INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DA PARAÍBA CAMPUS CAJAZEIRAS
CURSO SUPERIOR DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

Análise de Complexidade de Código com Cognitive-Driven Development:

Um Estudo sobre a Carga Cognitiva no Desenvolvimento de Software

Izaquiel Canuto da Silva

Cajazeiras

2024

Izaquiel Canuto da Silva

Análise da Complexidade de Código com Cognitive-Driven Development: Um Estudo sobre a Carga Cognitiva no Desenvolvimento de Software

Trabalho de Conclusão de Curso apresentado junto ao Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas do Instituto Federal de Educação, Ciência e Tecnologia da Paraíba - Campus Cajazeiras, como requisito à obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas.

Orientador: Prof. Me. Afonso Serafim Jacinto

Cajazeiras

2024

IFPB / Campus Cajazeiras
Coordenação de Biblioteca
Biblioteca Prof. Ribamar da Silva
Catalogação na fonte: Cícero Luciano Félix CRB-15/750

S586a Silva, Izaquiel Canuto da.

Análise de Complexidade de Código com Cognitive-Driven Development : um Estudo sobre a Carga Cognitiva no Desenvolvimento de Software / Izaquiel Canuto da Silva. – 2024.

52f. : il.

Trabalho de Conclusão de Curso (Tecnólogo em Análise e Desenvolvimento de Sistemas) - Instituto Federal de Educação, Ciência e Tecnologia da Paraíba, Cajazeiras, 2024.

Orientador(a): Prof. Me. Afonso Serafim Jacinto.

1. Desenvolvimento de sistemas. 2. Engenharia de software. 3. Carga cognitiva. 4. Complexidade de código. I. Instituto Federal de Educação, Ciência e Tecnologia da Paraíba. II. Título.

IFPB/CZ CDU: 004.4(043.2)

MINISTÉRIO DA EDUCAÇÃO
SECRETARIA DE EDUCAÇÃO PROFISSIONAL E TECNOLÓGICA
INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DA PARAÍBA

IZAQUIEL CANUTO DA SILVA

ANÁLISE DA COMPLEXIDADE DE CÓDIGO COM COGNITIVE-DRIVEN DEVELOPMENT: UM ESTUDO SOBRE A CARGA COGNITIVA NO DESENVOLVIMENTO DE SOFTWARE

Trabalho de Conclusão de Curso apresentado junto ao Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas do Instituto Federal de Educação, Ciência e Tecnologia da Paraíba - Campus Cajazeiras, como requisito à obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas.

Orientador

Prof. Me. Afonso Serafim Jacinto

Aprovada em: **17 de Outubro de 2024.**

Prof. Me. Afonso Serafim Jacinto - Orientador

Prof. Me. Francisco Paulo de Freitas Neto - Avaliador

IFPB - Campus Cajazeiras

Prof. Me. Michel da Silva

IFPB - Campus Cajazeiras

Documento assinado eletronicamente por:

- **Francisco Paulo de Freitas Neto**, **PROFESSOR ENS BASICO TECN TECNOLOGICO**, em 18/10/2024 09:51:06.
- **Afonso Serafim Jacinto**, **PROFESSOR ENS BASICO TECN TECNOLOGICO**, em 18/10/2024 11:18:02.
- **Michel da Silva**, **PROFESSOR ENS BASICO TECN TECNOLOGICO**, em 18/10/2024 18:43:56.

Este documento foi emitido pelo SUAP em 18/10/2024. Para comprovar sua autenticidade, faça a leitura do QRCode ao lado ou acesse https://suap.ifpb.edu.br/autenticar-documento/ e forneça os dados abaixo:

Código Verificador: 621784
Código de Autenticação: 4eecafb837

A meu pai Francisco Felix, a minha mãe Maria do Socorro, a minha avó Francisca, ao meu avô Manoel e a minha avó Francisca Ricardo.

AGRADECIMENTOS

A minha mãe Maria do Socorro, ao meu pai Francisco Felix, a meus avós que me apoiaram em toda minha trajetória e sempre me fizeram prosseguir por um caminho com muitos altos e baixos.

A todos meus familiares que me deram forças.

Aos meus amigos que nunca me deixaram desanimar e nem desistir dos meus objetivos sempre me dando forças, principalmente os que sabem de toda minha história de vida e minhas dificuldades.

A todos meus colegas de curso que fizeram parte da minha trajetória.

E lógico, a todos os meus professores e funcionários do campus , que são muito mais que professores, constituíram uma parte importante da minha história, fazendo com que eu adotasse a faculdade como uma segunda casa, me dando aquela força nas dificuldades.

