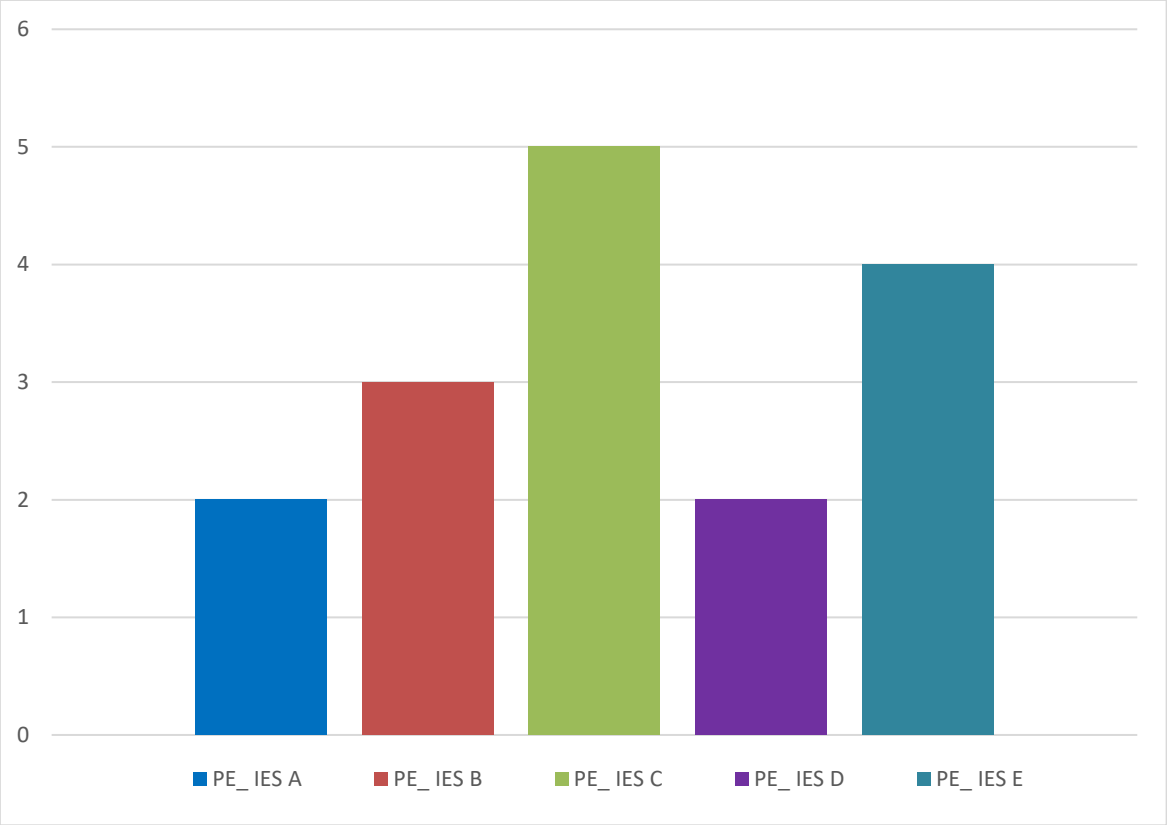
Fonte: Elaboração própria

Essa uniformidade indica que os fundamentos clássicos da Engenharia de Software ainda são considerados essenciais na formação dos estudantes. Os PE_IES A e D, em particular, demonstram grande dependência desses paradigmas, o que contribui para uma base teórica sólida, mas pode limitar a atualização metodológica se não houver complementaridade com práticas contemporâneas.

Modelos Ágeis

O gráfico 6 mostra que apenas três PE_IES B, C e E incorporam metodologias ágeis de maneira explícita em seus currículos. Destaca-se PE_IES C, cuja abordagem integra práticas como *SCRUM, XP e Kanban* não apenas em teoria, mas também em atividades práticas. O eixo vertical (y do gráfico) explica a escala de quantas vezes aparece no plano de ensino os itens sobre o processo de práticas do modelo ágil.

Gráfico 6: Modelos Ágeis

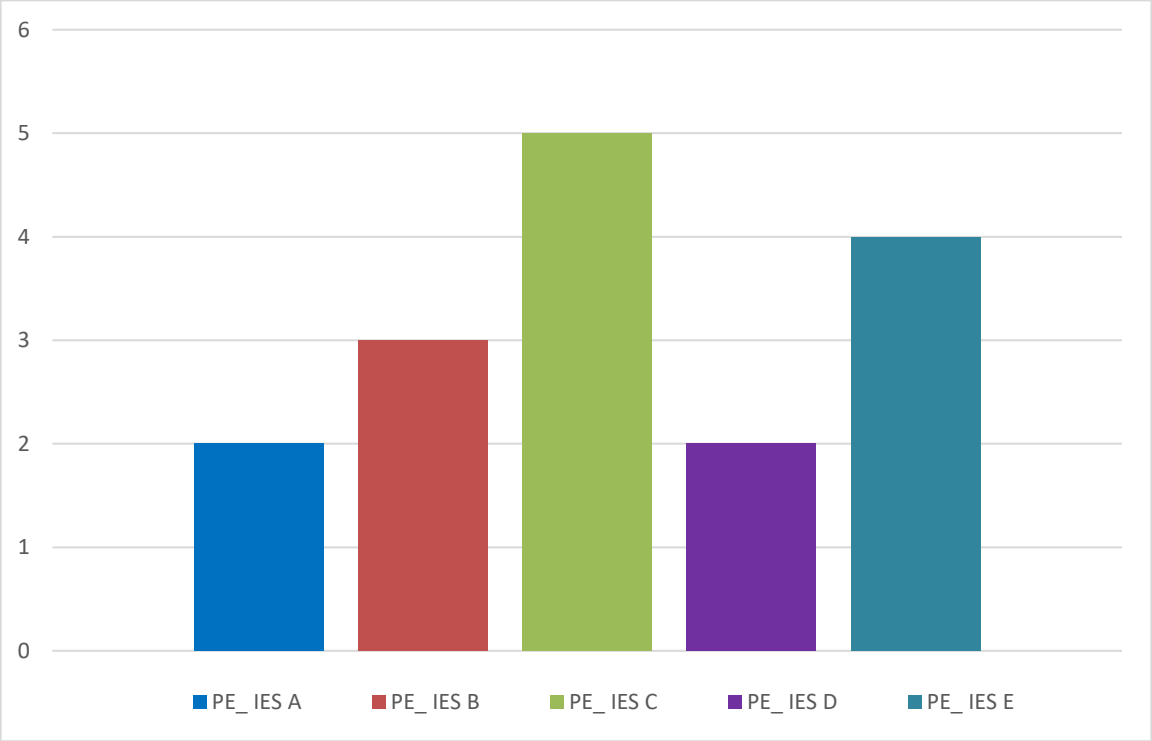


Fonte: Elaboração própria

Já os PE_IES A e D não apresentam qualquer menção a modelos ágeis, evidenciando uma lacuna relevante frente às exigências atuais da indústria de software, que valoriza a agilidade, adaptabilidade e entrega contínua de valor

O gráfico 7 evidencia uma disparidade marcante entre as instituições quanto à utilização de ferramentas modernas de desenvolvimento, como *Git, Jira, Jenkins* e plataformas de *CI/CD* que, do total analisado no documento do plano de ensino consta em cada IES, e assim, deixa o eixo do gráfico y, que consta todos os requisitos listado como índice máximo 5.

Gráfico 7: Ferramentas Modernas

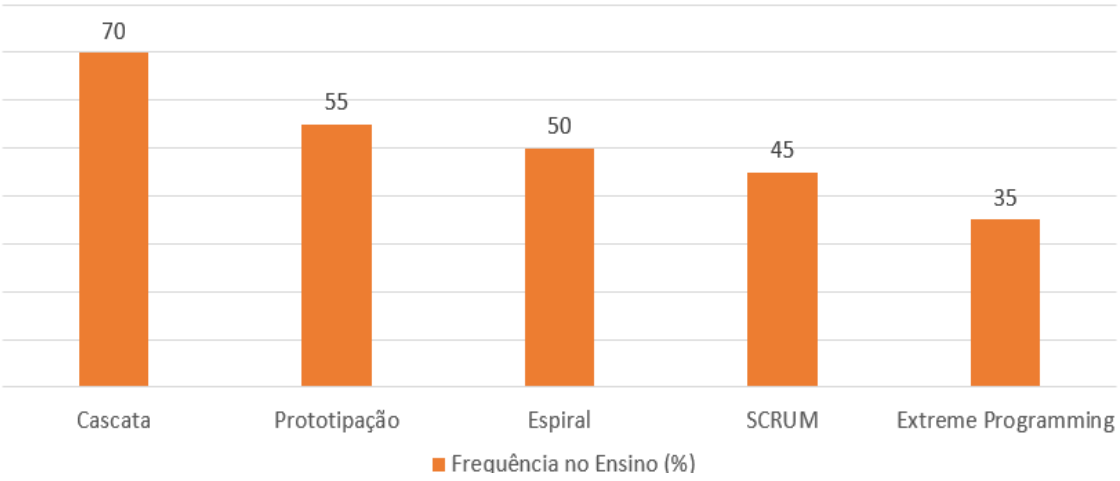


Fonte: Elaboração própria

O PE_IES C mais uma vez se sobressai, sendo a único a apresentar integração clara de ferramentas utilizadas no mercado real. Os demais PE_IES, em especial os das A, D e E não especificam o uso de tecnologias práticas, o que limita a preparação técnica dos estudantes para ambientes profissionais contemporâneos.

Os dados apresentados no gráfico 8 indicam quanto cada modelo de processo tradicional e ágil aparecem em % dentro dos planos de ensino das cinco instituições particulares de Sorocaba.

Gráfico 8: Modelos Tradicional e Ágil

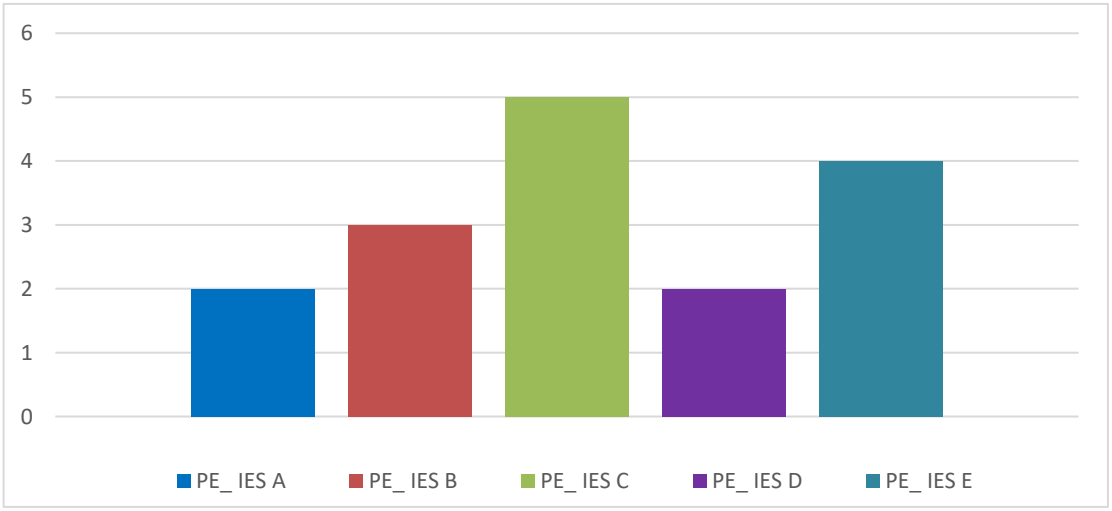


Fonte: Elaboração própria

Atualização Curricular

O gráfico 9 que sintetiza o grau de atualização curricular aponta que o PE_IES C possui o plano mais alinhado às práticas contemporâneas de Engenharia de Software. A instituição não apenas aborda conteúdos atualizados, como também os integra de forma prática e colaborativa. Para melhor entendimento do gráfico, cada valor da coluna vertical (eixo Y) corresponde ao grau de atualização curricular atribuído a cada IES (de 0 = ausência até 5 = presença muito forte). Esses valores foram obtidos a partir da análise documental dos currículos (quantificação qualitativa).

Gráfico 9: Atualização Curricular



Fonte: Elaboração própria

Por outro lado, as IES A e D apresentam currículos mais conservadores, baseados em referências clássicas e sem inserção de tecnologias emergentes. Essa constatação reforça a necessidade de revisão periódica das ementas, conforme defendido pelas Diretrizes Curriculares Nacionais, que recomendam a flexibilização e modernização contínua dos currículos universitários

Como reforça Pressman (2016), a Engenharia de Software deve evoluir com a indústria, sob o risco de formar profissionais para um passado que já não existe.

Todos os currículos analisados mantêm os fundamentos clássicos da Engenharia de Software, como o modelo Cascata, Espiral e análise estruturada. Isso evidencia uma uniformidade na base teórica da disciplina entre as instituições, servindo como um ponto de partida sólido para a formação técnica.

Somente os PE_IES B, C e E incluem modelos ágeis de forma explícita. Destas, a IES C é a única que alia teoria e prática, incorporando práticas reais como *SCRUM Ban* e XP. Já A e D permanecem com um viés exclusivamente tradicional, o que representa um descompasso com as práticas atuais de mercado.

A discrepância é mais acentuada neste critério. Apenas o PE_IES C demonstra uso efetivo de ferramentas modernas como *Git, Trello, Jira, Docker* e integração contínua (*CI/CD*). As demais instituições ou não mencionam essas ferramentas ou fazem apenas referências genéricas, indicando baixa aderência às práticas técnicas do setor.

Novamente, apenas PE_IES B e C se destacam com uso de projetos integradores, dinâmicas em grupo e avaliação prática. Isso favorece a construção de habilidades como trabalho em equipe e resolução de problemas. A ausência dessas estratégias no PE_IES A, D e E aponta para uma formação mais expositiva e menos experiencial.

No conjunto dos cinco critérios, o PE_IES C é a única que se aproxima de um modelo curricular moderno, dinâmico e alinhado com o mercado. PE_IES A e D apresentam estruturas mais engessadas, centradas em bibliografias antigas e ausência de práticas contemporâneas. B e E demonstram avanços pontuais, mas ainda não integrados de forma sistêmica.

A análise dos cinco indicadores revela que há um desequilíbrio entre tradição e inovação nos currículos das IES analisadas. Enquanto a IES C se posiciona como referência em atualização pedagógica e técnica, outras instituições permanecem

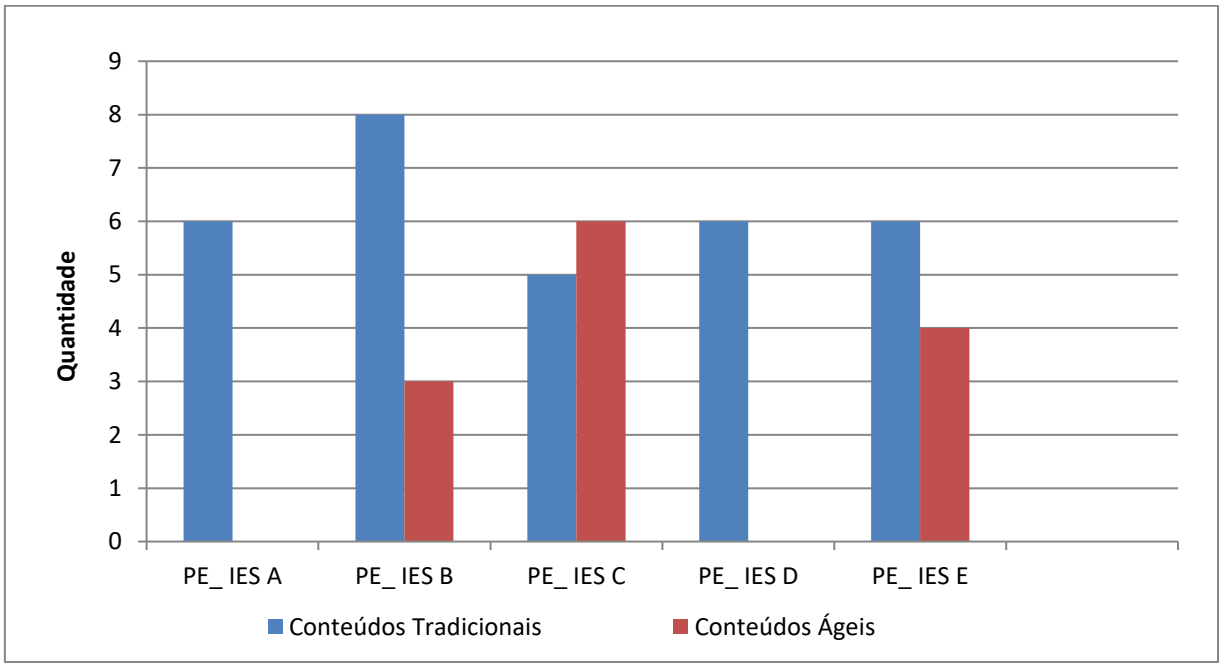
ancoradas em modelos ultrapassados, oferecendo uma formação que não contempla plenamente as necessidades do setor de tecnologia.

6.2.2 Desempenho e Relevância dos Modelos de Processos Tradicionais e Ágeis da Engenharia de Software

Na análise dos Modelos de Processos Tradicionais descreve resultados destacando as instituições que priorizam essas abordagens e os motivos que justificam essa escolha, como o modelo Cascata, Espiral, Incremental, Prototipação e processo unificado, que permanece amplamente presente nas propostas curriculares analisadas. A abordagem documental revelou que todas as cinco instituições investigadas (A, B, C, D e E) incluem, em maior ou menor grau, tais modelos como parte essencial da formação em Engenharia de Software.

O gráfico 10 mostra a distribuição de conteúdo nos cincos planos de ensino que tratam da quantidade de conteúdos de modelos de processos tradicionais e de modelos de processos ágeis. O eixo Y do gráfico 10 representa a quantidade absoluta de conteúdos relacionados a Modelos Tradicionais e Modelos Ágeis. A escala foi definida de 0 a 9, porque o maior valor encontrado foi 8 conteúdos, e o acréscimo até 9 garante melhor legibilidade.

Gráfico 10: Distribuição de Conteúdo Tradicionais e Ágeis



Fonte: Elaboração própria

Conforme o gráfico 11, os PE_IES A e D se destacam por uma ênfase marcadamente tradicionalista. Ambas estruturam seus conteúdos com base nos princípios clássicos da Engenharia de Software, focando em etapas sequenciais do ciclo de vida do software que são os levantamentos de requisitos (Funcional e Não Funcional), modelagem, codificação, testes e manutenção.

Os planos de ensino dessas instituições valorizam a abordagem sistemática e disciplinada, muitas vezes inspirada no modelo Cascata, com pouca ou nenhuma menção explícita a abordagens ágeis. O eixo X, separa as duas categorias principais (Tradicional e Ágil) para efeito comparativo entre as instituições. Já o eixo Y, mostra a pontuação atribuída à presença desses modelos nos currículos (de 0 a 5), permitindo comparar se cada IES tem mais aderência ao tradicional ou ao ágil.

Gráfico 11: Tradicional x Ágil

Fonte: Elaboração própria

Essa priorização pode ser justificada por diversos fatores:

- Estabilidade conceitual: Modelos tradicionais oferecem uma base teórica consolidada, útil para introdução de estudantes aos processos formais de engenharia.

- Bibliografia clássica: A adoção de autores como Pressman e Sommerville, recorrente nas instituições A e D, reforça o alinhamento com práticas consagradas.
- Enfoque acadêmico: Em contextos institucionais mais conservadores, o currículo tende a preservar conteúdos historicamente consolidados como essenciais.

Os PE_IES B e E adotam uma abordagem mais equilibrada. Em seus planos de ensino, os modelos tradicionais são ensinados em conjunto com os modelos ágeis, favorecendo uma visão comparativa e crítica entre as diferentes metodologias. A presença de tópicos como *RUP*, modelo Incremental, Espiral, *RAD e* Prototipação é explícita, geralmente associada a estudos de caso e análises reflexivas. Esse equilíbrio parece refletir um movimento curricular consciente de preparar os estudantes para diferentes contextos de projeto desde os ambientes mais regulados, que exigem documentação extensiva, até contextos mais dinâmicos e iterativos, que valorizam a adaptabilidade.

