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

Os computadores eletrônicos e o *software* emergiram como ferramenta de importância notória durante os eventos finais da segunda grande guerra. Seu emprego possibilitara computação de trajetórias balisticas, propriedades das detonações atômicas e quebra de de criptografia inimiga (Dodig-Crnkovic, 2001, p.4).

Poucos anos após o fim do conflito, na década de 1950, houve o desenvolvimento contínuo não somente da tecnologia física como também da lógica no *software*. Programas, antes escritos em binário, agora podiam ser criados em linguagens cada vez mais próximas da linguagem humana (Dodig-Crnkovic, 2001, p.5).

Na década 1960 a Ciência da Computação consolidara-se, de fato, como uma ciência formal e de importância notoria para as décadas futuras, como bem demonstrado no breve artigo (Newell; Perlis; Simon, 1967). Sua importância não mais retringia-se acadêmico-militar mas, agora, influenciava positivamente aos grandes negócios, instituições públicas e universidades.

Via-se um linear e vertiginoso aumento do poder computacional ao mesmo tempo em que diminuia-se as dimensões dos computadores. De acordo com Almeida (1967) o fundador da Intel, Gordon Moore, publicara na revista *Electronics Magazine* , um artigo, onde previra aumento em dobro do poder computacional a cada 18 meses.

Entretanto, nenhum avanço é insento de problemas e crises. Durante a décadas de 1960 e 1970 o grande aumento na necessidade por *software* por organizações comerciais, públicas e de ensino impeliram o agravamento da chamada "Crise do Software". O cenário teve como característica principal, além do aumento constante do poder computacional (Dijkstra, 1972, p.3), a ineficácia de processos e técnicas empregadas na implementação do *software*, isso em um cenário onde aplicações cada vez mais complexas eram exigidas como apresentado em resumo por (Hinchey, 2018).

O cenário problemático de outrora trouxera efeitos indesejáveis em todos os âmbitos do empreendimento de se escrever *software*. Complexidade crescia enquanto a imaturidade técnico-metodologica firmava-se como fator propulsor. Apresentava-se, como consequências negativas de tais malefeitos, o(a) :

- Estouro de prazos previamente estipulados.
- 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.
- Queda de qualidade rápida da base de código fonte.

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 que propõem soluções efetivas. Destacam-se:

- *Software Engineering: A Report on a Conference Sponsored by the NATO Science Committee*
- *The Mytical Man-Month: Essays on Software Engineering - Frederick P. Brooks Js.*
- *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*

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 anterior 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 diferente 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 corelacionados 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 util.

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 auxilio constante algoritmos não deterministicos 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 beneficios ? 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 especificos são atingidos ao:

- Revisar obras que expoem 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 construidos 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 viaveis.

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, implemantação, 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 contribuiram com o a engenharia de software compilando as mais contrapôem-se como solução a problemas relacionados aos males da forte dependência nos sistemas.

Antes de avançar sobre conceitos arquiteturais e de padrões de projetos, deve-se, primeiro, observar aos fundamentos intrínsecos do paradigma sobre o qual dispõe-se a trabalhar sobre.

O conceito de paradigma é introduzido por Floyd (2007) como sendo: "um padrão, um exemplo com o qual as coisas são feitas". O mesmo autor deixara claro, ao discutir acerca dos paradigmas de sua época, que o conceito, no âmbito do desenvolvimento de software, consiste na forma como programas são feitos.

Os paradigmas surgiram concomitantemente com o desenvolvimento de liguagens de programação, em especial nas de alto nível que abstraiam a implementação binária direta de instruções, possibilitando uma implementação mais humana e menos complexa (Sammet, 1969, p. 8-).

Inicialmente há o surgimento, com o desenvolvimento da arquitetura de Von Neumann, do primeiro paradigma, o Imperativo que trazia os conceitos fundamentais de estado e ação modificante do estado. Sua influência é, até hoje, enorme e serviu de núcleo para paradigmas posteriores (Jungthon; Goulart, 2009, p. 1).

Com o aumento de complexidade dos sistemas, surge o paradigma estruturado que definia a sequência, iteração e decisão como sendo as partes fundamentais de qualquer programa implementável (Dahl; Dijkstra; Hoare, 1972).

Por fim, a orientação a objetos, inicialmente implantadas nas linguagens Simula(1962) e Smalltalk(1972), traz os conceitos de objeto como uma abstração de qualquer elemento da vida real que interage separadamente com outros por meio de "mensagens"(Rentsch, 1982, p. 52-55).

O paradigma orientado a objetos originou, de acordo com a síntese de outros autores presente em Kasture e Jaiswal (2019), quatros pilares fundamentais sobre a implementação definidos como:

- Abstração capacidade de representar um subconjunto de atributos e comportamentos de uma entidade real
- Encapsulamento controle de acesso externo a atributos e comportamentos privados de um objeto
- Herança capacidade de extensão por superconjunto de atributos e comportamentos
- Polimorfismo capacidade de mutação de comportamentos por sobrescrita

O paradigma orientação a objetos possibilitou o desenvolvimento de arquiteturas e padrões de projetos largamente documentados por vários autores.

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

ALMEIDA, R. B. Evolução dos processadores. *Evolução dos processadores*, unicamp, 1967.

DAHL, O. J.; DIJKSTRA, E. W.; HOARE, C. A. R. (Ed.). *Structured programming*. GBR: Academic Press Ltd., 1972. ISBN 0122005503.

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

DODIG-CRNKOVIC, G. History of computer science. *Västerås: Mälardalen University*, 2001.

FLOYD, R. W. The paradigms of programming. In: *ACM Turing award lectures*. [S.l.: s.n.], 2007. p. 1978.

HINCHEY, M. Software crisis 2.0. In: ____. *Software Technology: 10 Years of Innovation in IEEE Computer*. [S.l.: s.n.], 2018. p. 1–16.

JUNGTHON, G.; GOULART, C. M. Paradigmas de programação. *Monografia (Monografia)—Faculdade de Informática de Taquara, Rio Grande do Sul*, v. 57, 2009.

KASTURE, D.; JAISWAL, R. C. Pillars of object oriented system. *Int. J. Res. Appl. Sci. Eng. Technol.*, v. 7, n. 12, p. 589–590, 2019.

NEWELL, A.; PERLIS, A. J.; SIMON, H. A. What is computer science? *Science*, American Association for the Advancement of Science, v. 157, n. 3795, p. 1373–1374, 1967.

RENTSCH, T. Object oriented programming. *ACM Sigplan Notices*, ACM New York, NY, USA, v. 17, n. 9, p. 51–57, 1982.

SAMMET, J. E. *Programming Languages: History and Fundamentals*. 1st. ed. Englewood Cliffs, NJ: Prentice-Hall, 1969. ISBN 0137300051.