Agradeço a todos por fazerem parte dessa minha jornada

Gostaria de descrever uma lista com todas as pessoas que me permitiram chegar aonde cheguei com cada nome e o significado importante que essas pessoas deram a minha vida, mas já que tantas pessoas especiais compuseram minha lista deixo meu agradecimento mais do que especial a todos que mesmo não sendo citados sabem que estão fazendo parte da minha história hoje e sempre.

”Não importa o tempo que leve para conquistar seus objetivos, o que realmente importa é a trajetória que tornou isso válido e as lições que isso trouxe .”

Izaquiel C. Silva

RESUMO

Uma solução que a engenharia de *software* utiliza para reduzir a complexidade intrínseca no desenvolvimento é o *Cognitive Driven Development* (CDD), que é uma estratégia utilizada para reduzir a sobrecarga cognitiva durante o desenvolvimento ao aprimorar o *design* do código. No entanto, os conceitos de compreensão de cada indivíduo se diferenciam, o que implica que, para que possamos obter uma relevância significativa de entendimento entre desenvolvedores distintos que possuem um grau de compreensão distinto é necessário avaliar sua experiência. Neste contexto, este trabalho tem como foco uma investigação a respeito do conceito e aplicação do CDD e avaliar a experiência do desenvolvedor juntamente com o CDD para ter maior relevância ao reduzir a carga cognitiva para indivíduos distintos.

Palavras-chave: Cognitive Driven Development; Carga Cognitiva; Commit.

ABSTRACT

One solution that software engineering uses to reduce the intrinsic complexity in development is Cognitive Driven Development (CDD), which is a strategy used to reduce cognitive overload during development by improving code design. However, the concepts of understanding of each individual differ, which implies that, in order to obtain a significant relevance of understanding among different developers who have a different degree of understanding, it is necessary to evaluate their experience. In this context, this work focuses on an investigation regarding the concept and application of CDD and evaluates the developer experience together with CDD to have greater relevance in reducing the cognitive load for different individuals.

Keywords: Cognitive Driven Development; Cognitive Load; Commit.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE QUADROS

LISTA DE CÓDIGOS

LISTA DE ABREVIATURAS E SIGLAS

AST	Abstract Syntax Tree
CBO	Coupling Between Objects
CDD	Cognitive-Driven Development
CK	Calculates Metrics
ICC	Individual CDD Commit
ICP	Intrinsic Complex Point
IDE	Integrated Development Environment
LCOM	Lack of Cohesion of Methods
LOC	Lines of Code
RFC	Response For a Class
WMC	Weighted Methods per Class

SUMÁRIO

1 INTRODUÇÃO

Explorar as limitações do entendimento humano e o funcionamento da mente são temas investigados há várias décadas. Um dos artigos mais influentes nesta área foi publicado em 1956 pelo psicólogo George A. Miller, da Universidade de Harvard. Seu artigo, intitulado *The Magical Number Seven, Plus or Minus Two: Some Limits on our Capacity for Processing Information* (em português, “O número mágico sete, mais ou menos dois: algumas limitações de nossa capacidade de processar informações”) propôs a teoria de que a capacidade da memória de curto prazo é limitada a aproximadamente sete itens, com uma variação de mais ou menos dois (SOUZA, 2021).

Miller (1956) argumenta que, independentemente das diversas formas e contextos em que a informação é apresentada, a capacidade de armazenamento da mente humana permanece relativamente constante. Ele propõe que a limitação se deve ao número de "pedaços" de informação que conseguimos manter simultaneamente na memória de curto prazo. Esta teoria tem sido fundamental para o avanço do entendimento sobre os processos cognitivos e continua a influenciar pesquisas e aplicações práticas na psicologia e áreas afins (SOUZA, 2021).

A Teoria da Carga Cognitiva, proposta por John Sweller em 1988, expande a compreensão das limitações da mente humana quando sugere que o *design* de instrução deve considerar as limitações da memória de trabalho para otimizar a aprendizagem. Essa teoria enfatiza a importância de minimizar a carga cognitiva extrínseca (esforço mental causado pela apresentação da informação) e maximizar as cargas cognitivas intrínseca (complexidade inerente ao conteúdo) e germinativa (processos mentais que facilitam a aprendizagem) para facilitar o processamento eficaz das informações (SWELLER, 1988).

Na área de desenvolvimento de *software*, a teoria da carga cognitiva encontra aplicação prática no *Cognitive Driven Development* (CDD), que é uma estratégia voltada para reduzir a complexidade intrínseca dos códigos-fonte durante o seu desenvolvimento. Esta abordagem engloba vários aspectos de estudos voltados para o aprendizado e compreensão humana, visando entender a cognição. Com o CDD, é possível obter uma medida da

complexidade intrínseca de um código, o que beneficia a engenharia de *software* (PEREIRA & PINTO, 2021).