Já o PE_IES C não apenas contempla os modelos tradicionais, como também os articula diretamente com práticas contemporâneas. Modelos como Espiral, Incremental e *RUP* são abordados em conjunto com práticas modernas como versionamento, testes automatizados e arquitetura baseada em microsserviços. Essa integração demonstra uma compreensão de que os modelos tradicionais ainda têm valor, especialmente no planejamento e controle, mas que precisam ser contextualizados às demandas reais de projetos de software modernos.

A análise dos planos de ensino evidencia que os modelos tradicionais de desenvolvimento continuam sendo uma peça central no ensino da Engenharia de Software nas instituições analisadas. No entanto, a forma como esses modelos são abordados varia significativamente:

- Enquanto os PE_IES A e D adotam uma postura mais ortodoxa e teórica;
- Os PE_IES B e E tentam construir uma ponte entre o tradicional e o contemporâneo;
- O PE_IES C aplica uma visão moderna, que reconhece o valor histórico dos modelos clássicos, mas promove sua contextualização prática frente às necessidades do mercado.

Essas variações curriculares refletem tanto o perfil institucional quanto o grau de abertura à inovação pedagógica e tecnológica. Para além da permanência dos

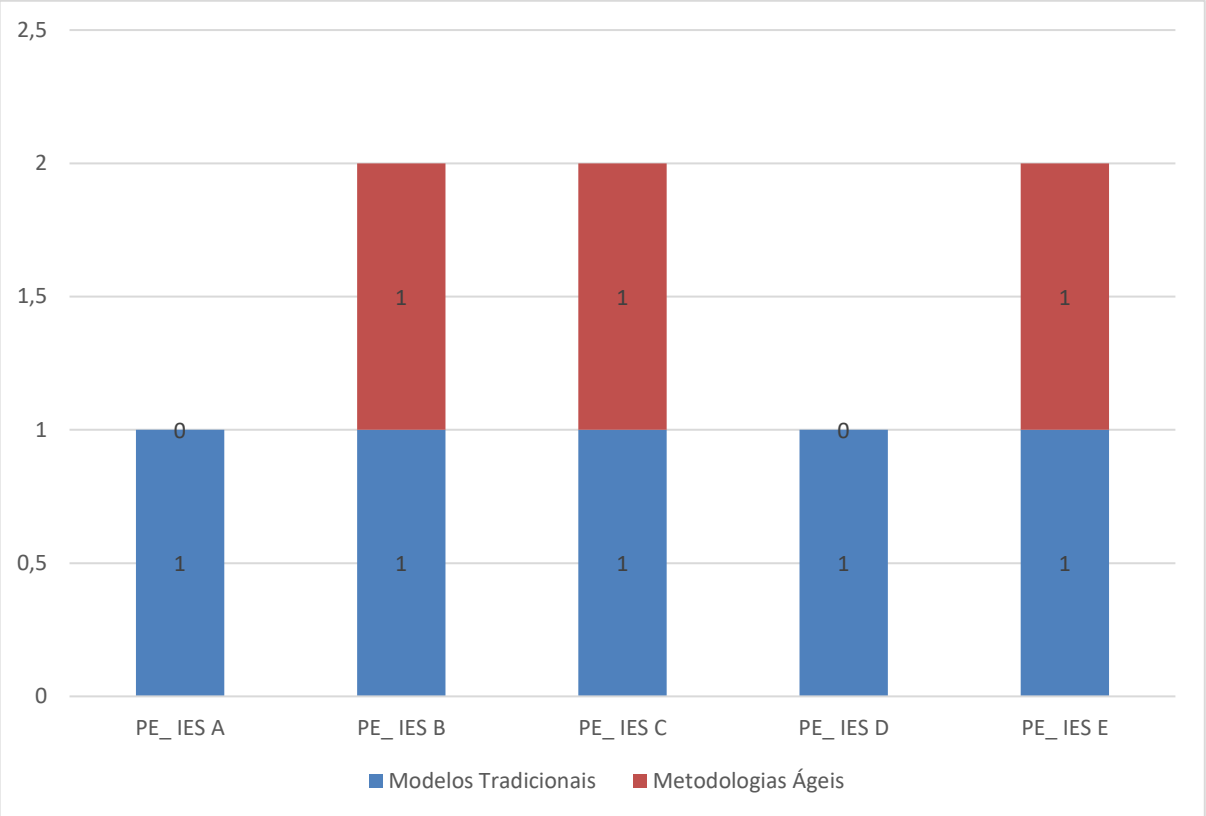
modelos tradicionais, torna-se essencial garantir que os estudantes desenvolvam senso crítico para avaliar, escolher e adaptar metodologias conforme o tipo de projeto, equipe, cliente e restrições envolvidas.

O gráfico 12 representa visualmente a distribuição curricular entre modelos tradicionais e metodologias ágeis nas cinco IES analisadas. Para esse gráfico a escala é definida se apresenta ou não os modelos de processos no plano de ensino. Se sim, considera uma barra, se não aparece, não tem a barra do gráfico.

Ele destaca:

- A forte ênfase tradicional nas IES A e D.
- O equilíbrio nas IES B e E.
- A abordagem mais integrada e contemporânea da IES C, que combina bem ambas as vertentes.

Gráfico 12: Distribuição Curricular: Modelos Tradicionais vs Metodologias Ágeis



Fonte: Elaboração própria

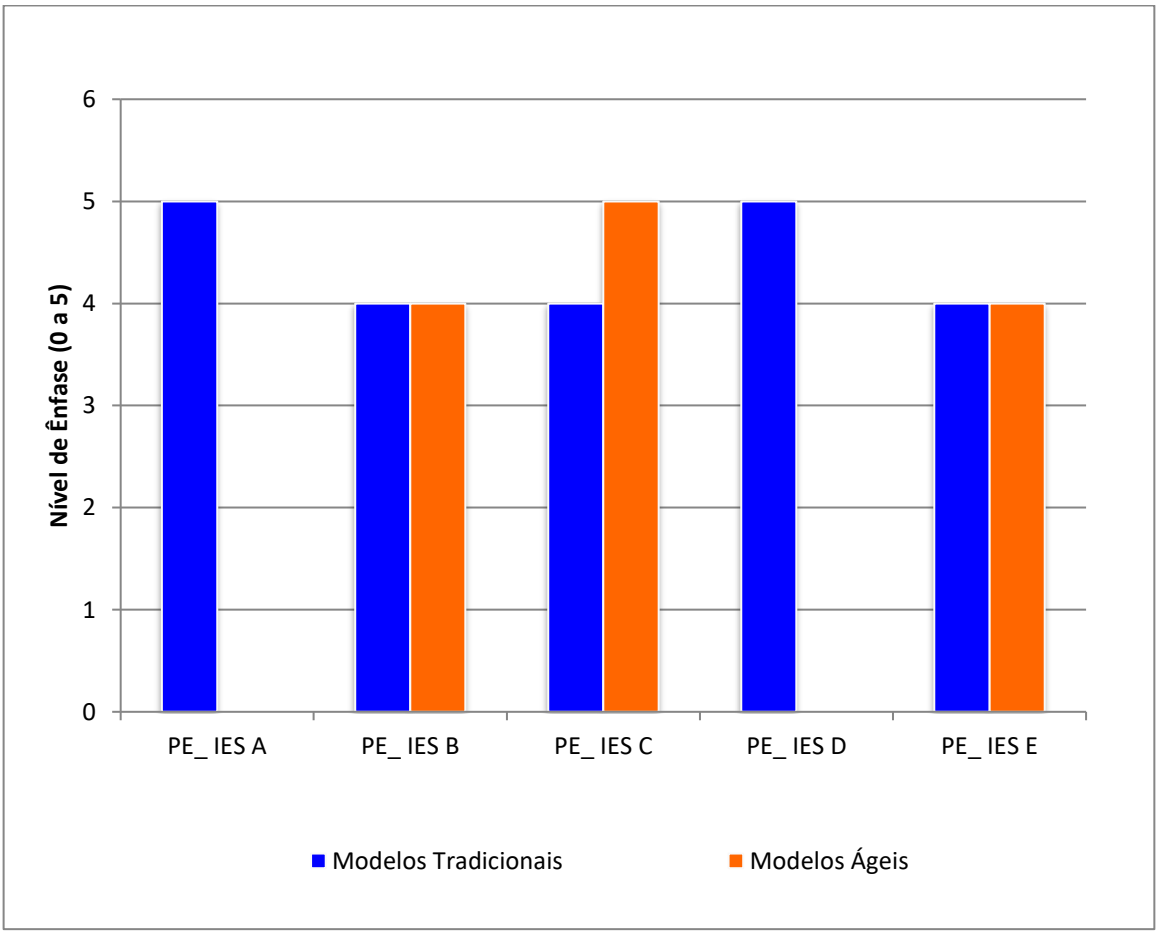
A análise dos planos de ensino das universidades investigadas revela diferentes níveis de integração entre modelos tradicionais e metodologias ágeis nos

cursos de Engenharia de Software. Os dados foram representados graficamente para facilitar a visualização comparativa entre as instituições.

Para o gráfico 13, os PE_IES A e D apresentam uma ênfase quase exclusiva em modelos tradicionais. Com pontuações máximas (5) em modelos como Cascata, Espiral e Incremental e Prototipação, essas instituições mantêm currículos fortemente orientados à formalização de processos, reforçando uma visão sequencial e estruturada do ciclo de vida do software. A ausência de modelos ágeis indica um ensino voltado a práticas mais antigas, pouco adaptadas às dinâmicas atuais do desenvolvimento de software.

O eixo Y do gráfico representa o Nível de Ênfase atribuído à presença dos modelos de desenvolvimento de software (tradicionais e ágeis). Escala varia de 0 a 5, o qual o nível 5 significa máxima ênfase (forte presença nos currículos), e o “0” indicaria ausência total.

Gráfico 13: Abordagem de Desenvolvimento de Software nos Documentos Curriculares:



Fonte: Elaboração própria

Os PE_IES B e E mostram uma distribuição mais equilibrada, com forte presença tanto de métodos tradicionais quanto de métodos ágeis. Essa dualidade aponta para uma tentativa de contextualização pedagógica, oferecendo aos estudantes uma visão comparativa entre metodologias preditivas e iterativas.

O PE_IES C se destaca como a única a apresentar uma abordagem robusta e simultânea dos dois paradigmas conforme aparece no gráfico 13. Além de contemplar os modelos tradicionais, ela introduz práticas modernas como *SCRUM, Kanban*, XP, versionamento com *Git, TDD*, arquitetura baseada em microsserviços e ferramentas de colaboração e integração. Tal abordagem indica um currículo alinhado com o mercado atual, que valoriza tanto o conhecimento técnico estrutural quanto a adaptabilidade prática.

A diversidade de abordagens curriculares demonstra que o ensino da Engenharia de Software ainda está em processo de transição entre a tradição e a modernidade. Algumas instituições preservam uma estrutura clássica por segurança didática ou por inércia institucional, enquanto outras avançam para modelos que integram práticas ágeis, ferramentas reais e metodologias ativas.

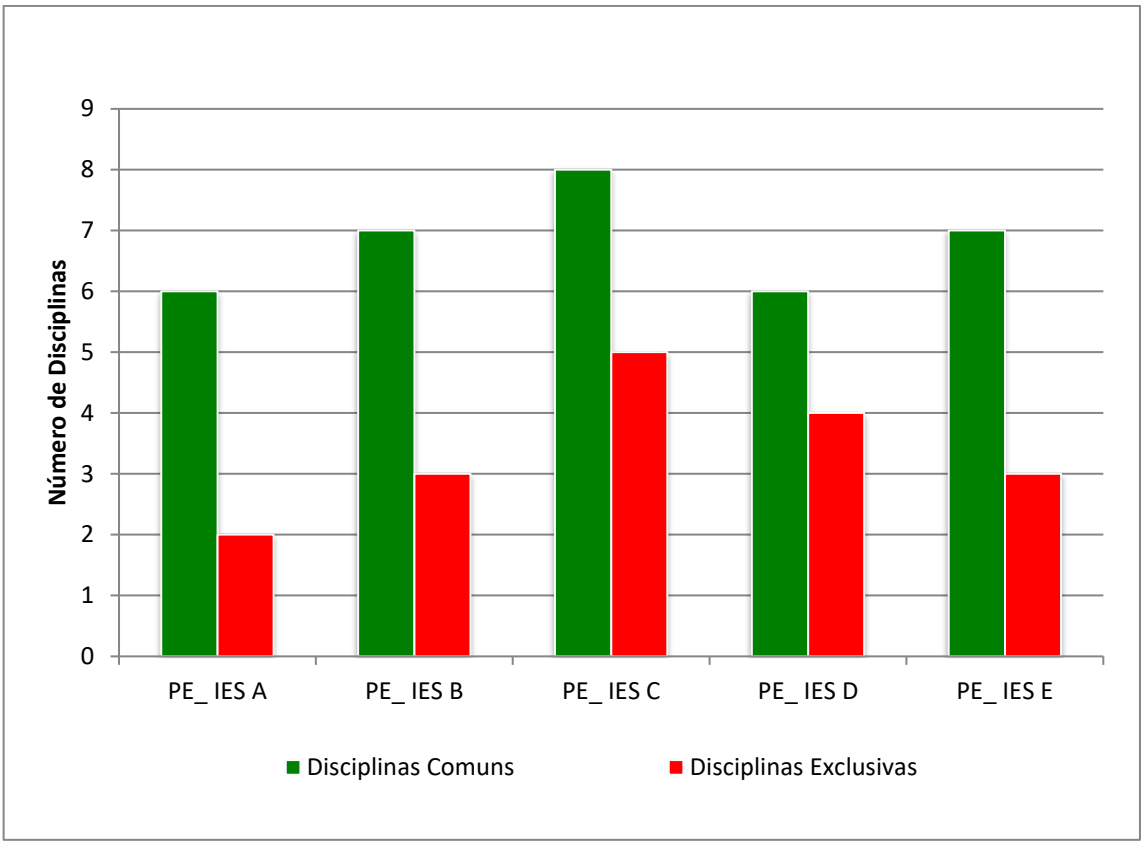
A compreensão dessa diversidade é essencial para a formulação de diretrizes curriculares mais flexíveis e coerentes com os desafios contemporâneos do desenvolvimento de software.

6.2.3 Disciplinas Comuns vs. Disciplinas Exclusivas

A análise documental realizada sobre os planos de ensino das disciplinas de Engenharia de Software nas cinco instituições investigadas permitiu identificar tanto áreas de convergência quanto trajetórias curriculares particulares. Um dos pontos observados diz respeito à quantidade de disciplinas comuns e exclusivas compartilhadas entre os cursos de Análise e Desenvolvimento de Sistemas dos PE_IES A, B, C, D e E como demonstra o gráfico 14.

Essa similaridade curricular pode ser atribuída a fatores como a consolidação histórica de determinados conteúdos na formação em Engenharia de Software e a influência direta das bibliografias consagradas adotadas pelas instituições (como por exemplo os autores Pressman, Sommerville.

Gráfico 14: Disciplinas comuns vs exclusivas



Fonte: Elaboração própria

As disciplinas comuns representam os conteúdos estruturantes, presentes em praticamente todos os currículos analisados. Dentre elas, destacam-se:

- Fundamentos de Engenharia de Software
- Modelagem com *UML*
- Testes de Software
- Qualidade de Software
- Engenharia de Requisitos

O número de disciplinas comuns varia entre 6 e 8 entre as instituições, sendo o PE_IES C a que apresenta o maior número de convergências com os demais cursos, indicando uma maior aderência às diretrizes curriculares amplamente aceitas na formação técnica da área.

As disciplinas exclusivas revelam traços distintivos de cada instituição. PE_IES como C e D apresentam o maior número de disciplinas próprias (5 e 4, respectivamente), o que demonstra uma tentativa de diferenciação curricular,

possivelmente orientada por demandas regionais, vocações institucionais ou estratégias pedagógicas específicas.

Essas disciplinas incluem tópicos como:

- Arquitetura de Software com Microsserviços (PE_IES C)
- Auditoria de Sistemas (PE_IES E)
- Projetos Integradores com Requisitos Reais de Cliente (PE_IES D)

Embora a presença de disciplinas exclusivas possa enriquecer o percurso formativo, ela também pode indicar uma falta de alinhamento com diretrizes comuns da área, o que levanta questionamentos sobre a equivalência formativa entre os cursos.

A relação entre disciplinas comuns e exclusivas é essencial para compreender a coesão e a diversidade curricular no ensino da Engenharia de Software. O gráfico 15 demonstra visualmente esse equilíbrio e sugere que:

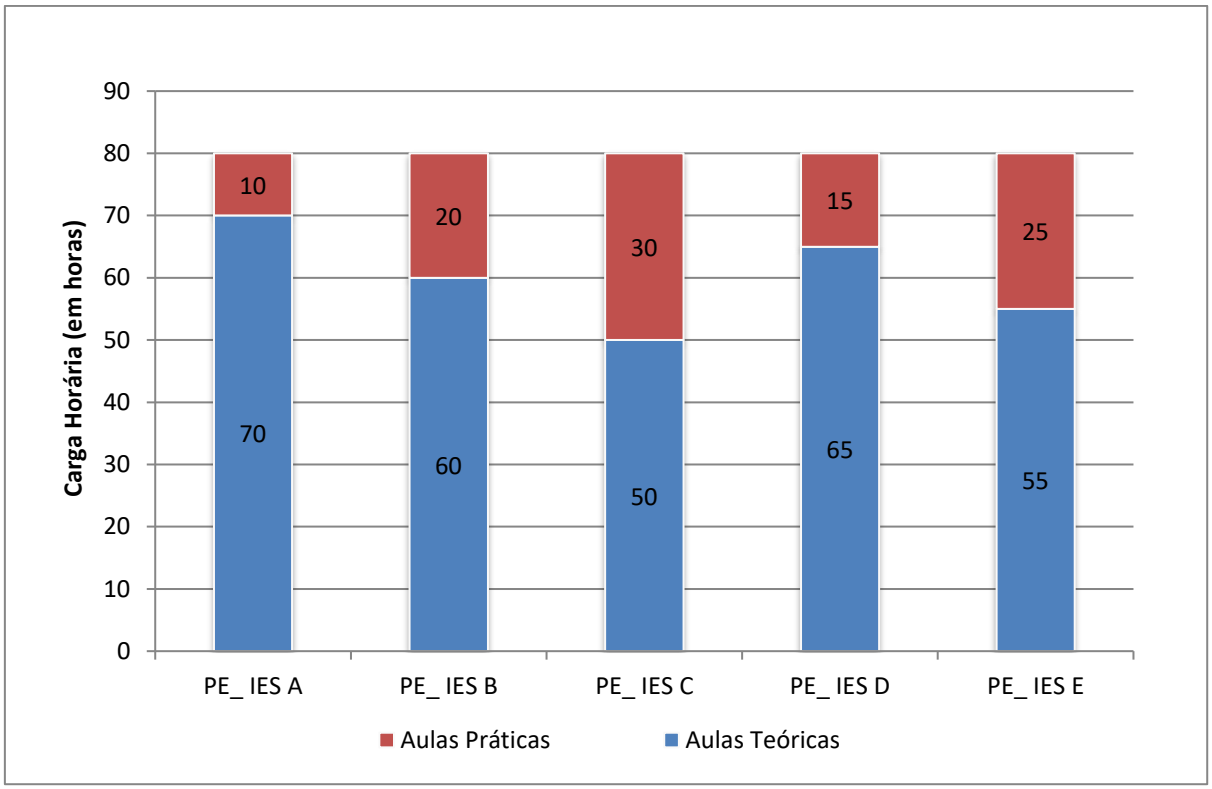
- Todas as instituições mantêm um núcleo curricular compartilhado, o que garante uma formação técnica mínima comum;
- Algumas instituições investem em disciplinas excluisivas para agregar valor à formação, mas devem equilibrar inovação e padronização;
- A análise reforça a necessidade de revisão contínua dos currículos, buscando não apenas atualização tecnológica, mas também equivalência e qualidade formativa interinstitucional.

6.3 Comparativo de Aulas Teóricas vs Práticas por Plano de Ensino

A análise da carga horária destinada às aulas teóricas e práticas nas disciplinas de Engenharia de Software revela diferentes enfoques pedagógicos entre as instituições avaliadas. Esses dados são cruciais para compreender o equilíbrio entre a formação conceitual e a aplicação prática das metodologias de desenvolvimento de software.

