



**INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
DO RIO GRANDE DO NORTE
CAMPUS CURRAIS NOVOS**

LUIZ PAULO DE LIMA ARAÚJO

TITULO DO TRABALHO: SUBTÍTULO

**CURRAIS NOVOS - RN
2025**

LUIZ PAULO DE LIMA ARAÚJO

TITULO DO TRABALHO: SUBTÍTULO:

Trabalho de conclusão de curso apresentado ao curso de graduação em Tecnologia em Sistemas para Internet, como parte dos requisitos para obtenção do título de Tecnólogo em Sistemas para Internet pelo Instituto Federal do Rio Grande do Norte.

Orientador(a): Orientador do Trabalho.

Co-orientador(a): .

CURRAIS NOVOS - RN

2025

LUIZ PAULO DE LIMA ARAÚJO

TITULO DO TRABALHO: SUBTÍTULO

Trabalho de conclusão de curso apresentado ao curso de graduação em Tecnologia em Sistemas para Internet, como parte dos requisitos para obtenção do título de Tecnólogo em Sistemas para Internet pelo Instituto Federal do Rio Grande do Norte.

CURRAIS NOVOS - RN, xx de mmm de 2025

Orientador do Trabalho

Orientador

Professor

Examinador(a) 1

Professor

Examinador(a) 2

CURRAIS NOVOS - RN

2025

Eu dedico esse trabalho à ...

*“Arquitecti est scientia
pluribus disciplinis et variis
eruditionibus ornata, quae ab ceteris artibus
perficiuntur. Opera ea nascitur et fabrica
et ratiocinatione.*

(DE ARCHITECTURA, Liber primus, Caput Primus, signum paragraphi I)

RESUMO

O resumo tem a função de resumir os pontos-chave da monografia, apresentando sucintamente a introdução, metodologia, resultados, discussão e conclusões, além de destacar a relevância do estudo. Deve ser conciso, informativo e atrativo, com uma extensão usualmente entre 150 e 300 palavras, dependendo das diretrizes da instituição ou da revista acadêmica.

Palavras-chave:

ABSTRACT

The abstract serves the purpose of summarizing the key points of the thesis, briefly presenting the introduction, methodology, results, discussion, and conclusions, while highlighting the study's relevance. It should be concise, informative, and engaging, typically ranging from 150 to 300 words, depending on the guidelines of the institution or academic journal.

Keywords:

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

DT Débito Técnico

IA Inteligência Artificial

TI Tecnologia da Informação

TSI Tecnologia em Sistemas para Internet

IEEE Institute of Electrical and Eletctronics Engineers

IFRN Instituto Federal do Rio Grande do Norte

ABES Associação Brasileira das Empresas de Software

LISTA DE SÍMBOLOS

Γ Letra Grega Gamma

LISTA DE ALGORITMOS

SUMÁRIO

1	INTRODUÇÃO	14
1.1	Contextualização e Problema	15
1.2	Objetivos	16
1.3	Justificativa	16
1.4	Apresentação do Trabalho	17
2	FUNDAMENTAÇÃO TEÓRICA	18
2.1	Trabalhos Relacionados	18
3	METODOLOGIA	19
4	DESENVOLVIMENTO DA PESQUISA	20
4.1	Proposta de Solução	20
4.2	Experimentos	20
5	RESULTADOS	21
6	CONCLUSÃO	22
6.1	Discussão	22
6.2	Contribuições	22
6.3	Limitações	22
6.4	Trabalhos Futuros	22
	REFERÊNCIAS	23

1 INTRODUÇÃO

A computadores eletrônicos e o software surgiram como ferramenta de importância reconhecida notoriamente durante os eventos finais da segunda grande guerra. Inicialmente tinha-se como foco utilitário a quebra de criptografia da comunicação inimiga e o cálculo da trajetória de mísseis balísticos.

Poucos anos após o fim dos conflitos armados houve o desenvolvimento contínuo não somente da tecnologia física como também da lógica (*software*). Programas, antes escritos em binário agora podiam ser criados em linguagens próximas da linguagem humana.