A complexidade intrínseca refere-se à dificuldade inerente de um conteúdo ou tarefa, independentemente de como é apresentada. Essa complexidade é determinada pela quantidade de elementos interativos que precisam ser simultaneamente processados pela memória de trabalho. Compreender a Complexidade Intrínseca é crucial para adaptar o material instrucional de modo a não sobrecarregar a capacidade cognitiva de indivíduos (SWELLER, 2010).

Para entender o CDD, é importante compreender que a obtenção da complexidade envolve uma análise de métricas definidas com o objetivo de manter a carga cognitiva dos códigos em níveis aceitáveis. O conhecimento dos padrões de projeto é fundamental para alcançar códigos limpos, embora seja importante ressaltar que mesmo ao seguir esses padrões, a complexidade dos códigos nem sempre será reduzida (PEREIRA & PINTO, 2021).

Focando no conhecimento de soluções como o *Cognitive Driven Development* e na análise de métricas, este trabalho se concentra nos fatores que podem afetar diretamente a compreensão da complexidade. É importante considerar que cada indivíduo possui um nível de compreensão distinto, e este estudo leva em consideração essas diferenças para melhorar o desempenho na aplicação do CDD.

1.1 PROBLEMÁTICA

Esta pesquisa aborda um problema comum em desenvolvimento de *software*: a perda de produtividade dos desenvolvedores ao tentar entender o código de outros membros da equipe. Em grandes projetos e, principalmente, em empresas de desenvolvimento, isso se torna um fator crucial. O tempo gasto está diretamente relacionado à capacidade do indivíduo de compreender a forma como o projeto está organizado e a complexidade intrínseca do projeto em si. Em outras palavras, se o grau de complexidade da carga cognitiva for elevado, o indivíduo levará mais tempo para compreender e assimilar, ou até mesmo pode não compreender completamente.

Na engenharia de *software*, as boas práticas de programação são enfatizadas. No

entanto, a teoria muitas vezes difere da prática. Embora seja satisfatório levar em consideração os conceitos básicos e enfatizar os padrões de projeto ao escrever código, a falta de utilização desses padrões pode resultar em um alto grau de complexidade.

1.2 MOTIVAÇÃO

A comunidade de desenvolvedores frequentemente enfrenta perda de produtividade ao tentar compreender e assimilar os padrões pessoais de outros membros da equipe. Isso se torna especialmente evidente em projetos nos quais é necessário substituir ou realocar membros da equipe. Embora o uso frequente de padrões de projetos seja uma prática que geralmente resulta em projetos com níveis de complexidade positivos, não há garantia de aceitação constante.

O *Cognitive Driven Development*, ou Desenvolvimento Orientado por Cognição, surge como uma abordagem que auxilia na melhoria da aceitação em projetos com diversos níveis de complexidade, entregando resultados com graus de complexidade cognitiva aceitáveis. O objetivo do CDD é reduzir o impacto da carga cognitiva e facilitar o processo de assimilação de conhecimentos, demonstrando-se bem-sucedido nesse aspecto.

Entretanto, mesmo sendo uma solução promissora, o CDD enfrenta desafios, sendo um deles o fator experiência do programador. Este fator será explorado neste trabalho, visando compor uma solução viável que seja benéfica para todos os envolvidos.

1.3. TRABALHOS RELACIONADOS

Gustavo Pinto e Alberto Santiago (2023) avaliaram a eficácia do CDD na construção de uma ferramenta de gerenciamento de aprendizagem, utilizando a metodologia de observação de uma equipe de desenvolvimento da Zup Innovation[1] ao longo de um ano. Durante esse período, foram analisados rastros de *commits*, gravações de videochamadas e documentações da equipe para compreender o impacto do CDD no processo de desenvolvimento. Os resultados indicaram que o CDD auxiliou na manutenção de pequenas

[1] https://zup.com.br/

unidades de código, facilitando os testes. A técnica também se mostrou flexível, permitindo que os desenvolvedores identificassem os elementos de maior complexidade e realizassem refatorações de forma consciente.

Pereira et al. (2021), apresenta uma ferramenta chamada "*Cognitive Load Analyzer*" que apoia o Desenvolvimento Orientado à Cognição CDD ao reduzir a complexidade cognitiva do código. Para isso, os pesquisadores desenvolveram um *plugin* para IntelliJ IDEA, compatível com a linguagem Java, que facilita a adoção do CDD, auxiliando na divisão do código em partes mais gerenciáveis e menos complexas. Os principais resultados mostram que a ferramenta ajuda a manter a complexidade cognitiva dentro de limites viáveis, mesmo com a expansão do *software*, contribuindo para a melhoria da qualidade do código e facilitando a compreensão dos desenvolvedores. Estudos experimentais indicam que o CDD é um método promissor para alcançar código de alta qualidade.