O gráfico 15, comparativo de aulas teóricas versus aulas práticas, demonstra que as aulas práticas descritas no plano de ensino são poucas em referência às aulas teóricas, isso causa um impacto negativo na instituição, visto que muitos estudantes, querem vivenciar de fato o que o mercado de trabalho utiliza no seu dia a dia.

Gráfico 15: Comparativo de Aulas Teóricas vs Práticas por Instituição



Fonte: Elaboração própria

Nos PE_IES A e D em que há predominância de aulas teóricas, essas apresentam o maior volume de carga horária teórica (70h e 65h, respectivamente), com apenas 10h e 15h dedicadas a atividades práticas. Esse dado reforça o perfil mais tradicional dessas instituições, que priorizam a construção de uma base teórica sólida, muitas vezes baseada em bibliografias clássicas e abordagens sequenciais. Embora essa base seja fundamental, a escassez de práticas pode comprometer a capacidade dos estudantes de aplicar os conhecimentos em cenários reais.

Os PE_IES B e E demonstram maior preocupação com a prática por meio de uma abordagem equilibrada, pois ambas apresentam proporções mais equilibradas, com 20h e 25h de aulas práticas, o que representa um esforço para integrar teoria e aplicação. Essa estratégia pode favorecer o aprendizado ativo, especialmente quando associada ao uso de projetos, estudos de caso e ferramentas de mercado.

O PE_IES C tem ênfase em práticas aplicadas e se destaca como a instituição com maior carga prática (30h) e menor carga teórica (50h), proporcionalmente. Isso revela uma proposta curricular voltada ao desenvolvimento de competências aplicadas, favorecendo o aprendizado por meio de experimentação, simulação de projetos reais e uso intensivo de ferramentas tecnológicas. Essa abordagem é

especialmente relevante no contexto atual do desenvolvimento ágil e colaborativo de software.

O comparativo revela uma tendência à prevalência do ensino teórico na maioria das instituições analisadas. No entanto, aquelas que conseguem integrar práticas de forma mais substancial oferecem um diferencial formativo mais alinhado às demandas do mercado.

O gráfico empilhado ajuda a visualizar esse equilíbrio (ou desequilíbrio) entre teoria e prática, servindo como subsídio para a reformulação curricular orientada por competências.

6.3.1 Análise do Ensino de Metodologias Ágeis

As metodologias ágeis tornaram-se um padrão amplamente adotado no desenvolvimento de software moderno, especialmente em ambientes que valorizam flexibilidade, entregas contínuas e colaboração multidisciplinar.

Dentre os *frameworks* mais utilizados estão *SCRUM, Kanban* e *Extreme Programming* (*XP*), todos orientados a ciclos iterativos, *feedback* constante e foco no cliente. Como descreve o gráfico 16, somente 2 IES utilizam todos os *frameworks* mais utilizados no mercado de trabalho e 2 IES não constam no plano de ensino registrado no PPC do curso.

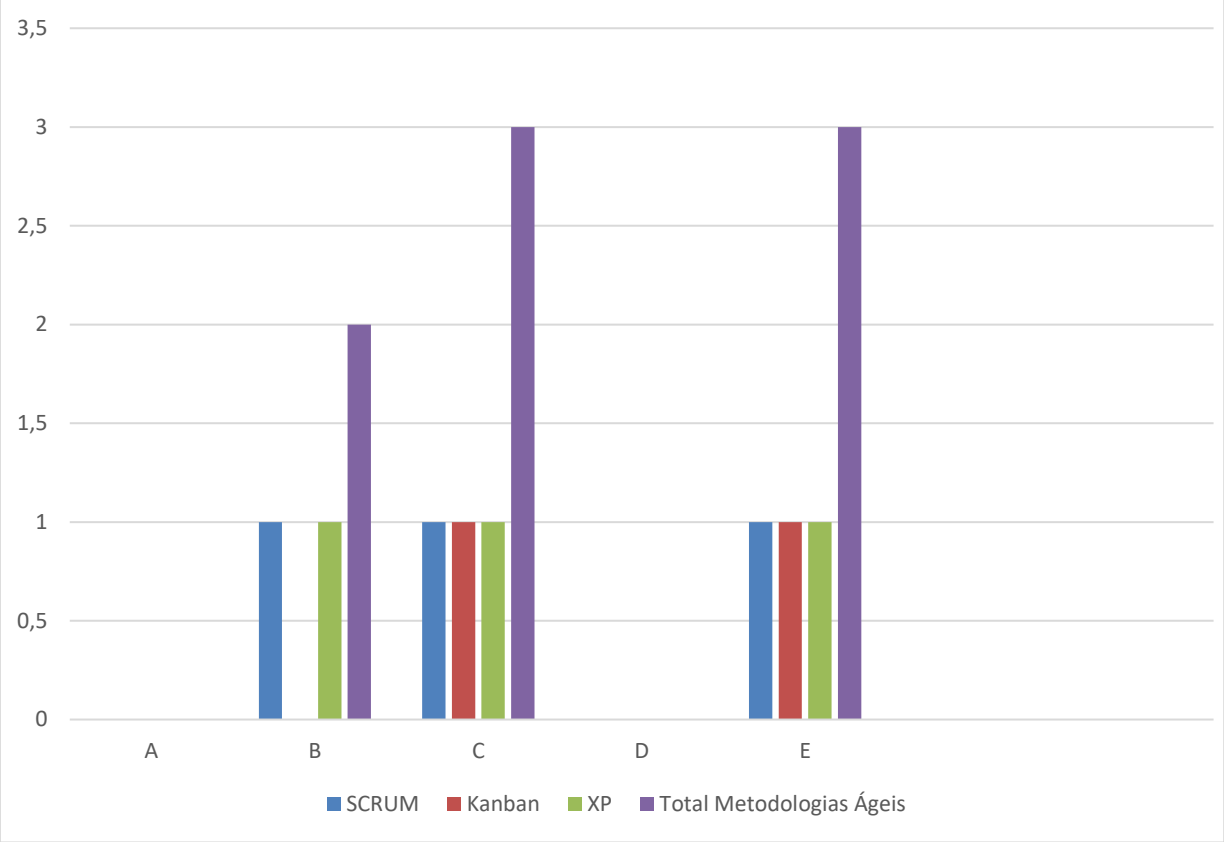
A avaliação dos planos de ensino das Instituições revelou que:

- PE_IES C e E são as únicas que abordam explicitamente as três metodologias ágeis analisadas (*SCRUM, Kanban, XP*), com aplicação prática por meio de ferramentas e simulações;

- PE_IES B contempla duas dessas metodologias (*SCRUM e XP),* demonstrando iniciativa de atualização curricular, ainda que sem detalhamento aprofundado;

- PE_IES A e D não apresentam qualquer menção a metodologias ágeis em seus planos de ensino, reforçando um foco exclusivamente tradicional.

Gráfico 16: Métodos Ágeis



Fonte: Elaboração própria

A adoção de metodologias ágeis no mercado de trabalho é quase ubíqua, principalmente em setores de tecnologia, *startups* e times de desenvolvimento de produtos digitais. A presença de *SCRUM, Kanban* e *XP* nos currículos é, portanto, um indicador importante de alinhamento com as práticas contemporâneas de gestão de projetos e Engenharia de Software.

PE_IES que incorporam essas metodologias estão não apenas transmitindo conhecimento técnico, mas também preparando os estudantes para atuarem de forma produtiva em times reais, usando *frameworks* que são frequentemente exigidos em entrevistas e certificações profissionais. Nessa parte da pesquisa, a análise, revela um cenário de desigualdade na incorporação de práticas ágeis no ensino superior. Enquanto algumas instituições já se adaptaram às exigências do mercado, outras permanecem ancoradas em modelos exclusivamente tradicionais.

A atualização curricular e a capacitação docente para trabalhar com metodologias ágeis são passos essenciais para garantir a formação de profissionais preparados para os desafios contemporâneos do desenvolvimento de software.

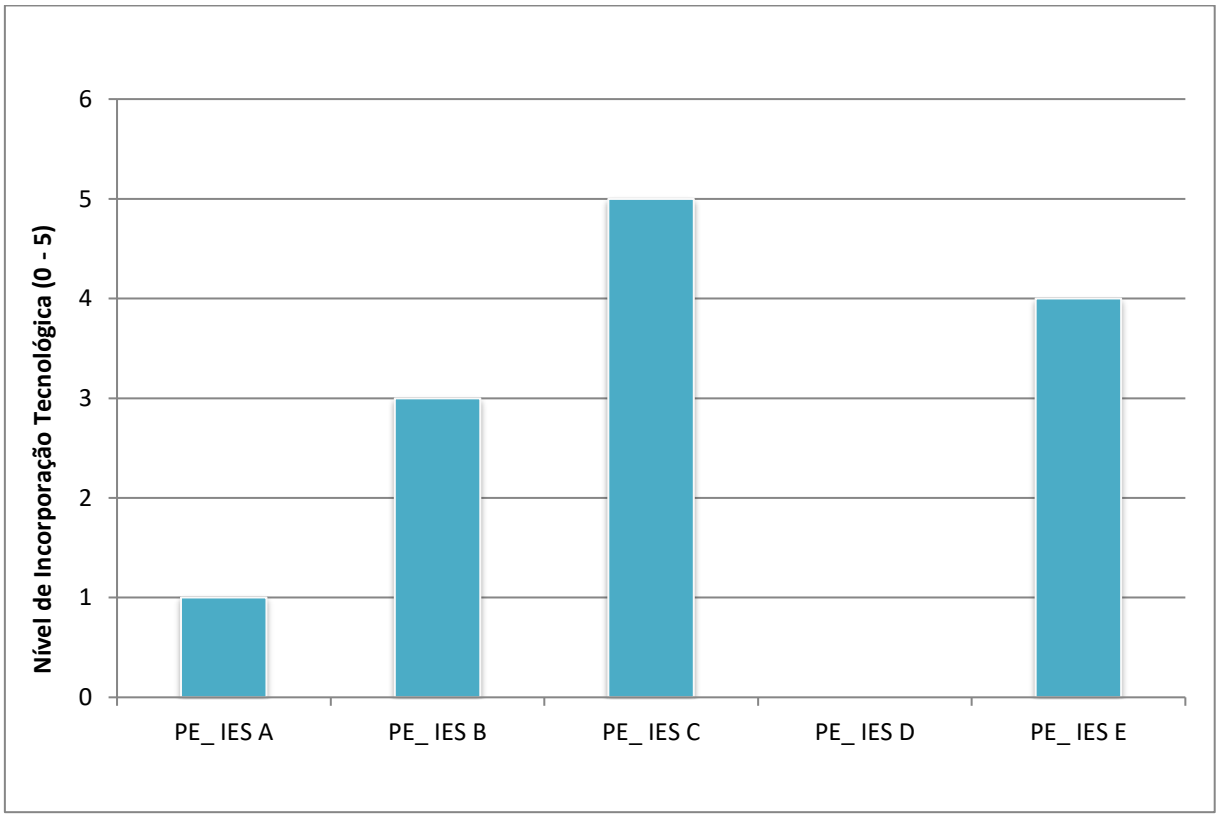
6.3.2 Impacto das Inovações Tecnológicas no Ensino

As instituições têm incorporado novas tecnologias no ensino de Engenharia de Software, conforme revelado pelos dados documentais e bibliográficos.

A análise documental revelou diferentes graus de integração de tecnologias educacionais e profissionais no ensino da disciplina de Engenharia de Software nas cinco instituições analisadas. Na sequência, apresenta-se uma leitura detalhada de como cada PE_IES incorpora ou menciona o uso de tecnologias e ferramentas, seja no conteúdo programático, nas atividades práticas ou nas referências bibliográficas.

No gráfico 17, pode-se verificar que 4 das 5 instituições estão no caminho da incorporação de tecnologias no ensino da disciplina de Engenharia de Software. A escala desse gráfico tem como base o quadro 7 e descreve o nível de incorporação tecnológica das IES que constam no plano de ensino.

Gráfico 17: Incorporação de Tecnologias no Ensino de Engenharia de Software



Fonte: Elaboração própria

No quadro 7, descreve-se o nível de incorporação tecnológica em que os PE_IES estão e o destaque de cada uma, que será demonstrado em uma análise individual.

Quadro 7: Incorporações Tecnológicas

IES	Nível de Incorporação Tecnológica	Pontuação (0-5)	Destaques
A	Baixo	1	Uso genérico de laboratório
B	Moderado	3	PIM e dinâmicas integradas
C	Alto	5	Ferramentas modernas e práticas de mercado
D	Muito Baixo	0	Ausência de menção a tecnologias
E	Moderado-Alto	4	Controle de versão e testes automatizados

Fonte: Elaboração própria

6.3.2.1 Sobre a PE_IES A

O nível de incorporação tecnológica foi detectado como baixo, pois não foi citada nenhuma ferramenta específica.

O PE_IES A é centrado em métodos tradicionais e referências clássicas, como Pressman e Sommerville. Apesar de mencionar o uso de laboratório com linguagem de programação instalada, não há indicação de quais ferramentas são utilizadas. Não há menção a versionamento, modelagem digital, ambientes colaborativos ou plataformas específicas.

O PE_IES A utiliza tecnologia de forma genérica, com ênfase teórica, sem destaque para ferramentas práticas de mercado.

6.3.2.2 Sobre a PE_IES B

O nível de incorporação tecnológica foi detectado como Moderado, por causa do uso genérico de laboratórios e ambiente de projeto multidisciplinar (PIM).

As estratégias incluem atividades em grupo e seminários, mas não há detalhamento sobre o uso de ferramentas específicas.

A presença do Projeto Integrador sugere que ferramentas de documentação e desenvolvimento podem ser utilizadas informalmente, embora não documentadas no plano de ensino.

Indícios de uso de tecnologia em contexto aplicado, porém sem clareza sobre as ferramentas ou metodologias digitais efetivamente adotadas.

6.3.2.3 Sobre a PE_IES C

O nível de incorporação tecnológica foi detectado como alto, visto que essa instituição, trabalha com as ferramentas como *Git e GitHub*, modelagem *UML, C4 Model, Draw.io*, Gerenciamento Ágil: *Trello, Jira, Azure DevOps* para uso em ambientes de desenvolvimento: *IntelliJ, VSCode*. Atua fortemente com a área de Testes e automação: *JUnit, Zephyr, Cucumber, TestLink*

O PE_IES C é o mais avançado tecnologicamente. Detalha a utilização de tecnologias reais do mercado em praticamente todos os tópicos da disciplina. Inclui práticas como versionamento, automação de testes, arquitetura de microsserviços e refatoração com ferramentas modernas.

O PE_IES C exemplifica excelente integração tecnológica, tanto no ensino como na prática profissional.

6.3.2.4 Sobre a PE_IES D

O nível de incorporação tecnológica foi detectado como muito baixo visto que essa instituição não teve nenhuma ferramenta especificada.

O PE_IES D tem foco teórico, com abordagem tradicional e referências datadas. Não há qualquer menção a ferramentas digitais, plataformas de desenvolvimento ou tecnologias emergentes.

Infelizmente a proposta do PE_IES D não reflete a evolução tecnológica do ensino de Engenharia de Software.

6.3.2.5 Sobre a PE_IES E

O nível de incorporação tecnológica foi detectado como Moderado a Alto visto que essa instituição, trabalha com as ferramentas como controle de versão (Ferramentas *CASE*, práticas com versionamento) a parte de testes (*TDD*, automação de testes) e se preocupa na parte de qualidade de software utilizando as ferramentas de controle e garantia de qualidade

O PE_IES E distribui conteúdos por unidade temática, e cita explicitamente o uso de ferramentas *CASE*, controle de versão e testes automatizados. No entanto, não especifica os nomes das ferramentas utilizadas.

Dessa forma, limita a avaliação mais precisa da atualização tecnológica. Integra tecnologias de forma conceitual e aplicada, mas carece de detalhamento técnico sobre ferramentas.

6.3.3 Inovações Pedagógicas e seus Resultados

Os resultados relacionados a inovações pedagógicas, no ensino superior, têm ganhado relevância à medida que a educação tradicional, centrada na transmissão unidirecional de conhecimento, mostra-se insuficiente para formar profissionais preparados para os desafios contemporâneos.

No contexto da Engenharia de Software, essa necessidade é ainda mais evidente, dada a natureza dinâmica e prática da área. Nesse cenário, metodologias como aprendizagem baseada em projetos (ABP), ensino híbrido, gamificação e o uso de Ambientes Virtuais de Aprendizagem (AVA) representam avanços significativos para aproximar o estudante da realidade do mercado e estimular competências como autonomia, colaboração e pensamento crítico.

No quadro 8, demonstram-se diferentes níveis de adoção dessas estratégias. O PE_IES C se destaca por integrar de forma consistente práticas pedagógicas inovadoras, como a utilização de ferramentas reais de mercado (*Git, Jira, Trello*), ambientes virtuais como o Canvas, além de dinâmicas em grupo e projetos colaborativos.

Quadro 8: Inovações Pedagógicas

IES	Ambientes Virtuais	Projetos Práticos	Nível de Inovação
A	Não	Não	Muito Baixo
B	Implícito (PIM)	Sim	Moderado
C	Sim (Canvas)	Extensivo	Muito Alto
D	Não	Não	Muito Baixo
E	Implícito	Sim	Moderado-Alto

Fonte: Elaboração própria

O PE_IES B também apresenta sinais positivos, ao incluir seminários, estudo de casos e projetos integradores (PIM) como instrumentos de aprendizagem ativo. Já as demais instituições (A, D e E), apesar de abordarem conteúdos atualizados em alguns casos, ainda não demonstram uso sistemático de práticas pedagógicas inovadoras, limitando-se a abordagens mais tradicionais, centradas em exposições teóricas.

Conforme destacam Bacich e Moran (2018), a inovação pedagógica não está apenas no uso da tecnologia, mas em colocar o aluno como protagonista do processo de aprendizagem. A adoção dessas práticas no ensino de Engenharia de Software é estratégica para reduzir a lacuna entre universidade e mercado, formando profissionais mais preparados técnica e socialmente para atuar em contextos complexos e em constante transformação.

A incorporação de inovações pedagógicas no ensino de Engenharia de Software tem mostrado um diferencial estratégico para o alinhamento entre a formação acadêmica e as exigências do mercado de trabalho. No entanto, a presença dessas inovações varia consideravelmente entre instituições, refletindo diferentes níveis de maturidade didático-metodológica.

A seguir, são apresentadas as evidências documentais sobre o uso de ambientes virtuais, projetos práticos e critérios avaliativos nas cinco instituições analisadas.

A análise tem como foco identificar práticas pedagógicas que promovam aprendizagem ativa, uso de tecnologia educacional e integração com problemas reais do setor.

A categorização é baseada nos planos de ensino oficiais das disciplinas de Engenharia de Software, fornecidos pelas instituições, e permite observar em que medida essas IES adotam abordagens contemporâneas e centradas no estudante.

PE_IES A

- **Ambientes Virtuais.** Não mencionados.
- **Projetos práticos.** Não especificados.
- **Avaliação.** Tradicional (duas provas e média de atividades).

O plano de ensino apresenta uma abordagem tradicional, sem sinalização de inovações metodológicas ou pedagógicas. Não há menção a plataformas digitais,

projetos interdisciplinares ou técnicas de ensino-aprendizagem ativas. A proposta se mantém centrada na exposição teórica e avaliações objetivas.

PE_IES B

- **Ambientes Virtuais.** Implicados no uso do Projeto Integrador Multidisciplinar (PIM).
- **Projetos práticos.** Sim, via PIM, com entregas parciais.
- **Avaliação.** Envolve participação, trabalhos e apresentações.

O PE_IES B adota práticas pedagógicas mais inovadoras, com destaque para o PIM, que propicia um espaço de aplicação prática e interdisciplinar. Embora não mencione ferramentas específicas de AVA, a estrutura sugere o uso de recursos complementares digitais e envolvimento dos estudantes na resolução de problemas reais.

PE_IES C

- **Ambientes Virtuais.** Uso explícito do *Canvas*.
- **Projetos práticos.** Extensivos — incluem modelagens, testes, versionamento, entregas contínuas.
- **Avaliação.** Diversificada e progressiva, com entregas semanais, protótipos e revisões práticas.