O trabalho da cientista Grace Murray Hopper, década de 1950, concebeu a ideia do compilador, programa capaz de transcrever/traduzir uma linguagem lógica mais abstrata e inteligível ao homem para o código de máquina executável pelo aparato físico computacional.

Na década 1960 a Ciência da Computação consolidara-se, de fato, como uma ciência formal e de importância notória para as décadas futuras. Sua importância não mais era militar mas também civil em áreas como negócios, administração e demais áreas das ciências exatas e da natureza.

Via-se um linear e vertiginoso aumento do poder computacional ao mesmo tempo em que diminuía-se as dimensões dos computadores. Gordon Moore, fundador da Intel, publicara na revista *Electronics Magazine*, em 1965, um artigo, onde previra o dobro do aumento do poder computacional a cada 18 meses.

Entretanto, como bem compreendido na historiologia tecnológica humana, nenhum avanço é isento de problemas e crises. Durante as décadas de 1960 e 1970 houve um grande aumento na necessidade por software por parte de organizações governamentais e privadas. Todavia, processos e técnicas praticadas no desenvolvimento conhecidas na época não eram eficazes o suficiente em atender as necessidades por soluções de complexidade crescente.

O cenário de crise possuía como principais características: Complexidade crescente das soluções exigidas e imaturidade técnico-metodológica. Apresentava-se como consequências negativas de tais características, o(a) :

- Estouro de prazos previamente estipulado
- Aumento de custo muito além do planejado
- Agravamento da inconsistência entre o que era exigido e o que era provido
- Expansão da ingerenciabilidade de projetos em pouco tempo de manutenção
- Crescimento da inqualidade do software apresentando falhas, bugs e

Apesar da crise em si nunca ter sido absolutamente resolvida de fato, esforços oriundos da experiência de vários profissionais e pesquisadores competentes resultaram na consolidação de obras técnicas com orientações fundamentais proporcionantes da solução mitigatória sobre tal realidade. Obras como:

- *Software Engineering: A Report on a Conference Sponsored by the NATO Science Committee*
- *The Mythical Man-Month: Essays on Software Engineering - Frederick P. Brooks Jr.*
- *Refactoring: Improving the Design of Existing Code - Martin Fowler*
- *Structured Programming - Dijkstra*
- *Design Patterns: Elements of Reusable Object-Oriented Software - Gamma, Helm, Johnson, Vlissides*
- *Clean Code - Robert C. Martin*

brilham dentre os esforços escritos.

Dentre os problemas enfrentados pela ciência da computação e engenharia de software tem-se o da dependência. O software, como solução, surge, da mesma maneira que a pesquisa científica, sobre obras anteriores pois sua construção faz-se de modo a reaproveitar códigos (soluções) desenvolvidas anteriormente por outros programadores. Conhece-se tais peças componentes, no âmbito técnico, como: trechos, módulos, bibliotecas e frameworks.

Apesar das dependências não serem um problema quando cuidadosamente gerenciadas e limitadas há, durante o desenvolvimento, o risco real de um projeto cair no clássico problema do inferno das dependências, situação em que o software tem seu funcionamento correto comprometido devido a peças de software dependidas estarem sob efeito de um ou vários dos seguintes problemas:

- Versões diferentes de uma mesma dependência exigida
- Dependências circulares onde A depende de B e B depende de A
- Emprego de dependências desatualizadas
- Esquema de instalação complexo

Problemas correlacionados às dependências são abundantes em projetos de software de todas as dimensões e resolvê-los não é uma das mais fáceis tarefas. Por conseguinte, haver trabalhos no âmbito acadêmico que contribuam copiosamente em boas demonstrações é um esforço necessário e útil.

1.1 Contextualização e Problema

O desenvolvimento de software atualmente é impelido, na maioria das organizações, por times ágeis auto gerenciados auxiliados pela maravilha desta terceira década: A inteligência artificial generativa. Em tal contexto de auxílio constante algoritmos não determinísticos pode-se acompanhar o desenrolar de um cenário novo

onde a qualidade dos sistemas desenvolvidos são, como nunca visto antes, fortemente influenciadas tais ferramentas levando a um desfecho ainda incerto.