PE_IES C se destaca por uma abordagem pedagógica moderna e altamente inovadora. O uso contínuo do *Canvas* como plataforma central, a combinação de aulas teóricas e práticas com ferramentas reais de mercado e a utilização de técnicas como *TDD*, refatoração e *SCRUM* sugerem um ensino orientado por competências e centrado no estudante.

PE_IES D

- **Ambientes Virtuais.** Não mencionados.
- **Projetos práticos.** Não identificados.
- **Avaliação:** Foco em provas e exercícios teóricos.

O PE_IES D apresenta um plano de ensino estático e conteudista, baseado exclusivamente em aulas teóricas e avaliações tradicionais. Não há qualquer menção à inovação didática, ambientes digitais ou integração com práticas de mercado. O ensino é essencialmente expositivo e disciplinar.

PE_IES E

- **Ambientes Virtuais.** Não especificados, mas implicados indiretamente.
- **Projetos práticos.** Sim — unidades temáticas estruturadas com práticas de testes, manutenção e auditoria.
- **Avaliação.** Foco em aplicação prática dos conteúdos, incluindo exercícios, simulações e análises de ferramentas *CASE*.

O PE_IES E apresenta um nível intermediário de inovação pedagógica. Os conteúdos são organizados de forma prática e modular, com atividades teóricas e aplicadas. A falta de menção a um AVA específico reduz a visibilidade da estrutura digital de suporte, mas o foco em práticas aplicadas evidencia uma tentativa de aproximação com o mercado.

A comparação evidencia que apenas algumas instituições, como o PE_IES C, adotam de forma consistente práticas pedagógicas inovadoras, integrando tecnologias, dinâmicas colaborativas e ferramentas reais do mercado ao processo de ensino. Apresenta-se no quadro 9 a comprovação de práticas pedagógicas

Quadro 9: Comprovação De Práticas Pedagógicas De Aprendizagem Ativa

PE_IES	Ambientes Virtuais	Projetos Práticos	Metodologias Ativas	Avaliação
A	Não mencionados	Não especificados	Exercícios e extraclasse	Provas tradicionais
B	Indiretamente no PIM	Via PIM com entregas	Seminários, estudos de caso	Trabalhos e apresentações
C	Uso explícito do Canvas	Extensivos com ferramentas	Workshops, ABP, feedback	Diversificada com protótipos
D	Não mencionados	Não identificados	Ausentes	Provas e exercícios
E	Implicados indiretamente	Temas com testes e manutenção	Presentes sem detalhamento	Simulações e ferramentas CASE

Fonte: Elaboração própria

Essa abordagem sinaliza uma ruptura com o modelo conteudista tradicional e demonstra maior aderência às demandas de um ensino baseado em competências. Em contrapartida, instituições como as IES A e D ainda se mantêm ancoradas em metodologias convencionais, com pouca ou nenhuma evidência de práticas ativas ou uso de tecnologias educacionais estruturadas.

Esse panorama reforça a importância de políticas institucionais que incentivem a modernização curricular, a capacitação docente contínua e a adoção de estratégias que valorizem o protagonismo discente.

A transição para modelos educacionais mais interativos e conectados ao mundo do trabalho é essencial para garantir que o ensino de Engenharia de Software, nas instituições analisadas, seja não apenas tecnicamente sólido, mas também socialmente relevante e profissionalmente eficaz.

6.4 Proposição e Validação Teórica de Estratégias para o Plano de Ensino de Engenharia de Software

A formulação das estratégias de melhoria para o plano ensino de Engenharia de Software nesta pesquisa foi fundamentada em dois eixos principais: a análise documental de planos de ensino das cinco IES particulares de Sorocaba/SP e o embasamento teórico proveniente da literatura especializada em educação, inovação pedagógica e metodologias de desenvolvimento de software.

A literatura revisada destaca que a integração entre práticas tradicionais e metodologias ágeis é essencial para formar profissionais versáteis, capazes de atuar em ambientes corporativos diversos e em constante transformação.

Pressman e Maxim (2016) reforçam que a Engenharia de Software moderna exige não apenas domínio técnico, mas também adaptabilidade, colaboração e fluência em ferramentas práticas.

Nesse sentido, estratégias como a revisão de planos de ensino, a inclusão de módulos práticos colaborativos, o uso estruturado de ambientes virtuais de aprendizagem, a capacitação docente e a criação de parcerias com empresas locais encontram respaldo direto na literatura e respondem aos principais desafios observados nos planos curriculares analisados.

Para Bacich e Moran (2018) defendem que o protagonismo discente, o uso de projetos reais, a aprendizagem baseada em problemas (PBL) e o ensino híbrido são

estratégias eficazes para conectar a teoria à prática, promover o engajamento e desenvolver competências críticas.

As propostas delineadas nesta pesquisa se alinham a essa perspectiva, buscando reestruturar o plano de ensino de Engenharia de Software para que ele não apenas transmita conhecimento técnico, mas também forme estudantes críticos, autônomos e preparados para os desafios da prática profissional.

Essa abordagem permitiu uma validação teórica consistente, assegurando que as propostas apresentadas não apenas respondessem às lacunas identificadas nos currículos, mas também estivessem em consonância com os princípios pedagógicos contemporâneos e as exigências do mercado de trabalho atual.

Do ponto de vista da coerência interna, a proposta foi construída com base nas evidências empíricas extraídas dos documentos institucionais e, posteriormente, confrontada com diretrizes pedagógicas estabelecidas pelas Diretrizes Curriculares Nacionais (DCN) e pelas recomendações de autores de referência na área. Isso reforça a viabilidade de aplicação das estratégias sugeridas dentro do contexto educacional das instituições analisadas. Dessa forma, a validação teórica das propostas não apenas legitima as sugestões formuladas, como também oferece um arcabouço sólido para sua implementação e adaptação às realidades institucionais.

A análise documental realizada nos planos de ensino das cinco instituições de ensino superior (PE_IES A, B, C, D e E) identificou lacunas e boas práticas no da disciplina de Engenharia de Software. A partir dessas evidências, elaborou-se um conjunto de estratégias concretas com o objetivo de alinhar a formação acadêmica às competências exigidas pelo mercado de trabalho contemporâneo.

Tais estratégias foram concebidas considerando-se a necessidade de modernização curricular, metodológica e tecnológica, e incluem:

- Revisão das ementas para incorporar práticas modernas de desenvolvimento de software;
- Inclusão de módulos específicos sobre metodologias ágeis, como *SCRUM, Kanban* e *Extreme Programming* (*XP*);
- Ampliação e implementação de projetos práticos colaborativos, orientados por situações reais e com uso de ferramentas profissionais.
- Criação de parcerias com empresas do mercado de trabalho, levando-as para dentro da universidade por meio de parcerias.

Além da proposição, realizou-se uma validação teórica das estratégias sugeridas, fundamentando-as por meio da literatura científica revisada e da compatibilidade com as DCN para os cursos de computação (Brasil, 2018).

Essa validação teve como objetivo garantir que as propostas fossem pedagogicamente viáveis, tecnicamente relevantes e coerentes com o perfil de egresso desejado.

6.5 Proposição de Estratégias

As proposições de validação nos levam para a recomendações estratégicas, que refletiram uma proposta de transformação curricular sustentada em fundamentos teóricos sólidos e práticas educacionais inovadoras. Ao promover essa articulação entre ensino e mercado, acredita-se que as instituições analisadas estarão mais bem preparadas para formar profissionais mais competentes, colaborativos e tecnicamente atualizados, conforme exigem os desafios contemporâneos da Engenharia de Software.

A proposição dessas estratégias visa a melhorar o alinhamento entre a formação acadêmica e as demandas do mercado, preparando melhor os estudantes para a prática profissional.

6.5.1 Propostas de Melhoria

A análise documental comparativa dos planos de ensino da disciplina de Engenharia de Software dos PE_IES A, B, C, D e E revelou diferentes níveis de aderência às demandas atuais do mercado de desenvolvimento de software. A partir desse estudo, foram identificadas algumas críticas e boas práticas que fundamentam a formulação de estratégias concretas para aprimorar o ensino da disciplina.

6.5.2 Revisão de Planos de Ensino e Atualização Curricular

A análise comparativa dos planos de ensino da disciplina de Engenharia de Software evidenciou discrepâncias significativas quanto ao grau de atualização dos conteúdos oferecidos pelas instituições.

Os PE_IES A e D mantêm estruturas curriculares fortemente ancoradas em paradigmas tradicionais, com pouca ou nenhuma menção a metodologias ágeis, ferramentas contemporâneas de desenvolvimento ou práticas alinhadas às demandas atuais da indústria de software.

Essa configuração indica uma defasagem preocupante entre o conteúdo ofertado e as exigências do mercado de trabalho.

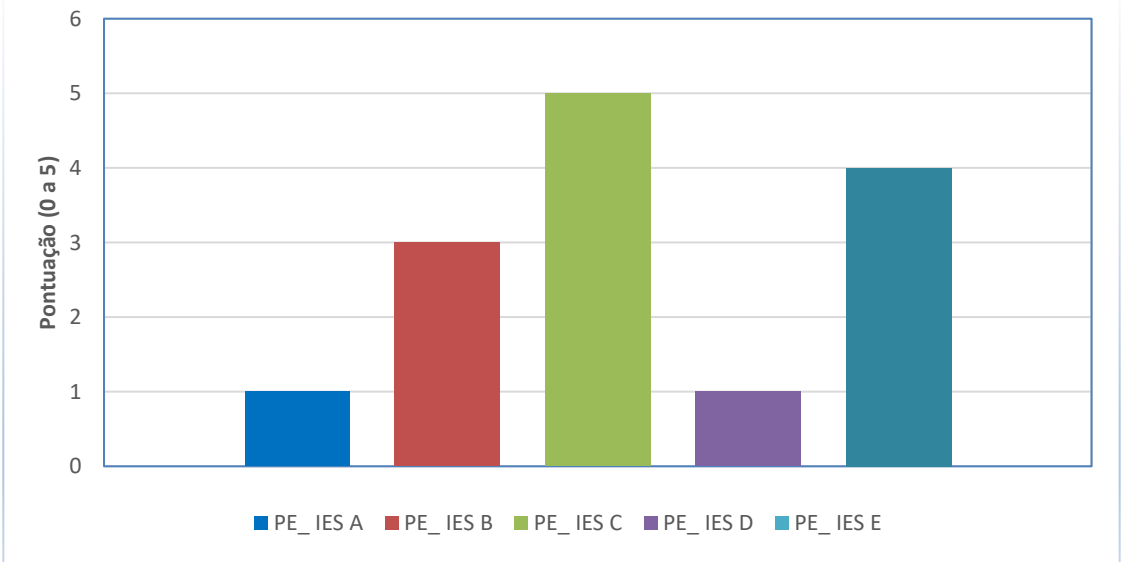
Para os autores, Pressman; Maxim (2016), o ensino de Engenharia de Software precisa ser adaptável e incorporar práticas que reflitam o desenvolvimento real de software na indústria.

Para Sommerville (2011), a evolução dos sistemas computacionais impõe ao ensino superior a necessidade de revisão constante de seus currículos, a fim de acompanhar o dinamismo do setor produtivo.

Conforme ilustrado no gráfico 18, o PE_IES C apresenta um plano de ensino mais moderno e abrangente, incorporando de forma consistente práticas ágeis, tecnologias emergentes e estratégias pedagógicas ativas.

Dentre as instituições analisadas, é a que menos demanda revisão imediata, servindo como referência positiva no processo de atualização curricular. O Eixo Y representa o nível de necessidade de revisão dos planos de ensino. Os valores foram atribuídos com base em uma escala de 0 a 5, em que: 5 = ausência total de revisão registrada; 4 e 3 = baixa necessidade de revisão; 2 e 1 = revisão moderada necessária e 0 = forte necessidade de revisão.

Gráfico 18: Revisão de Planos de Ensino



Fonte: Elaboração própria

Uma proposta para esse item, seria atualizar os planos de ensino para incluir tópicos como:

- Metodologias ágeis (*SCRUM, Kanban*, *XP*)
- Desenvolvimento orientado a testes (*TDD*)
- Versionamento com *Git/GitHub*
- Integração contínua *(CI/CD)* e *DevOps*

Com a inclusão das propostas é esperado gerar um impacto como a atualização técnica da formação e aproximação com as exigências do setor tecnológico atual.

6.5.3 Inclusão de Módulos Práticos Colaborativos

No contexto atual da formação em Engenharia de Software, observa-se uma crescente necessidade de alinhar o ensino acadêmico às demandas concretas do mercado de trabalho.

Dentre os diversos elementos que compõem essa modernização curricular, a presença de módulos práticos colaborativos destaca-se como estratégia fundamental para o desenvolvimento de competências técnicas e comportamentais. Ao possibilitar que os estudantes atuem em equipes, solucionem problemas reais e utilizem ferramentas amplamente adotadas na indústria, esses módulos promovem uma aprendizagem ativa e significativa, favorecendo a empregabilidade e a autonomia dos egressos.

Essa constatação emerge com clareza a partir da análise comparativa realizada entre as cinco instituições de ensino investigadas. Enquanto algumas IES demonstram avanços significativos nesse aspecto, outras ainda mantêm uma abordagem predominantemente teórica, limitando o potencial formativo de seus currículos.

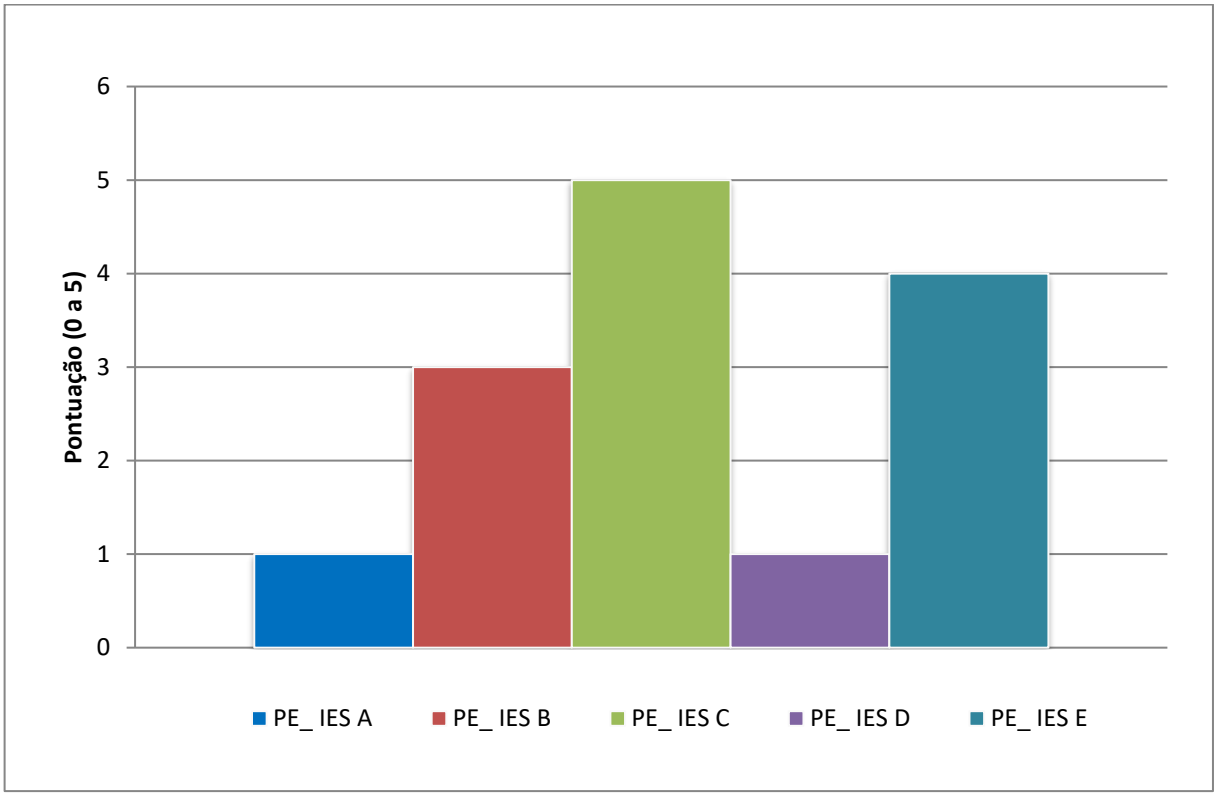
Para os autores, Moura; Oliveira (2020), os projetos interdisciplinares não apenas promovem a aprendizagem ativa, mas também desenvolvem habilidades socioemocionais e competências colaborativas.

Discutiu-se a relevância da inclusão de módulos práticos colaborativos e a forma como essa estratégia contribui para o aprimoramento do ensino de Engenharia de Software.

Os planos de ensino que mais se destacaram, como o PE_IES C, aplicam projetos reais e ferramentas de mercado no processo de ensino, enquanto outras (como A e D) oferecem carga horária prática mínima conforme demonstrado no gráfico 19.

Schach (2010) relata que, o desenvolvimento de software é essencialmente uma atividade prática e, por isso, o ensino deve fornecer oportunidades contínuas para que os estudantes construam software de forma Incremental. O eixo Y representa o grau de presença (0 a 5) de projetos práticos colaborativos nos currículos. A escala refere-se 0: ausência total de projetos práticos colaborativos; 1 e 2: presença mínima ou pontual de atividades práticas; 3 e 4: presença intermediária, indicando aplicação moderada de projetos e 5: presença forte e estruturada de projetos práticos colaborativos, com alinhamento a práticas reais do mercado.

Gráfico 19: Projetos Práticos Colaborativos



Fonte: Elaboração própria

Uma proposta para esse item é introduzir módulos baseados em projetos integradores reais como:

- Equipes multidisciplinares simulando squads ágeis

- *Sprints* com entregas parciais
- Revisões por pares e demonstrações (*review*)
- Ferramentas sugeridas: *Jira, Trello, GitHub, Figma, Jenkins, Docker*.

Com a inclusão das propostas é esperado gerar um impacto com o desenvolvimento de competências práticas, colaboração, autonomia e preparo para ambientes de trabalho reais.

6.5.4 Ampliação do Uso de Ambientes Virtuais e Ferramentas Educacionais

A análise dos planos de ensino revelou uma lacuna importante no uso estruturado de Ambientes Virtuais de Aprendizagem (AVA) e ferramentas tecnológicas de apoio ao ensino.

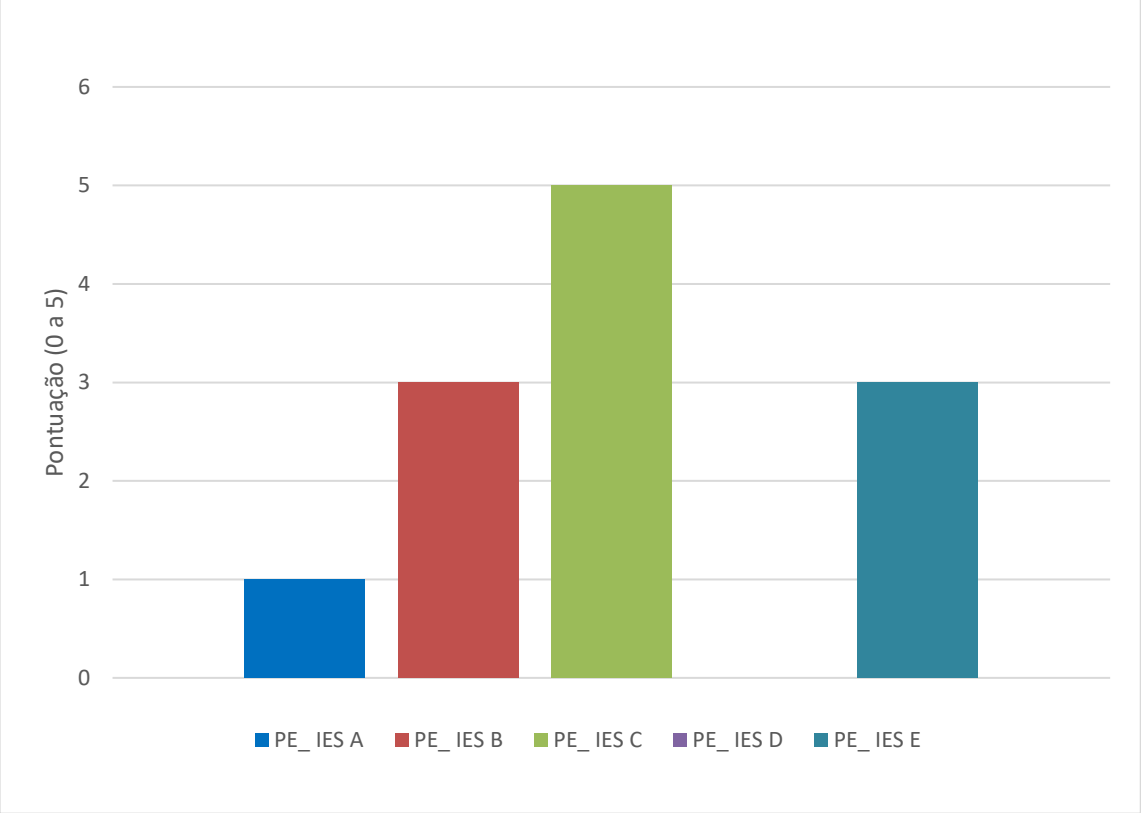
Entre as instituições analisadas, apenas o PE_IES C menciona explicitamente a utilização de um AVA institucional *(Canvas),* com integração clara às atividades pedagógicas da disciplina. Os demais PE_IES, embora possam utilizar alguma plataforma, não apresentam essa informação de forma clara ou estruturada, tampouco detalham abordagens híbridas ou recursos complementares utilizados no processo de ensino-aprendizagem.

Segundo Bates (2019), o uso de tecnologias digitais no ensino superior deve ser centrado na promoção da autonomia e da personalização da aprendizagem.

Conforme ilustrado no gráfico 20, a carência de menções a AVA e ferramentas educacionais modernas reflete a necessidade de revisão das práticas pedagógicas para acompanhar os avanços tecnológicos e as exigências de um público discente cada vez mais digital.

Eixo Y (vertical): Pontuação (0 a 5) para nível de adoção/integração de AVA e ferramentas escala ordinal. 0: Não há menção a AVA/ferramentas; 1: Menção genérica; uso eventual, sem detalhe; 2: Uso pontual, sem integração às atividades/avaliações; 3: Uso regular e parcialmente integrado (materiais/entregas), porém sem desenho pedagógico consistente; 4: Integração clara com a disciplina (materiais, tarefas, fóruns, *feedback*), com diretrizes de uso; e 5: Integração plena e planejada (ex.: *Canvas* com módulos, rubricas, *quizzes*, *fóruns*, LTI/integrações como *GitHub, Figma, CI/CD, analytics*).

Gráfico 20: Uso de Ambientes Virtuais



Fonte: Elaboração própria

Uma proposta para esse item do estudo é

- Adotar AVA robustos como *Moodle, Canvas ou Google Classroom*.
- Integrar repositórios de código *(Git),* ambientes de simulação (*Glitch, Replit*) e recursos audiovisuais
- Oferecer *feedbacks* automatizados em tarefas e quizzes adaptativos

Segundo Anderson (2008), o ensino online não deve ser apenas uma réplica da sala de aula física, mas uma experiência interativa e significativa.

Com a inclusão das propostas, é esperado gerar um impacto no aumento da flexibilidade no ensino, aprendizagem contínua e melhor acompanhamento do desempenho discente.

6.5.5 Criação de Parcerias com Empresas e Startups Locais

A ausência de contextualização prática no ensino de Engenharia de Software foi uma das principais limitações observadas nos planos de ensino analisados. Essa

lacuna compromete o desenvolvimento de competências aplicadas e a capacidade dos estudantes de atuarem com segurança em ambientes profissionais reais. Nesse sentido, a criação de parcerias institucionais com empresas e startups locais surge como uma estratégia eficaz para aproximar o ambiente acadêmico da realidade do mercado de trabalho.

Para Rezende (2005), o aprendizado significativo na área de tecnologia depende do enfrentamento de problemas reais, com demandas, prazos e interações reais.

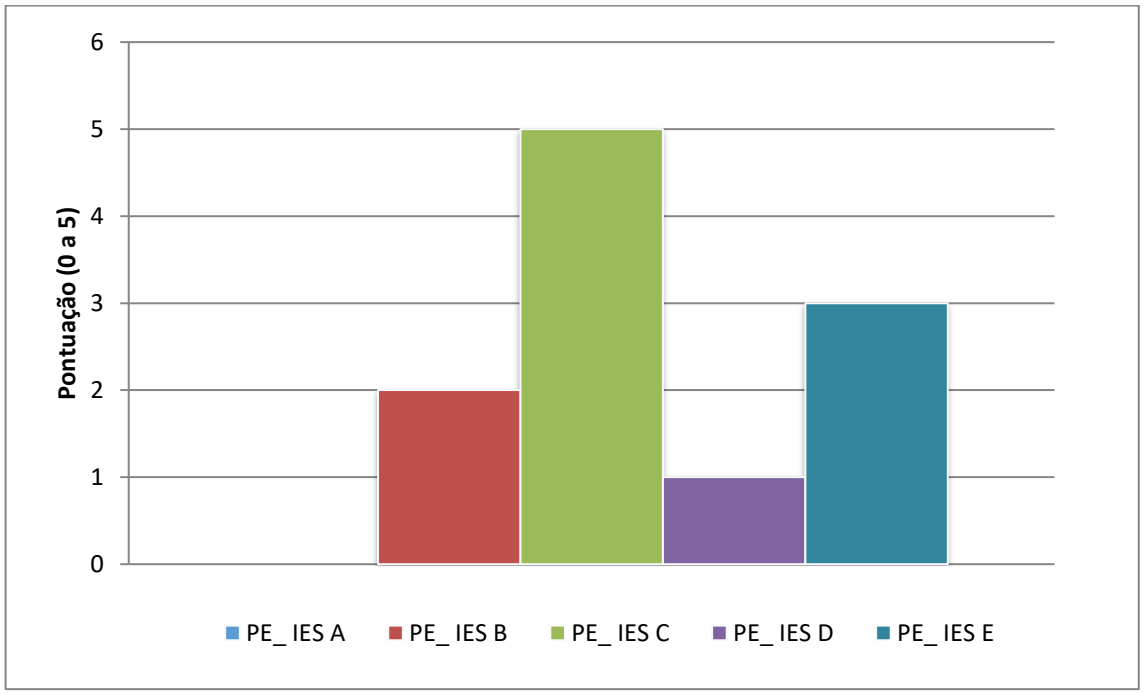
As propostas de intervenção nesse eixo incluem, em estabelecer convênios com empresas de tecnologia da região, permitindo que projetos de desenvolvimento de software façam parte efetiva da carga horária da disciplina, sendo acompanhados por docentes e profissionais do mercado e buscar incentivar a participação de estudantes em *hackathons*, programas de estágio supervisionado, laboratórios de inovação e residências tecnológicas, promovendo desafios reais e colaboração interinstitucional.

Com a implementação dessas propostas, espera-se uma maior imersão dos estudantes em ambientes reais de trabalho, o que contribuirá significativamente para o desenvolvimento de habilidades práticas e *soft skills* essenciais, como trabalho em equipe, resolução de problemas, comunicação e adaptabilidade. Além disso, tais experiências tendem a elevar os índices de empregabilidade, tornando os egressos mais competitivos e preparados para os desafios da indústria de software contemporânea

Conforme indicado no gráfico 21, a maior parte das instituições avaliadas não apresenta, de forma explícita, mecanismos de integração com o setor produtivo, como convênios, programas de estágio supervisionado ou participação em projetos reais. Tal ausência evidencia a necessidade de ações estruturadas que promovam a inserção dos estudantes em contextos de desenvolvimento de software sob demanda real, contribuindo para a formação de competências técnicas e comportamentais. A representação da Pontuação (0 a 5) atribuída à presença de parcerias institucionais com empresas nos planos de ensino fica: como escala ordinal, baseada na mesma lógica de Likert usada em outros gráficos que para 0: ausência total de menção a parcerias; 1 e 2: presença muito fraca (menções superficiais ou pouco estruturadas); 3 e 4: presença moderada/consistente (alguns convênios, projetos ou estágios); e 5: presença forte e estruturada (parcerias formais, convênios regulares, integração

efetiva ao currículo). O gráfico vai até a escala 6 apenas por estética, mas o máximo de valor real é 5.

Gráfico 21: Parcerias com Empresas



Fonte: Elaboração própria

Segundo Etzkowitz; Leydesdorff, (2000), as parcerias universidade-empresa ampliam as possibilidades de inovação, ao mesmo tempo em que inserem os estudantes em ecossistemas reais de produção.

Com base em uma análise documental comparativa, foi possível evidenciar as lacunas existentes nos currículos de algumas universidades e, ao mesmo tempo, reconhecer boas práticas que já vêm sendo implementadas em outras, como é o caso da IES C.

A revisão de ementas, a inclusão de módulos práticos colaborativos, o uso estruturado de ambientes virtuais de aprendizagem, a capacitação docente contínua e a aproximação com o setor produtivo compõem um conjunto articulado de ações que, se implementadas de forma integrada, têm o potencial de transformar a experiência formativa dos discentes, promovendo uma educação mais aplicada, dinâmica e aderente às exigências contemporâneas da Engenharia de Software.

Mais do que apenas atualizar conteúdos, tais propostas apontam para uma reconfiguração metodológica e institucional do ensino, em que teoria e prática se

complementam, e a sala de aula se expande para os contextos reais de desenvolvimento. Acredita-se que esse movimento é essencial para reduzir a distância entre a academia e o mercado, preparar os estudantes para os desafios da atuação profissional e fomentar o desenvolvimento de competências técnicas e socioemocionais.

Nesse sentido, o sucesso dessas mudanças depende não apenas da revisão curricular em si, mas do compromisso institucional com a inovação pedagógica e a valorização do corpo docente. Como aponta Valente (2014), a integração das tecnologias e metodologias ativas ao ensino requer mais do que infraestrutura, exige a formação continuada dos professores e o reposicionamento de seus papéis como mediadores do conhecimento

Portanto, espera-se que estas propostas sirvam como base para reflexões institucionais e políticas educacionais mais amplas, estimulando ações concretas e contínuas de aprimoramento curricular nos cursos de tecnologia e computação. Ao alinhar as diretrizes pedagógicas às práticas emergentes da indústria de software, avança-se não apenas na qualidade da formação, mas também na relevância social do ensino superior.

6.6 Alinhamento com as Demandas do Mercado de Trabalho

O setor de tecnologia da informação caracteriza-se por sua constante transformação e pela crescente complexidade dos ambientes de desenvolvimento. Nesse cenário, o alinhamento entre a formação acadêmica e as demandas do mercado de trabalho é considerado um fator essencial para a empregabilidade dos egressos e para a efetividade do processo educacional.

As empresas buscam profissionais que, além de sólida base conceitual, possuam experiência prática, domínio de ferramentas contemporâneas e capacidade de adaptação a metodologias ágeis e colaborativas.

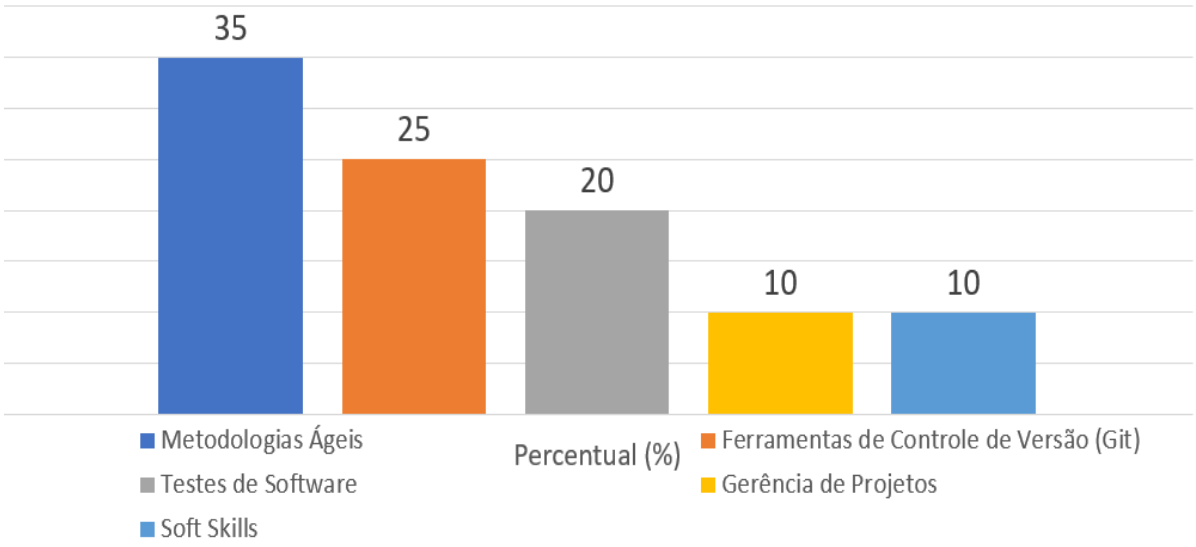
A percepção do mercado sobre a formação dos graduados em cursos de Computação e Engenharia de Software, reconhece-se o esforço das instituições em prover fundamentos teóricos, mas aponta-se como insuficiente a formação prática e o domínio de competências aplicadas.

Empresas frequentemente relatam dificuldades em encontrar profissionais que já tenham familiaridade com processos ágeis, versionamento, integração contínua,

testes automatizados e *frameworks* modernos como *React, Docker ou Spring*, competências raramente enfatizadas nos currículos mais tradicionais.

O gráfico 22 demonstra essa falta de habilidades pelos egressos quando vão para o mercado de trabalho.

Gráfico 22: Habilidade mais negligenciadas pelos egressos



Fonte: Elaboração própria

A análise realizada nesta pesquisa, com base nos documentos curriculares das cinco universidades particulares de Sorocaba/SP, confirma essa percepção. PE_IES como A e D ainda apresentam currículos predominantemente teóricos, com forte ênfase em modelos tradicionais e ausência de práticas integradas ao mercado.

Esse desalinhamento compromete a preparação dos estudantes para os desafios reais enfrentados em ambientes de desenvolvimento contemporâneos. Por outro lado, instituições como o PE_IES C evidenciam maior sintonia com o setor produtivo, ao incorporarem ferramentas, projetos práticos e metodologias ágeis no plano de ensino, promovendo uma formação mais próxima da realidade profissional.

A aceitação dos graduados no mercado tende a ser diretamente proporcional ao nível de exposição prática que tiveram durante a formação.

Graduados oriundos de cursos que promovem projetos reais, estágios supervisionados, uso de AVA e aprendizagem baseada em projetos (ABPj) demonstram maior autonomia, capacidade de trabalho em equipe e habilidade de adaptação a novos contextos tecnológicos. Isso os torna mais competitivos em processos seletivos e mais eficazes em sua inserção profissional.

Assim, os resultados indicam que o alinhamento entre ensino e mercado ainda é desigual entre as instituições analisadas. A existência de boas práticas em algumas universidades mostra que é possível promover uma formação técnica de qualidade com base em metodologias ativas, integração com o setor produtivo e currículo atualizado. Entretanto, a ausência dessas práticas em parte significativa das instituições evidencia a necessidade de reformas estruturais e contínuas para que os cursos atendam efetivamente às expectativas e necessidades da indústria de software.

O *gap* entre as universidades e o mercado de trabalho na área de TI é uma realidade recorrente, e ele pode ser explicado por uma combinação de fatores estruturais, pedagógicos e culturais que são:

- **Ritmo de evolução tecnológica versus lentidão institucional**

O setor de TI evolui em uma velocidade muito maior do que os processos institucionais das universidades. Tecnologias, linguagens de programação, *frameworks* e metodologias mudam rapidamente no mercado.

Já as grades curriculares acadêmicas exigem tempo para serem revisadas, passando por trâmites burocráticos, aprovação em colegiados e autorização dos órgãos reguladores. Isso gera desatualização curricular e atrasos na incorporação de tendências como *DevOps, DataOps,* segurança no ciclo de vida, *cloud computing* e metodologias ágeis.

- **Falta de parcerias efetivas com empresas e ambientes reais**

A ausência de parcerias estruturadas com empresas, *startups e hubs* de inovação contribui diretamente para o distanciamento entre ensino e prática. Sem a vivência em projetos reais, estágios supervisionados relevantes ou laboratórios de inovação integrados com o setor produtivo, o aluno tem pouco contato com problemas reais, ferramentas profissionais e metodologias colaborativas, tornando sua transição para o mercado mais difícil.

- **Foco excessivo na formação conteudista e avaliação tradicional**

Muitos cursos ainda operam com uma estrutura pedagógica centrada em conteúdos e não em competências. Há pouca valorização da aprendizagem baseada em projetos, da interdisciplinaridade, da prototipação rápida e da avaliação por entregas práticas. Isso gera um perfil de egresso que, embora tenha boa base teórica, enfrenta dificuldades em aplicar esse conhecimento em contextos reais em que a adaptabilidade, o trabalho em equipe e o domínio de ferramentas são exigidos desde o primeiro dia.

- **Desconexão das Diretrizes Curriculares**

Embora as Diretrizes Curriculares Nacionais (DCN) da área de Computação incentivem a atualização, interdisciplinaridade e inserção em contextos profissionais, muitas instituições não conseguem operacionalizar essas diretrizes, seja por falta de estrutura, cultura institucional ou resistência à mudança.

O *gap* entre universidade e mercado não é uma falha exclusiva da área de TI, mas ele é mais evidente nesse campo devido à aceleração tecnológica e às demandas práticas do setor. Superar esse distanciamento exige esforço conjunto: atualização curricular ágil, criação de ecossistemas de inovação com o setor produtivo e maior foco em metodologias de ensino baseadas em problemas reais.

6.7 Proposta do Plano de Ensino Futura

A presente pesquisa teve como objetivo investigar, por meio da análise documental, como o plano de ensino da disciplina de Engenharia de Software vem sendo estruturado em cinco instituições particulares de ensino superior da região de Sorocaba/SP.

A partir da leitura crítica dos planos de ensino, ementas e projetos pedagógicos, foi possível identificar tendências, lacunas e boas práticas relacionadas à abordagem dos modelos tradicionais e metodologias ágeis no currículo acadêmico.

Os dados evidenciaram uma heterogeneidade significativa entre os planos de ensino das instituições.

Enquanto algumas ainda priorizam uma abordagem centrada na teoria e em modelos clássicos como a Cascata e o Espiral, outras já demonstram uma evolução curricular que inclui práticas ágeis como *SCRUM, Kanban e TDD*, bem como o uso de ferramentas de versionamento *(Git*), plataformas de desenvolvimento colaborativo (*GitHub, Jira)* e projetos interdisciplinares e aprendizagem baseada em problemas (PBL).

A constatação reforça a importância da atualização curricular contínua no ensino superior de computação. Como afirmam Sommerville (2011) e Pressman e Maxim (2016), a Engenharia de Software é um campo dinâmico e altamente impactado pela evolução tecnológica, o que exige dos cursos superiores uma postura responsiva e inovadora.

Pressman e Maxim, (2016), o ensino de Engenharia de Software precisa ser adaptável e incorporar práticas que reflitam o desenvolvimento real de software na indústria.

6.7.1 Plano de Ensino - Engenharia de Software Híbrida

A partir da análise comparativa e da proposição de uma ementa integradora, buscou-se delinear uma estratégia curricular que contemplasse os modelos tradicionais e ágeis, articulando teoria e prática com uso de tecnologias e metodologias ativas, respeitando as diretrizes curriculares do MEC. Uma abordagem didático-metodológica alinhada às demandas do mercado, ao mesmo tempo em que respeita os princípios pedagógicos das DCN, promovendo o desenvolvimento de competências técnicas e socioemocionais.

Como resultado, o quadro 10 apresenta nossa proposta de plano de ensino de Engenharia de Software, um modelo que pode servir de referência para outras instituições.

Quadro 10: Proposta de plano de ensino de Engenharia de Software

PLANO DE ENSINO - ENGENHARIA DE SOFTWARE HÍBRIDA
Curso: Análise e Desenvolvimento de Sistemas
Carga horária total: 80 horas
Modalidade: Presencial com apoio em AVA (Ambiente Virtual de Aprendizagem)
Período: 3º semestre
Professor: A designar

EMENTA

- Estudo integrado dos modelos tradicionais (Cascata, Espiral, Incremental, Prototipação) e metodologias ágeis (*SCRUM, Kanban, XP*) de desenvolvimento de software.
- Práticas de engenharia de requisitos, modelagem com *UML e C4 Model*, versionamento com *Git*, integração contínua (CI/CD), testes automatizados (TDD, BDD) e qualidade de software.
- Desenvolvimento de projetos colaborativos com aplicação de ferramentas reais utilizadas no mercado. Introdução à arquitetura de microsserviços e *DevOps.*

OBJETIVOS GERAIS

- Formar profissionais aptos a compreender, aplicar e adaptar diferentes metodologias de desenvolvimento de software.
- Desenvolver competências técnicas e colaborativas alinhadas às demandas contemporâneas do mercado de TI.