Tem-se como certo que problemas de qualidade implementacional originadas pela terceirização indiscriminada de boa parte da responsabilidade de pensar e agir em desenvolvendo um software à Inteligência Artificial (IA) podem levar ao reagravamento da nunca propriamente resolvida crise do software das décadas de 1960 e 1980.

Nunca foi tão importante como agora a necessidade pela plurificação de trabalhos que visem reforçar a exemplificação do emprego de conhecimentos filosófico-técnicos que interfiram positivamente na entrega de valor do software moderno.

Tendo em vista a realidade corrente, determina-se como problema cerne desta pesquisa, a irmos ao campo da implementação de soluções comerciais com ferramentas de terceiros, o seguinte problema:

Há a possibilidade de desenvolver um sistema cujas regras de negócio são independentes de soluções técnicas a vista de obras filosófico-técnicas atuais ? Quais são os benefícios ? Quais são as adversidades ?

1.2 Objetivos

Este trabalho tem como objetivo geral determinar os efeitos, adversos e ou benéficos, de se implementar software cujas classes possuidoras de regras de negócio, no paradigma orientado a objetos, possuam independência, do código servidor provido na forma de bibliotecas ou framework, através de técnicas expostas em obras técnicas populares.

Seus objetivos específicos são atingidos ao:

- Revisar obras que expõem métodos pro mitigação de dependências
- Modelar regras de negócios independentes de soluções de terceiros
- Compreender benefícios e adversidades

1.3 Justificativa

Este trabalho justifica-se como sendo, no âmbito acadêmico, um esforço experimental para aumento do catálogo de obras que focam na mitigação ou solução de problemas do software por meio de conhecimento de engenharia construídos por outrem.

Quanto ao âmbito pessoal/profissional, este trabalho é uma forma de aprimorar os conjuntos de conhecimentos técnico-arquiteturais do docente, permitindo-lhe um acesso mais seguro ao mercado de trabalho de TI em implementações comercialmente viáveis.

Trata-se do emprego do conhecimento pré-existente sobre técnicas implementacionais em um sistema real de modo a determinar os limites do que se pré-existe e é tido como bom conhecimento edificado.

1.4 Apresentação do Trabalho

A introdução contextualiza brevemente o leitor à história da ciência da computação direcionando-o a um fato omnitemporal dessa ciência: as dependências e seus problemas potenciais.

A fundamentação teórica introduzirá o leitor aos fundamentos do paradigma de programação utilizado no projeto experimental e, em seguida, a seções de obras técnicas populares entre programadores juntando métodos conhecidos e padronizados com potencial para mitigar ou eliminar a dependência de um software de outro.

A metodologia apresentará como o objeto experimentado foi concebido na forma de uma explanação não enlongada das etapas de: levantamento de requisitos e problema de negócio, modelagem, implementação, testes, implantação. De forma a orientar como possíveis pesquisas posteriores organiza-se-ão.

Os resultados

A conclusão ou considerações finais, advindos da experiência obtida pela execução do projeto, apresentar-se-ão na forma explanativa separativamente o que foi praticável do que não foi e apontando possíveis armadilhas das dependências escolhidas.

2 FUNDAMENTAÇÃO TEÓRICA

Esta seção apresenta partes significativas das obras dos autores que contribuíram com o a engenharia de software compilando as mais contrapõem-se como solução a problemas relacionados aos males da forte dependência.

Quando fala-se de arquitetura, no contexto de desenvolvimento de software, fica claro que o objeto de estudo manifesta-se como um conjunto de códigos.

2.1 Trabalhos Relacionados

3 METODOLOGIA

Legere iste nunc non oportet.

4 DESENVOLVIMENTO DA PESQUISA

4.1 Proposta de Solução

4.2 Experimentos

5 RESULTADOS

Neste capítulo são expostos os resultados de sua pesquisa. No caso do TCC I, resultados esperados ou parciais, para TCC II, resultados "finais".

6 CONCLUSÃO

6.1 Discussão

6.2 Contribuições

6.3 Limitações

6.4 Trabalhos Futuros

REFERÊNCIAS