OBJETIVOS ESPECÍFICOS

- Aplicar modelos de ciclo de vida de software conforme o tipo de projeto, incluindo prototipação como ferramenta de validação e refinação de requisitos.
- Desenvolver software utilizando *frameworks* ágeis com ferramentas de apoio.
- Modelar requisitos e sistemas com ferramentas visuais e técnicas de documentação.
- Aplicar testes automatizados desde o início do desenvolvimento.
- Integrar conceitos de qualidade, segurança e manutenibilidade ao produto de software.

CONTEÚDO PROGRAMÁTICO

1. Fundamentos de Engenharia de Software
2. Modelos de Desenvolvimento:
    - Cascata, Incremental, Espiral, Prototipação, *RUP*
    - *SCRUM, XP, Kanban*
3. Engenharia de Requisitos:
    - Requisitos de Negócio
    - Requisitos funcionais e não funcionais
    - Requisitos Cognitivos
    - Histórias de usuário, *MVP*, testes A/B
4. Modelagem:
    - *UML* (Casos de Uso, Sequência, Atividades)
    - *C4 Model* (níveis 1 a 3)
5. Versionamento e Integração Contínua:
    - *Git, GitHub, CI/CD com GitHub Actions*
6. Testes de Software:
    - *TDD, BDD*, testes unitários, funcionais e não funcionais
7. Qualidade e Auditoria:
    - ISO 9126, ISO 25000, MPS.BR, CMMI

8. Arquitetura Moderna:
    - Microsserviços, *MVC, DDD*
9. Projeto Final:
    - Equipe ágil, *sprints*, revisões, demonstrações e entrega

METODOLOGIA E AVALIAÇÃO

- **Metodologia:** Aulas expositivas dialogadas, estudos de caso, simulações com ferramentas reais, dinâmicas de grupo e desenvolvimento orientado por projetos (ABPj).

AVALIAÇÃO

- P1 (atividades e entregas parciais) – 20%
- P2 (projeto em equipe e aplicação prática) – 30%
- Prova Individual (teórico-prática) – 20%
- Apresentação final + Documentação técnica – 30%

BIBLIOGRAFIA BÁSICA

- BECK, K. **Test-Driven Development:** by Example**.** Boston: Addison-Wesley, 2002.
- PRESSMAN, Roger; MAXIM, B. R. **Software engineering**: a practitioner's approach. McGraw-Hill, 2020.
  SCHWABER, Ken; SUTHERLAND, Jeff**. The Scrum Guide:** The Definitive Guide to Scrum: The Rules of the Game. Novembro de 2020. Disponível em: https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-US.pdf. Acesso em: 16 ago. 2025.
- SOMMERVILLE, Ian. Engenharia de software. Tradução de Ivan Bosnic e Kalinka G. de O. Gonçalves; revisão técnica de Kechi Hirama. 9. ed. São Paulo: Pearson Prentice Hall, 2011.

BIBLIOGRAFIA COMPLEMENTAR

- BATES, A. W. Teaching in a Digital Age: **Guidelines for Designing Teaching and Learning**. 2. ed. Vancouver: BCcampus, 2019.
- FREEMAN, E.; ROBSON, E. **Use a Cabeça! Padrões de Projeto**. São Paulo: Alta Books, 2021.
- KOSCIANSKI, A.; SOARES, M. S. **Qualidade de Software**. Rio de Janeiro: Novatec, 2016.
- PIMENTEL, M. da G. C.; CASTRO, J. C. **Engenharia de Requisitos**: Software Orientado ao Negócio. São Paulo: Atlas, 2020.

Fonte: Elaboração própria

Ao integrar inovação, prática profissional e fundamentação teórica, essa proposta reafirma a necessidade de que a universidade cumpra seu papel de ponte entre o conhecimento científico e as transformações sociais e tecnológicas.

6.7.2 Estrutura MEC

No quadro 11, apresenta-se uma análise comparativa entre as DCN do MEC para cursos de Análise e Desenvolvimento de Sistemas e a proposta da disciplina Engenharia de Software Híbrida, descrita no tópico anterior.

A comparação foi estruturada com base em critérios fundamentais descritos nas DCN, visando avaliar o grau de alinhamento da proposta com as competências, práticas e metodologias exigidas pelo ensino superior brasileiro.

Quadro 11: Comparativo MEC x Proposta do Plano de Ensino

Eixo/Item	Diretriz MEC (Resumo)	Proposta Desenvolvida	Alinhamento
Competências e Habilidades	Projetar e manter sistemas de software	Modelos de desenvolvimento e arquitetura moderna com ferramentas reais	Total
Atualização Tecnológica	Usar metodologias e práticas emergentes	*SCRUM, Kanban, DevOps, TDD, CI/CD*	Total
Integração Teoria e Prática	Basear-se em problemas reais	Projetos reais, revisões, entregas incrementais	Total
Trabalho Colaborativo	Atividades colaborativas e integradoras	Projeto em equipe com papéis ágeis	Total
Desenvolvimento com Qualidade	Desenvolvimento, teste e manutenção de software de qualidade	Requisitos, testes, qualidade ISO e MPS.BR	Total
Ética e Responsabilidade	Ênfase ética e social	Não explícito	Parcial
Ambientes Virtuais	Tecnologias digitais na aprendizagem	Canvas, *GitHub, Trello*, simuladores	Total
Flexibilidade Curricular	Contextualização e adaptação curricular	Estrutura modular e adaptável	Total

Fonte: Elaboração própria

A proposta da disciplina apresenta alto grau de conformidade com as diretrizes do MEC, evidenciando a modernização curricular e a integração entre teoria, prática e mercado.

O único aspecto parcialmente atendido refere-se à ética e responsabilidade social, o qual poderá ser aprimorado com a inclusão explícita de temas transversais e projetos voltados ao impacto social da tecnologia.

Ética profissional e impactos sociais da tecnologia, devem ser incluidos de forma mais explícita no conteúdo programático ou como módulo transversal ao projeto final. Essa adição garantiria aderência total a todas as competências previstas pelas DCN.

Pontos Fortes Identificados

- Aderência total às competências esperadas pelo MEC para profissionais de computação.
- Atualização tecnológica superior à média encontrada nos documentos do PE_IES A–E.
- Forte ênfase em práticas ativas, aprendizagem por projeto, integração com o mercado e simulações reais.
- Uso de ferramentas modernas e metodologias ágeis, favorecendo a empregabilidade e a inovação.

6.7.3 Sugestão de Acompanhamento e Avaliação Contínua

Considerando a natureza dinâmica do setor de tecnologia da informação e as constantes transformações nas práticas de desenvolvimento de software, recomenda-se a institucionalização de um processo contínuo de acompanhamento e avaliação das estratégias curriculares implementadas.

O processo de acompanhamento e avaliação contínua é necessário para monitorar a eficácia das estratégias adotadas ao longo do tempo, garantindo que elas continuem a atender às necessidades do mercado de trabalho em constante evolução. A avaliação contínua permitirá ajustes e melhorias contínuas, assegurando que a formação oferecida pelos cursos de graduação mantenha-se atualizada e relevante.

Esse processo deve ser estruturado em três frentes complementares, capazes de garantir um ciclo de melhoria contínua nos cursos de Engenharia de Software: o monitoramento de resultados acadêmicos, a interação com o mercado de trabalho e a revisão pedagógica periódica.

A primeira frente, o monitoramento de resultados acadêmicos, visa a acompanhar indicadores de desempenho dos estudantes, como taxas de aprovação, evasão, participação em projetos e evolução das competências técnicas e

comportamentais ao longo do curso. Esse acompanhamento permite identificar fragilidades pedagógicas, descompassos entre conteúdo e avaliação e a necessidade de reforço em áreas críticas do currículo. Ferramentas de gestão acadêmica, *feedbacks* sistemáticos dos discentes e autoavaliações regulares devem compor esse processo.

A segunda frente está relacionada à interação contínua com o mercado de trabalho, por meio de parcerias com empresas, participação em eventos técnicos, programas de estágio supervisionado e pesquisas de egressos. Essa articulação permite avaliar se os conteúdos e as metodologias adotadas estão efetivamente alinhados com as exigências da indústria de software. Além disso, facilita a adaptação curricular a novas demandas tecnológicas, linguagens emergentes e práticas ágeis cada vez mais predominantes no setor.

Por fim, a revisão pedagógica periódica constitui a terceira frente estratégica. Trata-se de um processo institucionalizado e colaborativo de reavaliação das ementas, métodos de ensino, bibliografias e estratégias avaliativas, com participação ativa de docentes, coordenadores e estudantes. Essa revisão deve ocorrer com base em evidências, como os dados coletados nas duas frentes anteriores, permitindo o aprimoramento constante do curso e assegurando sua relevância diante das transformações da sociedade digital e das inovações tecnológicas que impactam diretamente o campo da Engenharia de Software.

A proposta é que os cursos mantenham mecanismos regulares de coleta de dados quantitativos e qualitativos, como avaliações discentes, análise de desempenho em projetos práticos, *feedbacks* de docentes e entrevistas com egressos e empregadores. Tais dados devem embasar decisões sobre ajustes no currículo, inclusão de novas tecnologias, remoção de conteúdos obsoletos e revisão das metodologias pedagógicas utilizadas.

Para Dias Sobrinho (2005), a avaliação institucional contínua não deve ser vista apenas como uma exigência normativa, mas como um mecanismo de transformação e garantia de qualidade acadêmica.

Além disso, é essencial a criação de comitês pedagógicos integrados por professores, estudantes e representantes do setor produtivo, garantindo que o currículo esteja permanentemente alinhado às necessidades profissionais e sociais. Essa prática é especialmente recomendada nas Diretrizes Curriculares Nacionais, que

enfatizam a formação por competências e a flexibilidade adaptativa como princípios estruturantes.

A revisão curricular deve ser cíclica, colaborativa e contextualizada, considerando os avanços científicos, tecnológicos e sociais. (Brasil, 2018)

A avaliação contínua permitirá, ainda, que as mudanças implementadas na disciplina de Engenharia de Software evoluam de maneira Incremental, seguindo os mesmos princípios que norteiam as práticas ágeis: *feedback* constante, melhoria contínua e adaptação iterativa.

Dessa forma, assegura-se que a formação oferecida se mantenha não apenas tecnicamente atualizada, mas pedagogicamente eficaz, promovendo o desenvolvimento de profissionais preparados para atuar em um cenário de complexidade crescente, inovação permanente e profunda transformação digital

7 CONSIDERAÇÕES FINAIS

Este trabalho teve como objetivo analisar criticamente o plano de ensino da disciplina de Engenharia de Software em cinco instituições particulares de ensino superior da cidade de Sorocaba/SP, buscando compreender como os modelos tradicionais e ágeis de desenvolvimento de software estão incorporados nos currículos e de que forma esse ensino dialoga com as demandas reais do mercado de tecnologia da informação. A partir da análise documental dos planos, foi possível mapear as abordagens metodológicas adotadas, identificar lacunas formativas e propor estratégias concretas de melhoria.

Os resultados revelaram um cenário heterogêneo. Enquanto alguns planos, como PE_IES C, apresentam currículos atualizados, com inserção de metodologias ágeis, ferramentas modernas e práticas pedagógicas inovadoras, outras, como os PE_IES A e D, ainda se mantêm ancoradas em modelos tradicionais, com foco predominantemente teórico e pouca articulação com o mercado. Essa diversidade reflete o desafio enfrentado pelas universidades em alinhar formação acadêmica e exigências profissionais num setor que evolui constantemente.

A Engenharia de Software é, portanto, uma tecnologia de importância crítica para o futuro da humanidade e que convém continuar a orientar os estudantes que pretendem atuar neste assunto, a aplicar os modelos já existentes. Convém absorver o que há de melhor em cada processo e seguir os conceitos para que se obtenham projetos de software cada vez melhores, facilitando o processo, principalmente quando houver a necessidade de mudanças e inserções de novos requisitos pelo cliente da complexidade do software.

Pode-se concluir após este estudo que o conhecimento em Engenharia de Software representa grande parte do necessário para a solução do desenvolvimento de software, pois fornece uma infraestrutura a partir da qual os modelos, métodos e ferramentas podem ser aplicados e avaliados conforme o método mais apropriado. Não existe uma solução mágica para o desenvolvimento de software, portanto, faz-se necessário aprender os conceitos durante a universidade e aplicar uma abordagem disciplinada. Seguindo os princípios da Engenharia de Software, seja em processos convencionais ou ágeis, alcança-se a construção de produtos com qualidade para o cliente.

O sucesso dos métodos ágeis como *Extreme Programming* (Beck, 2000), *SCRUM* (Schwaber; Sutherland, 1997), e sua integração com métodos mais tradicionais como Cascata (Royce, 1970), Prototipação (Yourdon, 1990) e Espiral (Boehm, 1988) resulta em abordagens híbridas que representam o que há de melhor na aplicação da disciplina de Engenharia de Software.

Os resultados também demonstram que as instituições que adotam estratégias baseadas em práticas de mercado, tendem a apresentar melhor alinhamento entre o ensino acadêmico e as demandas profissionais. O PE_IES C, por exemplo, destaca-se por integrar ambientes virtuais (*Canvas*), ferramentas profissionais, entregas ágeis e desenvolvimento Incremental com avaliação contínua, prática e colaborativa. Essa aproximação com o mercado está em consonância com os princípios estabelecidos pelas Diretrizes Curriculares Nacionais (Brasil, 2018), que preveem a formação por competências e o estímulo à interdisciplinaridade e à inovação.

Algumas instituições demonstraram defasagem na incorporação de tecnologias e metodologias contemporâneas, mantendo uma estrutura curricular que pouco dialoga com o ecossistema atual de desenvolvimento de software. Essa lacuna reforça o alerta de autores como Perrenoud (2000), ao defender que as instituições de ensino devem superar o modelo tradicional de transmissão de conteúdo e incorporar práticas pedagógicas que promovam a autonomia, a colaboração e a resolução de problemas reais. Não basta conhecer o conteúdo, o professor moderno precisa saber mediar, guiar e estimular processos de aprendizagem colaborativos e ativos. (Perrenoud, 2000)

As propostas elaboradas neste estudo, como a revisão de ementas, a adoção de metodologias ativas, a ampliação do uso de ambientes virtuais, a capacitação docente e a integração com empresas locais foram validadas teoricamente a partir de literatura especializada e indicam caminhos viáveis para o aprimoramento do ensino. A implementação dessas estratégias, acompanhada por um processo contínuo de avaliação, pode reduzir significativamente o *gap* entre universidade e mercado, promovendo uma formação mais prática, atualizada e centrada no desenvolvimento de competências profissionais.

Recomenda-se o uso do melhor de cada processo, com atenção total ao escopo para que se possa fazer um excelente levantamento de requisitos. Independentemente do modelo adotado, é essencial descrever todas as atividades, respeitar prazos, adotar práticas como *Clean Code*, manter a documentação

atualizada e planejada para facilitar a manutenção e o treinamento contínuo da equipe.

Ao confirmar a hipótese levantada, conclui-se que a transformação do ensino de Engenharia de Software passa pela superação de modelos pedagógicos estanques, pela abertura à inovação e pela construção de pontes efetivas entre o mundo acadêmico e o universo profissional. Como destaca Moran (2015), educar para o futuro exige criar experiências significativas que preparem os estudantes para contextos que ainda estão em construção. Nesse sentido, esta pesquisa busca oferecer essa contribuição ao apresentar um diagnóstico detalhado, propostas aplicáveis e uma reflexão crítica sobre os caminhos para uma educação mais alinhada com o século XXI.

REFERÊNCIAS

ALI, J. **Estudo para Haystack Analytics**. Relatório técnico. JL Partners para Haystack Analytics, dez. 2023.

BACICH, Lilian; MORAN, José (orgs.). **Metodologias ativas para uma educação inovadora: uma abordagem teórico-prática.** Porto Alegre: Penso, 2018.

BATES, A. W. Teaching in a Digital Age: **Guidelines for Designing Teaching and Learning**. 2. ed. Vancouver: BCcampus, 2019.

BECK, Kent. et al. **Manifesto for Agile Software Development.** 2001. Disponível em: https://agilemanifesto.org/. Acesso em: 13 jan. 2025

BECK, Kent.; Rensburg, J. **Extreme Programming Explained:** Embrace Change Addison-Wesley, 2021.

BECK, Kent; ANDRES, **Cynthia. Extreme Programming Explained:** Embrace Change. 2. ed. Boston: Addison-Wesley, 2004.

BECK, Kent. **Extreme programming explained**. Reading: Addison-Wesley, 2000.

BECK, Kent. **Test-Driven Development:** by Example**.** Boston: Addison-Wesley, 2002.

BERKUN, Scott. **The Myths of Innovation.** Sebastopol, CA: O'Reilly Media, 2010

BOEHM, Barry W. **Software Engineering Economics**. Englewood Cliffs, NJ: Prentice Hall, 1988.

BOEHM, Barry. A Spiral Model of Software Development and Enhancement**. IEEE Computer**, v.21, n.5, maio. 1988.

BRASIL. Ministério da Educação. **Diretrizes Curriculares Nacionais para os Cursos de Graduação na área de Computação e Informática.** MEC/CNE/CES, 2018.

BRASIL. Ministério da Educação. Conselho Nacional de Educação. Câmara de Educação Superior. **Diretrizes Curriculares Nacionais para os cursos de graduação na área de Computação**. Resolução CNE/CES nº 5, de 16 de novembro de 2016. Brasília: MEC, 2016. Disponível em: http://portal.mec.gov.br/. Acesso em: 16 ago. 2025.

COHN, M. **Desenvolvimento de Software com SCRUM**. Bookman, 2011.

CRAIG, M.; CONRAD, P.; LYNCH, D.; LEE, N.; ANTHONY, L. Listening to Early Career Software Developers. **Journal of Computing Sciences in Colleges**, v. 33, n. 4, p. 138–149, Apr. 2018.

CUNNINGHAM, Ward. *The Wyatt Earp Effect*. In: BECK, Kent (Ed.). ***Extreme Programming Perspectives*.** The XP Series. Boston: Addison-Wesley, 2001. p. 19–23.

DAVENPORT, Thomas H.; HARRIS, Jeanne G. ***Competing on Analytics: The New Science of Winning*.** Boston: Harvard Business School Press, 2007.

DIAS SOBRINHO, José. **Avaliação, compromisso social e qualidade: os desafios da educação superior no século XXI**. Campinas: Papirus, 2005.

DIJKSTRA, E. W. The humble programmer. **Communications of the ACM**, New York, v. 15, n. 10, p. 859-866, Oct. 1972.

ETZKOWITZ, Henry; LEYDESDORFF, Loet. The Dynamics of Innovation: From National Systems and “Mode 2” to a Triple Helix of University–Industry–Government Relations. **Research Policy,** v. 29, p. 109–123, 2000.

FACIONE, Peter A. **Critical Thinking:** What It Is and Why It Counts. Millbrae, CA: Measured Reasons and The California Academic Press, 2011.

FRANZONI, Vilma; BOAVENTURA, Regina Célia Ferreira; GONÇALVES, Maria Carla Pascotte Freitas; UNIVERSIDADE DE SOROCABA. **Manual para normalização de trabalhos acadêmicos**: tese, dissertação, trabalho de conclusão de curso (TCC), artigo científico e pôster. Sorocaba, SP: EDUniso, 2012.

GARCIA, L. J.; DUFRESNE, D. Designing **Flexible Curricula for the Future of Software Engineering Education**. Wiley, 2018.

GIL, Antonio Carlos. **Como elaborar projetos de pesquisa.** 5. ed. São Paulo: Atlas, 2010.

GIL, Antonio Carlos. **Métodos e técnicas de pesquisa social**. 6. ed. São Paulo: Atlas, 2008.

GRAHAM, C. R.: Blended Learning Systems: **Definition, Current Trends, and Future Directions**. Pfeiffer, 2019.

HAZZAN, Orit; DUBINSKY, Yael. **Agile Software Engineering**. London: Springer, 2008.

HIGHSMITH, Jim. **Agile Software Development Ecosystems**. Boston: Addison-Wesley, 2002.

KATZENBACH, J. R., & Smith, D. K. **The Wisdom of Teams: Creating the high-performance organization.** New York, NY: HarperBusiness, 1993.

KOLMOS, A., De Graaff, E., & Du, X. **Problem-Based and Project-Based Learning in Engineering Education**. Sense Publishers, 2020.

LAWRENCE, P. S. **Engenharia de Software**: teoria e prática. 2. ed. São Paulo: Pearson Education, 2004.

LAKATOS, Eva Maria; MARCONI, Marina de Andrade. **Fundamentos de metodologia científica**. 9. ed. São Paulo: Atlas, 2021.

LESSA, Rafael Orivaldo; LESSA, Edson Orivaldo Junior. **Modelos de processos de engenharia de software.** Universidade do Sul de Santa Catarina (UNISUL). Disponível em: https://ead.uepg.br/apl/sigma/assets/editais/PS0026E0068.pdf. Acesso em: maio 2024.

LETHBRIDGE, T. C., & Laganière, R. **Object-Oriented Software Engineering: Practical Software Development Using UML and Java**. McGraw-Hill, 2019.

LOBO, Edson J. R. **Curso de engenharia de software**. São Paulo: Digerati Books, 2008. Disponível em: https://books.google.com.br/books?id=ZJznA9UrtVAC&printsec=frontcover&dq=isbn:8578730100&hl=pt-PT&sa=X&ved=0ahUKEwjbpraQ_ZzlAhXnDrkGHU4BD7oQ6AEIKDAA#v=onepage&q&f=false. Acesso em: jun. 2024.

MANIFESTO PARA DESENVOLVIMENTO ÁGIL DE SOFTWARE. [S.l.: s.n.], 2001. Disponível em: http://www.agilemanifesto.org/iso/ptbr/
. Acesso em: 02 abr. 2014.

MORAN, José Manuel. **A educação que desejamos: novos desafios e como chegar lá.** 5. ed. Campinas: Papirus, 2015.

MCKINSEY & COMPANY. **The Future of Work: A Journey to 2025**, 2021

MINAYO, Maria Cecília de Souza. **O desafio do conhecimento:** pesquisa qualitativa em saúde. 14. ed. São Paulo: Hucitec, 2014.

MORAVEC, J. W. **Learning in the Digital Age: Adaptation and Flexibility in Higher Education**. Springer, 2021.

MOURA, D.; OLIVEIRA, L. T. **Aprendizagem baseada em projetos:** fundamentos e práticas. Campinas: Papirus, 2020.

NÓVOA, António. **Os professores e a sua formação**. Lisboa: Educa, 2007.

PARNAS, D. L., & CLEMENTS, P. C. "A Rational Design Process: How and Why to Fake It." **IEEE Transactions on Software Engineering**, 32(12), 980-990, 2021.

PAULA FILHO, W. de P. **Engenharia de Software:** fundamentos, métodos e padrões**.** 3.ed. Rio de Janeiro: LTC, 2012.

PERRENOUD, Philippe. **Dez novas competências para ensinar**. Porto Alegre: Artmed, 2000.

POLYA, George. *How to Solve It:* **A New Aspect of Mathematical Method.** Princeton: Princeton University Press, 1957.

PRESSMAN, Roger & MAXIM, B. R.Software **Engineering: A Practitioner's Approach**: McGraw-Hill, 2020.

PRESSMAN, Roger; MAXIM, B. R. **Software engineering:** a practitioner's approach. McGraw-Hill, 2020.

PRESSMAN, Roger. **Engenharia de software**: uma abordagem profissional. 7. ed. São Paulo: McGraw-Hill, 2011.

OGUZ, Derya; OGUZ, Ozlem. Traditional and agile software development methodologies: A survey. **International Journal of Computer Science and Information Security (IJCSIS),** v. 17, n. 4, p. 30-36, 2019.

REZENDE, Denis Alcides. **Engenharia de Software e Sistemas de Informação**. Rio de Janeiro: Brasport, 2005.

ROYCE, W. Managing the development of large software systems: **Concepts and techniques**. In: Proc. IEEE WESCOM. IEEE Computer Society Press, Los Alamitos. 1970.

SCHACH, Stephen R. **Engenharia de Software.** 7. ed. Porto Alegre: AMGH, 2010.

SCHWABER, K.; SUTHERLAND, J. **Guia do SCRUM**. Disponível em: https://www.SCRUMguides.org/docs/SCRUMguide/v1/SCRUM-Guide-Portuguese-BR.pdf . Acesso em: maio 2024.

SCHWABER, Ken; SUTHERLAND, Jeff. **The Scrum Guide:** The Definitive Guide to Scrum: The Rules of the Game. Novembro de 2020. Disponível em: https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-US.pdf . Acesso em: 16 ago. 2025.

SCHWABER, Ken. Scrum development process. In: **Business object design and implementation: OOPSLA'95 workshop proceedings 16 October 1995, Austin, Texas**. London: Springer London, 1997. p. 117-134.

SCHWABER, Ken; BEEDLE, Mike. **Agile Software Development with Scrum**. Upper Saddle River: Prentice Hall, 2002.

SENGE, P. M. **The Fifth Discipline: The Art and Practice of the Learning** Organization. Doubleday, 1990.

SHULMAN, L. S. **The Wisdom of Practice: Essays on Teaching, Learning, and Learning to Teach** Jossey-Bass, 2019.

SMITH, John. **Lifelong Learning in Software Engineering:** A Key to Success. Journal of Software Engineering, 2012.

SOMMERVILLE, Ian. **Engineering software products**. London, UK: Pearson, 2020.

SOMMERVILLE, Ian. **Engenharia de software.** Tradução de Ivan Bosnic e Kalinka G. de O. Gonçalves; revisão técnica de Kechi Hirama. 9. ed. São Paulo: Pearson Prentice Hall, 2011.

VALENTE, José Armando. ***Blended learning e as mudanças no ensino superior:*** *a proposta da sala de aula invertida***.** *Educar em Revista*, Curitiba, n. 4, p. 79-97, 2014.

YOURDON, Edward. **Análise Estruturada Moderna**. 3ª Ed. Trad. Dalton C. de Alencar. Rio de Janeiro: Campus, 1990.

ANEXO A – QUADRO DESCRITIVO DAS SIGLAS

Sigla/Termo	Significado	Descrição resumida
OOPSLA	*Object-Oriented Programming, Systems, Languages & Applications*	Conferência científica focada em programação orientada a objetos e paradigmas emergentes.
Git	Sistema de controle de versão	Ferramenta usada para rastrear mudanças em código fonte e colaborar em equipe.
FDD	*Feature-Driven Development*	Metodologia ágil baseada no desenvolvimento incremental por funcionalidades.
DSDM	*Dynamic Systems Development Method*	*Framework* ágil voltado para desenvolvimento rápido, iterativo e baseado em entregas.
CMMI	*Capability Maturity Model Integration*	Modelo de maturidade para melhorar processos organizacionais em desenvolvimento de software.
MPS.Br	Melhoria de Processo do Software Brasileiro	Modelo nacional semelhante ao *CMMI*, adaptado à realidade brasileira.
DevOps	*Development + Operations*	Prática que integra desenvolvimento e operações para acelerar entregas e automação.
PSP	*Personal Software Process*	Método para desenvolvedores melhorarem produtividade e qualidade de forma individual.
IDE	*Integrated Development Environments*	Ambientes integrados para escrever, compilar, depurar e testar código (ex.: VSCode, Eclipse).
MVP	*Minimum Viable Product*	Versão mínima funcional de um produto usada para validar hipóteses com usuários reais.

Testes A/B	Testes comparativos	Técnica para comparar duas versões de um produto com usuários reais e medir desempenho.
UML	*Unified Modeling Language*	Linguagem padrão para modelagem de sistemas orientados a objetos.
C4 Model	*Context, Containers, Components, Code*	Modelo de arquitetura de software em quatro níveis de abstração visual.
MVC	*Model-View-Controller*	Padrão arquitetural que separa dados (modelo), interface (visão) e lógica (controle).
DDD	*Domain-Driven Design*	Estratégia de *design* focada em modelar sistemas baseados no domínio de negócio.
BDD	*Behavior-Driven Development*	Abordagem de testes focada no comportamento esperado do software, em linguagem natural.
PBL	*Problem-Based Learning*	Metodologia ativa em que os estudantes aprendem resolvendo problemas reais e complexos.

Fonte: Autoria própria

ANEXO B – PLANOS DE ENSINO - PE_IES A

1. Caracterização do componente curricular
Componente curricular: ENGENHARIA DE SOFTWARE
Curso: 115 - ANÁLISE E DESENVOLVIMENTO DE SISTEMAS
Carga horária: 80Créditos
Docente responsável:
Semestre/ano:

2. Objetivos
Apresentar os conceitos básicos da engenharia de software. Estudar os principais métodos para o desenvolvimento de software. Estudar o processo unificado. Estudar os princípios do gerenciamento de projetos. Conhecer as principais métricas de software e seu uso nas estimativas. Conhecer as principais arquiteturas de software. Conhecer os processos de teste de software e suas técnicas. Conhecer os padrões de qualidade de software. Entender o processo de configuração e mudança.

3. Ementa
Métodos de análise e de projeto de software; Qualidade de software; Verificação, validação e teste; Manutenção; Gerenciamento de configurações de software; Ambientes de desenvolvimento de software.

4. Conteúdo Programático e Competências (Conhecimento, Habilidades e Atitudes)
1. Métodos de análise e de projeto de Software.
2. Métricas de Software.
3. Projeto e implementação.
4. Teste de Software.
5. Qualidade de Software.
6. Gerenciamento de configuração de software.
7. Avaliações.

5. Estratégias, recursos humanos e materiais
Estratégias
Exposição teórica.
Exemplos para ilustrar as aplicações.
Atividades
Resolução de Exercícios em sala de aula e proposta de exercícios para serem realizados extraclasses;
Recursos Materiais
Projetor Multimídia, lousa para exposição de dados e informações, material de apoio didático.
Laboratório com computadores, linguagem de programação instalada.

Instrumentos de Avaliação:
Os conceitos de aproveitamento acadêmico serão expressos em notas, atribuídas numa escala de zero a dez, com variação decimal de 0,5 (meio) ponto. A média final do discente dar-se-á a partir de duas avalições objetivas individuais e da média das atividades. Será aprovado o aluno que obtiver média final maior ou igual a 6.

O discente que não entregar a atividade até a data definida pelo professor ficará com nota zero nessa atividade.
Para os estudantes que não atingirem a média mínima exigida, ou que por algum motivo não compareceram em uma das avaliações, será oferecida uma avaliação substitutiva (NP3), no final do semestre, correspondente a todo o conteúdo estudado, assim, o aluno terá a oportunidade de uma nova média substituindo a menor nota entre as duas avaliações.

7. Bibliografia

7.1 Bibliografia Básica

PRESSMAN, Roger S. **Engenharia de software.** Rio de Janeiro: McGraw-Hill; Porto Alegre: AMGH, 1995/2011. 10e. C / 4e. T (005.1 P938e) MB
SOMMERVILLE, Ian**. Engenharia de software**. São Paulo: Addison-Wesley: Pearson Prentice-Hall, 2003/2012. **10e.** (005.1 S679e)
PFLEEGER, Shari Lawrence. **Engenharia de software:** teoria e prática. São Paulo: Pearson, 2004/2012. **13e.** (005.1 P629e) Pearson

7.2 Bibliografia Complementar

CARVALHO, Ariadne M. B. Rizzoni; CHIOSSI, Thelma C. dos Santos. **Introdução à engenharia de software.** Campinas: Universidade Estadual de Campinas, 2001. **3e.** (005.1C321i)
PETERS, James F.; PEDRYCZ, Witold. **Engenharia de software:** teoria e prática. Rio de Janeiro: Campus, 2001. **5e.** (005.1 P575e)
PAULA FILHO, Wilson de Pádua. **Engenharia de software:** fundamentos, métodos e padrões. Rio de Janeiro: LTC, 2003. **2e.** (005.1 P347e) MB
BOOCH, Grady; RUMBAUGH, James; JACOBSON, Ivar. **UML, guia do usuário.** Rio de Janeiro: Campus, 2000. **5e.** (005.1 B715um)
KOSCIANSKI, André; SOARES, Michel dos Santos. **Qualidade de software:** aprenda as metodologias e técnicas mais modernas para o desenvolvimento de software. São Paulo:Novatec, 2006. **5e.** (005.1 K88q)

ANEXO C – PLANOS DE ENSINO - PE_IES B

PARTE A – IES B

CURSO: Tecnologia em Análise e Desenvolvimento de Sistemas
SÉRIE: 2º Semestre
DISCIPLINA: ENGENHARIA DE SOFTWARE I
CARGA HORÁRIA SEMANAL: 03 horas-aula
CARGA HORÁRIA SEMESTRAL: 60 horas-aula

I - EMENTA

Engenharia de Software. Fundamentação dos Princípios da Engenharia de Software. Conceituação de Produto de software. Processo de Software. Processo Pessoal de Software. Processo de Equipe de software. Modelos de Processos de Software. Modelos Ágeis. Introdução a Engenharia de Requisitos

II – OBJETIVOS GERAIS

Proporcionar ao aluno conhecimento em métodos e técnicas de projeto que auxiliam o processo de desenvolvimento de software.

III – OBJETIVOS ESPECÍFICOS

Apresentar os principais métodos e técnicas de projeto que auxiliam o processo de desenvolvimento de software.
Identificar, descrever e comparar os modelos de processo de desenvolvimento de software.
Habilitar o aluno para escolher, utilizar e definir modelos, técnicas e ferramentas para auxiliar o processo como produto.

IV - CONTEÚDO PROGRAMÁTICO

1. Fundamentos da engenharia de software
    - Conceitos e objetivos
    - O papel evolutivo do software
    - Software produto
    - A natureza mutável do software
    - Tipos de aplicações de software
    - Problemas com prazo, planejamento e custos
    - Qualidade de software
2. O processo de software
    - Etapas do processo de software
3. PSP - Processo Pessoal de Software
4. TSP – Processo de Equipe de Software
5. Modelos de processos de software (Modelos de ciclo de vida de software)
    - Codifica-remenda
    - Waterfall (cascata)
    - Incremental
    - RAD
    - Evolucionário: Espiral, Prototipação
    - Modelos especializados de processo: Baseado em Componentes, Baseado em Métodos Formais
    - Processo Unificado

RUP
Praxis
Cleanroom (Sala Limpa)
Iconix
6. Modelos Ágeis
O que é um processo ágil?
Modelos ágeis de processos
XP – Extreme programming
Scrum
Feature Driven Development (FDD)
Adaptative Software Development (ASD)
Dynamic Systems Development Method (DSDM)
Crystal
Modelagem Ágil (AM)
7. Prática da Engenharia de Software
Princípios centrais
Práticas de Comunicação, Planejamento, Modelagem, Construção e Implantação

V – ESTRATÉGIA DE TRABALHO

- Aulas expositivas
- Aulas reflexivas com análise de casos
- Dinâmica de grupos
- Seminários
- Vídeos
- Debates

Será sempre indicada a bibliografia básica e específica necessária ao acompanhamento do curso e orientação do aluno na vida acadêmica e profissional.
A exposição será feita por meio de colocação dos pontos a serem discutidos de forma esquemática, seguida de apresentação por parte do professor. Para todas as exposições e para todos os pontos deverão ser utilizadas
apresentações de casos práticos.

VI – AVALIAÇÃO

A avaliação será obtida por meio de provas, trabalhos e seminários, dentre outros, bem como pela participação do aluno durante as aulas e demais atividades, a critério do professor e em conformidade com o respectivo plano de ensino. Serão feitas avaliações, assim distribuídas:

- Duas Notas do Professor (NP) para as atividades curriculares, com peso 4 (quatro) cada uma, na composição da nota semestral de cada disciplina;
- Uma nota referente ao Projeto Integrado Multidiscipinar (PIM), com peso 2 (dois) no cálculo da Média Semestral (MS) de cada disciplina. Esse Projeto será desenvolvido durante o semestre.

PARTE B – UNIVERSIDADE B

CURSO: Tecnologia em Análise e Desenvolvimento de Sistemas
SÉRIE: 2º Semestre
DISCIPLINA: ENGENHARIA DE SOFTWARE II
CARGA HORÁRIA SEMANAL: 03 horas-aula
CARGA HORÁRIA SEMESTRAL: 60 horas-aula

I - EMENTA

Qualidade de Software. Modelos de qualidade de software. Verificação e Validação. Testes. Plano e casos de teste. Técnicas e tipos de teste. Revisões técnicas formais. Manutenção de Software. Tipos de manutenção. Gerência de configuração.

II – OBJETIVOS GERAIS

Apresentar ao aluno os aspectos mais importantes para a qualidade de um produto de software, bem como a importância das atividades de testes e da manutenção de software.

III – OBJETIVOS ESPECÍFICOS

Apresentar ao aluno conceitos de qualidade de produto e de processo, bem como os modelos de qualidade de software. Conscientizar os estudantes sobre a importância da Verificação e Validação para a qualidade do software que é produzido, bem como a importância dos testes e seu impacto nos custos de desenvolvimento do software. Apresentar as atividades de teste e as principais técnicas empregadas. Apresentar ao aluno a importância da manutenção no ciclo de vida de um software. Mostrar que esta atividade envolve não somente o código, mas também todos os documentos do projeto.

IV - CONTEÚDO PROGRAMÁTICO

1. Qualidade de software
- Conceitos de qualidade de produto e de processo
- Qualidade do produto de software: ISO/IEC 9126 e ISO 25000
- Sistemas da Qualidade: ISO 90003 e ISO 9001
- Processos do Ciclo de Vida do Software: ISO 12207
- Modelos de qualidade de software

CMMI (Capability Maturity Model Integration)
MPS.Br (Melhoria de Processos de software Brasileiro)
SPICE - ISO 15504

2. Verificação e Validação de software
- Definição e importância da Verificação e Validação ao longo do ciclo de vida
- Classificação das técnicas
- Revisões técnicas: Passeio (walkthrough); Inspeção do produto
- Abordagens formais: Prova de correção; O processo sala limpa (clean room)
- Testes: Fundamentos; Os testes e o ciclo de vida
- Testes unitários: Testes Estruturais; Testes Funcionais
- Outras estratégias de teste

3. Manutenção de software

- Manutenção: definição e características
- Manutenabilidade
- Processos de Manutenção
- Técnicas de Desenvolvimento para a Manutenabilidade
- Padrões de Desenvolvimento
- Padrões de Manutenção
- Desenvolvimento Baseado em Componentes e Impactos na Manutenção
- Desenvolvimento Orientado a Aspectos e Impactos na Manutenção
- Atividades de Apoio a Manutenção

4. Gerência de Configuração

V - BIBLIOGRAFIA BÁSICA

PAULA FILHO, W. P. *Engenharia de software*: fundamentos, métodos e padrões. 3. ed. Rio de Janeiro: LTC, 2012.
PRESSMAN, R. S. *Engenharia de software*: uma abordagem profissional. 7. ed. AMGH, 2011.
COSTA, I.; MOLLO NETO, P. L. O.; CARDOSO JÚNIOR, J. L.. *Qualidade em tecnologia da informação*: conceitos de qualidade nos processos, produtos, normas, modelos e testes de software no apoio às estratégias empresariais. São Paulo: Atlas, 2013.

VI - BIBLIOGRAFIA COMPLEMENTAR

SOMMERVILLE, I. *Engenharia de software*. 9. ed. São Paulo: Pearson Prentice Hall, 2011.
COHN, M. *Desenvolvimento de Software com Scrum*. Bookman, 2011.
SCHACH, S. R. *Engenharia de software*: os paradigmas clássicos orientados a objetos. 7. ed. Porto Alegre : AMGH, 2010.
BECK., and Kent. *TDD Desenvolvimento Guiado por Testes*. Bookman, 2010. PEZZÉ, M.; YOUNG, M. *Teste e Análise de Software*: processo, princípios e técnicas. Porto Alegre: Bookman, 2008.

ANEXO D – PLANOS DE ENSINO - PE_IES C

ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

COMPONENTE CURRICULAR: ENGENHARIA DE SOFTWARE

TEORIA 40H / PRÁTICA 40H

Estratégias de ensino	Atividade desenvolvida	Recursos
Webinar presencial sobre Mitos e Verdades sobre o mercado de T.I.	Apresentações de alunos e profissionais que atuam no mercado de trabalho de Tecnologia.	Participação presencial dos alunos, bate-papo, tira dúvidas sobre o mercado de trabalho e profissões.

4 – Apresentação do Plano de aulas

Objetivos de aprendizagem: Competências, habilidades, atitudes e comunicação.

Objetivos de aprendizagem:

- Boas-vindas aos alunos e apresentação do objetivo da aula;
- Breve apresentação da Professora;
- Comunicação, empatia e respeito;
- Introdução dos Fundamentos da Engenharia de Software e a importância desses conceitos no desenvolvimento de sistemas de qualidade atualmente utilizados no mercado;
- Apresentação do material didático e das ferramentas que serão utilizadas durante a aula.

Competências:

- Conhecer o professor, o conteúdo que será estudado, os objetivos de aprendizagem;
- Critérios de avaliação, e as referências utilizadas;
- Integração com os alunos.

5 – Introdução Engenharia de Software

Objetivos de aprendizagem: Competências, habilidades, atitudes e comunicação.

Objetivos de aprendizagem:

- História, definições e contextos;
- Áreas do conhecimento da Engenharia de Software: SWEBOK;
- Prévia do conceito de Projeto de Software;
- Construção de Software;
- Ética da Engenharia de Software.

Competências:

Compreender o papel da história da Engenharia de Software, assim como explanar de forma inicial sobre Projeto de Software e como é realizada sua construção de software baseado em modelo de esteira de desenvolvimento. Exemplos de algumas ferramentas disponíveis.

Estratégias de ensino	Atividade desenvolvida	Recursos
Aula expositiva dialogada sobre a Introdução da Engenharia de Software, apresentação de subáreas da disciplina.	Dinâmica com apresentação de conceitos, metodologias e ferramentas.	Apresentação do SWEBOK, ferramentas e recursos para construção de software (modelo de esteira de desenvolvimento). Exemplos de modelagem com draw.io. Canvas

6 – Ciclo de Vida de Software		
Objetivos de aprendizagem: Competências, habilidades, atitudes e comunicação.		
Objetivos de aprendizagem: • Processo de Software; • Modelo Cascata; • Modelo Incremental; • Modelo Espiral; • Prototipação. **Competências:** Capacitar os alunos(as) para a compreensão dos Ciclos de Vida do Software. Período que se inicia com um conceito para um produto de software e acaba sempre que o software deixa de estar disponível para utilização.		
Estratégias de ensino	**Atividade desenvolvida**	**Recursos**
Aula expositiva dialogada sobre modelos de Ciclo de Vida de Software com exemplos dos modelos.	Questões com exemplos de vantagens e desvantagens de cada modelo.	A livre escolha dos grupos. Entrega na ferramenta Word pelo Canvas de cada modelo apresentado.

7 – Engenharia de Requisitos I		
Objetivos de aprendizagem: Competências, habilidades, atitudes e comunicação.		
Objetivos de aprendizagem: • Técnicas para coleta de dados; • Identificação das regras de negócio; • Requisitos funcionais e não funcionais; • Histórias de Usuários. **Competências**: Capacitar os alunos(as) para o entendimento das técnicas de coleta de requisitos. Diferenciar requisitos funcionais e não funcionais. Identificar as regras de negócio e História de Usuários.		
Estratégias de ensino	**Atividade desenvolvida**	**Recursos**
Aula expositiva dialogada sobre os tópicos abordados na Engenharia de Requisitos I.	Exercícios sobre o tema da aula com modelagem.	Uso de ferramenta de modelagem para requisitos funcionais ou não funcionais. Ou utilizar ferramenta para prototipagem (Ex. Figma). Canvas

8 – Engenharia de Requisitos II

Objetivos de aprendizagem: Competências, habilidades, atitudes e comunicação.

Objetivos de aprendizagem:

- Gerenciamento de Requisitos;
- Casos de Uso;
- MVP;
- Testes A/B.

Competências: Capacitar os alunos(as) para escrever o que são requisitos funcionais e suas características. Identificar quais são os requisitos funcionais da solução que compõem o escopo. Organizar a documentação dos requisitos funcionais. Importância do MVP e dos testes A/B.

Estratégias de ensino	Atividade desenvolvida	Recursos
Aula expositiva dialogada sobre os tópicos abordados na Engenharia de Requisitos II.	Exercícios sobre o tema da aula com modelagem.	Uso de ferramenta de modelagem para Casos de Uso, Fluxograma (UML, Draw.io, Miro ou similares. Canvas

9 – Projeto de Software I

Objetivos de aprendizagem: Competências, habilidades, atitudes e comunicação.

Objetivos de aprendizagem:

- Conceitos Fundamentais de Projeto;
- Integridade Conceitual;
- Ocultamento de Informação.
- Tipos de requisitos não funcionais.

Competências: Capacitar os alunos(as) para compreensão, análise e implementação de um sistema que atenda aos requisitos funcionais e não funcionais, assim como garantir a qualidade e integridade da solução/software definidos pelo cliente.

Estratégias de ensino	Atividade desenvolvida	Recursos
Aula expositiva dialogada sobre os tópicos abordados em Projeto de Software I.	Exercícios sobre o tema da aula com caso de uso sobre os tópicos abordados.	Link do Caso de Uso disponível para acesso no Laboratório. Canvas.

10 – Modelagem I

Objetivos de aprendizagem: Competências, habilidades, atitudes e comunicação.

Objetivos de aprendizagem:

- UML (Diagrama de Casos de Uso);
- UML (Diagrama de Sequência);
- UML (Diagrama de Atividades).

Competências: Capacitar os alunos(as) para utilizaram ferramentas para modelagem de projetos em UML (seguindo os modelos de diagramas mencionados acima).

Estratégias de ensino	Atividade desenvolvida	Recursos
Aula expositiva dialogada sobre os tópicos abordados em Modelagem I com diagramas de UML.	Modelagem de Diagramas de: Casos de Uso; Sequência e Atividades)	Uso de ferramenta de modelagem para os Diagramas (Star UML, Draw.io, etc.). Canvas

11 – Modelagem II

Objetivos de aprendizagem: Competências, habilidades, atitudes e comunicação.

Objetivos de aprendizagem:

- Modelagem C4 Model (Nível 1);
- Modelagem C4 Model (Nível 2);
- Modelagem C4 Model (Nível 3).

Competências: Capacitar os alunos(as) para utilizaram ferramentas para modelagens (seguindo os modelos de diagramas mencionados acima).

Estratégias de ensino	Atividade desenvolvida	Recursos
Aula expositiva dialogada sobre os tópicos abordados em Modelagem II com diagramas de UML e C4 Model.	Modelagem de Diagramas de: Engenharia Reversa, Nuvem, C4 Model níveis 1, 2 e 3.	Uso de ferramenta de modelagem para os Diagramas (Draw.io e C4 Model). Canvas

12 – Versionamento com GIT

Objetivos de aprendizagem: Competências, habilidades, atitudes e comunicação.

Objetivos de aprendizagem:

- Criação de conta e setup;
- Init e clone;
- Commit;
- Pull Request (PR);
- Push e Pull;
- Branches;
- Forks;
- Rebase e demais comandos.

Competências: Capacitar os alunos(as) para a utilização de ferramentas de controle de versionamento (documentação e códigos).

Estratégias de ensino	Atividade desenvolvida	Recursos
Aula expositiva dialogada sobre os tópicos abordados em Versionamento com GIT.	Criação de contas, setup e comandos de versionamento com a ferramenta GIT.	GitHub, VSCode, etc., Canvas

13– Processos e Agilidade I

Objetivos de aprendizagem: Competências, habilidades, atitudes e comunicação.

Objetivos de aprendizagem:

- Importância dos processos;
- Manifesto Ágil;
- XP Programming.

Competências: Capacitar os alunos(as) para compreensão da importância dos processos e agilidade: Manifesto Ágil e XP Programming e demais frameworks de agilidade na Engenharia de Software.

Atividade desenvolvida	Atividade desenvolvida	Recursos
Aula expositiva dialogada sobre os tópicos abordados em Processos e Agilidade I.	Questões com exemplos de vantagens e desvantagens de cada modelo.	Entrega do trabalho via Canvas.

14 – Processos e Agilidade II

Objetivos de aprendizagem: Competências, habilidades, atitudes e comunicação.

Objetivos de aprendizagem:

- Scrum;
- Kanban;
- ScrumBan;
- Quando não usar Métodos Ágeis;
- Ferramentas utilizadas para Metodologias Ágeis e funções de cada um no time.

Competências: Capacitar os alunos(as) para prática utilizando ferramentas de frameworks de Agilidade. Demostrar a importância de cada membro no time de acordo com suas funções.

Estratégias de ensino	Atividade desenvolvida	Recursos
Aula expositiva dialogada sobre os tópicos abordados em Processos e Agilidade II.	Exercício prático com o modelo Scrum e Kanban.	Jira, Azure DevOps, Trello ou similares. Canvas

15 – Projeto de Software II

Objetivos de aprendizagem: Competências, habilidades, atitudes e comunicação.

Objetivos de aprendizagem:

- Coesão;
- Acoplamento;
- SOLID;
- Métricas de Código Fonte.

Competências: Capacitar os alunos(as) para compreensão, análise e implementação de um sistema que atenda aos requisitos funcionais e não funcionais, assim como garantir a qualidade e integridade da solução/software definidos pelo cliente.

Estratégias de ensino	Atividade desenvolvida	Recursos
Aula expositiva dialogada sobre os tópicos abordados em Projeto de Software II.	Exercícios sobre o tema da aula com caso de uso sobre os tópicos abordados.	Exercícios sobre os temas da aula. Canvas.

16 – Testes de Software

Objetivos de aprendizagem: Competências, habilidades, atitudes e comunicação.

Objetivos de aprendizagem:

- Introdução;
- Cobertura de Testes;
- Testes unitários (caixa-branca);
- Testes funcionais (caixa-preta);
- Testes caixa-cinza;
- Testes não funcionais;
- Smoke Test, BDD;
- Ferramentas de Testes.

Competências: Capacitar os alunos(as) a compreender e aplicar conceitos de Testes de Software, para garantir que o software tenha qualidade, segurança e integridade dentre outros requisitos funcionais e não funcionais. Compreender a identificação de bugs em fases inicias dos projetos.

Estratégias de ensino	Atividade desenvolvida	Recursos
Aula expositiva dialogada sobre os tópicos abordados em Testes de Software.	Desenvolvimento de atividades práticas básicas sobre Testes de Software.	InteliJ ou VSCode, JUnit, Zephyr, TestLink, Cucumber, GIT. Canvas

17 – Arquitetura

Objetivos de aprendizagem: Competências, habilidades, atitudes e comunicação.

Objetivos de aprendizagem:

- Introdução;
- Arquitetura em Camadas;
- Arquitetura MVC;
- Arquitetura com Microsserviços;
- DDD (Domain-Driven-Design).

Competências: Capacitar os alunos(as) a compreender e aplicar conceitos de arquitetura na disciplina de Engenharia de Software.

Estratégias de ensino	Atividade desenvolvida	Recursos
Aula expositiva dialogada sobre os tópicos abordados em Arquitetura.	Desenvolvimento de atividades práticas básicas sobre arquitetura.	InteliJ ou VSCode, Draw.io GIT. Canvas

18 – Padrões de Projetos

Objetivos de aprendizagem: Competências, habilidades, atitudes e comunicação.

Objetivos de aprendizagem:

- Introdução;
- Design Patterns;
- Fábrica;
- Singleton;
- Strategy;
- Observador;
- Quando não usar Padrões de Projeto.

Competências: Capacitar os alunos(as) a compreender e aplicar conceitos de Padrões de Projetos, pois cada projeto descreve um problema que sempre ocorre em nosso contexto e uma solução para ele, de forma que possamos usá-la um milhão de vezes.

Estratégias de ensino	Atividade desenvolvida	Recursos
Aula expositiva dialogada sobre os tópicos abordados em Padrões de Projetos.	Desenvolvimento de atividades práticas básicas sobre Padrões de Projetos.	InteliJ ou VSCode, Spring MVC, Draw.io e GIT. Canvas

19 – Refactoring

Objetivos de aprendizagem: Competências, habilidades, atitudes e comunicação.

Objetivos de aprendizagem:

- Introdução;
- Extração de Classes;
- Refactorings Automatizados;
- Code Smells;
- Código Duplicado.

Competências: Capacitar os alunos(as) a compreender e aplicar conceitos de Refatoração. Pois quando um bug é detectado, temos que realizar uma manutenção corretiva. Quando os usuários ou o dono do produto solicitam uma nova funcionalidade, temos que realizar uma manutenção evolutiva. Quando uma regra de negócio ou alguma tecnologia usada pelo sistema muda, temos que reservar tempo para uma manutenção adaptativa.

Estratégias de ensino	Atividade desenvolvida	Recursos
Aula expositiva dialogada sobre os tópicos abordados em Refactoring.	Exercícios sobre o tema da aula com exercícios básicos sobre Refatoração.	InteliJ, Java, VSCode

Avaliação	Descrição	Peso média final
AC1	Avaliação Continuada 1 Média da nota das atividades desenvolvidas conforme Plano de Aulas ao longo do período	15%
AC2	Avaliação Continuada 2	30%
	Média da nota das atividades desenvolvidas conforme Plano de Aulas ao longo do período	
AG	AG – Avaliação Geral, De acordo com as orientações institucionais	10%
AF	Avaliação Final Prevista no calendário institucional avaliação individual envolvendo todo o conteúdo da componente curricular.	45%

ANEXO E – PLANOS DE ENSINO - PE_IES D

Disciplina: Engenharia de Software

Ementa:
Introdução à Engenharia de Software,
Paradigmas,
Visão Geral de Metodologias,
Metodologias
OO,
Análise do Sistema (Diagrama Use Case),
Planejamento do Projeto de Software (Métricas, Estimativas, Plano de Projeto),
Análise de Requisitos (DEF+DES),
Projeto de Software (Diagramas de
Classes, Interação, Atividades, Transição de Estados),
Codificação (Linguagens e Tendências),
Teste de Software (Casos de Teste),
Manutenção de Software (Correção, Melhorias, Adaptação),
Transição/Implantação de Software (Processos ITIL).

Bibliografia Básica:
PRESSMAN, R. S. Engenharia de Software. SP: Makron Books, 2016. 16
SOMMERVILLE, IAN - Engenharia De Software- Editora PEARSON, 2007. 13
TONSIG, S. Engenharia de Software: Análise e Projeto de Sistemas. Ciência Moderna, 2008. 6
PAULO FILHO, Wilson De. Engenharia De Software. Editora: LTC, 2009.
RUMBAUGH, James; BLAHA, Michael. Modelagem e Projetos Baseados em Objetos com UML 2. Rio de
Janeiro: Campus, 2006. 8

Bibliografia complementar:
BEZERRA, E. Princípios de análise e projeto de sistemas com UML. 2ª Ed. Rio de Janeiro: Campus. 2007.
GANE & SARSON. Análise Estruturada de Sistemas. Rio de Janeiro: LTC, 1995. 2
LAUDON, K. C. LAUDON, J.P. Sistemas de informação gerenciais. Rio de Janeiro: LTC, 1999. 6
NOGUEIRA, M. - Engenharia De Software - Editora Ciência Moderna, 2009. 5
WIXOM, Barbara Haley; DENNIS, Alan. Análise e Projeto de Sistemas .2ª Ed. TLC, 2005. 3
REZENDE, Denis Alcides. Engenharia de Software e Sistemas de Informação. Rio de Janeiro: Brasport,
1999. 4

ANEXO F – PLANOS DE ENSINO - PE_IES E

DISCIPLINA: ENGENHARIA DE SOFTWARE - 158166
UNIDADE DE ENSINO 1: FUNDAMENTOS DE ENGENHARIA DE SOFTWARE
COMPETÊNCIAS: Conhecer os fundamentos da Engenharia de Software e da metodologia tradicional de desenvolvimento de produtos de software

RESULTADO DE APRENDIZAGEM:
Aplicar fundamentos de engenharia de software, metodologias ágeis e controle de versões no processo de Desenvolvimento

SEÇÃO 1: Introdução à engenharia de software
(Teórico)
CONTEÚDO: Aspectos gerais, objetivos, evolução do software e crise do software.
Mitos do desenvolvimento de software. Modelos e evolução de processos de desenvolvimento de software e ciclo de vida.
Princípios de engenharia de software.
Tipos de software sob o ponto de vista da engenharia.

SEÇÃO 2: Metodologias ágeis
(Teórico)
CONTEÚDO: Aplicações e ferramentas de metodologias ágeis
Extreme Programming (XP): valores, práticas e suas contraindicações
Introdução a metodologias ágeis e comparações com a tradicional
Metodologia Scrum e suas características

SEÇÃO 3: Controle de versões
(Teórico)
CONTEÚDO: Envio de Versões
Ferramentas para controle de versão
Repositório e política de compartilhamento de itens
Versões de itens de configuração de software. Baseline e release.

UNIDADE DE ENSINO 2: QUALIDADE DE SOFTWARE
COMPETÊNCIAS: Conhecer e compreender processos de qualidade de software

RESULTADO DE APRENDIZAGEM:
Identificar e aplicar diferentes normas de qualidade de produto e conceitos de qualidade de processo

SEÇÃO 1: Introdução à qualidade de software
(Teórico)
CONTEÚDO: Benefícios de processos de qualidade de software
Conceitos de Qualidade de software. Erros, falhas, defeitos, qualidades e bugs
Garantia de qualidade de software - Elementos de garantia, características do produto, SQA. Métricas de qualidade

SEÇÃO 2: Qualidade de produto
(Teórico)
CONTEÚDO: Gestão da qualidade do produto

Medição, requisito e avaliação da qualidade
Modelos de qualidade
Normas ISO de qualidade

SEÇÃO 3: Qualidade de processo
(Teórico)
CONTEÚDO: Melhorias de processo de Software brasileiro - MPS.BR
Melhorias de processos individuais e de equipes - PSP
Modelos de maturidade - CMM e CMMI
Normas ISO de qualidade de processos

UNIDADE DE ENSINO 3: TESTES DE SOFTWARE
COMPETÊNCIAS: Saber identificar os tipos de testes e ferramentas case
RESULTADO DE APRENDIZAGEM:
Aplicar os diferentes tipos de testes de software para cada etapa de desenvolvimento

SEÇÃO 1: Conceitos de testes de software
(Teórico)
CONTEÚDO: Casos de testes
Depuração
Plano de testes
Validação, verificação e testes de software

SEÇÃO 2: Tipos de teste
(Teórico)
CONTEÚDO: Testes de aplicações orientadas a objetos
Testes de aplicações web e móveis
Testes de funcionalidade e funcionais
Testes estruturais e suplementares

SEÇÃO 3: Desenvolvimento orientado a testes e ferramentas case
(Teórico)
CONTEÚDO: Desenvolvimento orientado a testes (TDD)
Ferramentas case para testes de software
Gerenciamento de testes
Testes automatizados de software

UNIDADE DE ENSINO 4: AUDITORIA DE SISTEMAS
COMPETÊNCIAS: Conhecer e os fundamentos de auditoria de sistemas, bem como a manutenção e evolução de software
RESULTADO DE APRENDIZAGEM:
Analisar e avaliar o contexto de aplicação da auditoria de sistemas de informação e manutenção e evolução de software

SEÇÃO 1: Fundamentos de auditoria de sistemas
(Teórico)
CONTEÚDO: Abordagens da auditoria de sistemas de informação - ao redor, através e com o computador
Ciclos de vida da auditoria

Conceitos de auditoria de tecnologia da informação - Controle geral e controle interno
Tipos de auditoria

SEÇÃO 2: Auditoria de sistemas de informação
(Teórico)
CONTEÚDO: Auditoria durante o ciclo de desenvolvimento de sistema
Controles de aplicativos
Controles de software de sistema
Controles gerais da auditoria de sistemas de informação

SEÇÃO 3: Manutenção e evolução de software
(Teórico)
CONTEÚDO: Classificação e tipos de atividades de manutenção de software
Engenharia Reversa e Reengenharia
Necessidade de evolução e manutenção de software
Processos e ferramentas de manutenção de software