Sousa e Pinto (2020) apresentam o CDD, uma abordagem que integra medições de complexidade cognitiva e a Teoria da Carga Cognitiva com o objetivo de aprimorar o processo de desenvolvimento de *software*. Utilizando-se da extensão de métricas de complexidade cognitiva existentes e da introdução de diretrizes para calcular pontos de complexidade intrínseca, o método busca mitigar a sobrecarga cognitiva dos desenvolvedores. Estudos experimentais preliminares sugerem que o CDD pode reduzir significativamente o esforço necessário para futuras manutenções e correções de bugs, indicando uma melhoria nas práticas de desenvolvimento de *software* ao considerar fatores cognitivos.

Barbosa et al. (2022) teve como objetivo principal avaliar o impacto do CDD na legibilidade do código. O estudo foi conduzido em duas fases distintas. Na primeira fase, desenvolvedores profissionais avaliaram a legibilidade de pares de trechos de código, sendo que um deles foi codificado utilizando CDD. Foram obtidas 133 respostas nesta fase. Na segunda fase, os autores aplicaram um modelo avançado de legibilidade a 10 pares de refatorações feitas com CDD.

Os resultados mostraram que, na percepção dos desenvolvedores, sete das dez refatorações feitas com CDD foram consideradas mais legíveis, enquanto em dois casos os desenvolvedores ficaram indecisos e em um caso preferiram o código original. No entanto, a avaliação por modelos de legibilidade revelou que apenas uma refatoração CDD apresentou

melhor desempenho. Assim, os autores concluem que, embora o CDD possa ser uma abordagem promissora para o design de *software*, são necessários mais estudos para compreender plenamente seus impactos na legibilidade do código.

1.4 OBJETIVOS

1.4.1 Objetivo Geral

Analisar e compreender a metodologia *Cognitive-Driven Development* (CDD), explorando seus princípios, processos e impacto no desenvolvimento de *software*, visando contribuir para que os desenvolvedores trabalhem de maneira mais eficiente e produzam *softwares* de melhor qualidade.

1.4.2 Objetivos Específicos

1. Investigar e compreender o que é o *Cognitive-Driven Development* (CDD), seus conceitos fundamentais e como ele pode ser aplicado nos projetos.
2. Identificar e selecionar ferramentas que possam auxiliar na implementação do CDD.
3. Utilizar os conceitos do CDD para verificar a complexidade de códigos já existentes em projetos reais.
4. Avaliar o impacto da análise de complexidade cognitiva nos projetos de estudo.

1.5. METODOLOGIA

Etapa I: Revisão da Literatura

Foi realizado um breve mapeamento da literatura visando compreender os principais aspectos entorno do CDD assim possibilitando o entendimento das teorias e práticas que compõem a abordagem, assim tornasse possível compreender métricas, complexidades, e outros fatores que de forma indireta tornam-se relevantes, com os padrões de projetos para uma qualidade de *software*.

Etapa II: Análise das Métricas de Complexidade de Código

A análise das métricas de complexidade em um código permite um melhor entendimento do mesmo. Avaliando os valores que essas métricas proporcionam, é possível determinar quando o código é menos complexo e mais fácil de compreender. Dessa forma, é possível identificar os elementos que aumentam a complexidade do código e exigem um maior esforço de entendimento.

Etapa III: Seleção de Ferramentas de Apoio

Com base nos trabalhos analisados, foram selecionadas ferramentas que auxiliam na utilização do CDD. A principal ferramenta utilizada foi o *plugin* CDD, que pode ser instalado na IDE IntelliJ, permitindo ao usuário analisar e compreender as métricas de complexidade de seus códigos-fonte.

Além disso, há o CK (*Metrics Calculator*), outra ferramenta para calcular métricas. O GitHub também é utilizado para complementar a pesquisa, servindo como plataforma onde as análises de métricas são realizadas, de forma que cada trecho de código em um projeto recebe um valor de métrica. E a linguagem de programação que neste caso foi utilizada era o Java.

Etapa IV: Planejamento e Aplicação do CDD

O planejamento e aplicação do CDD envolve a utilização de técnicas de desenvolvimento orientadas por inteligência cognitiva para aprimorar a eficiência e a eficácia do processo de criação de *software*. O CDD integra a análise de dados e comportamentos dos usuários, promovendo uma compreensão mais profunda das necessidades e preferências. No planejamento, isso se traduz em estratégias mais alinhadas com o comportamento real dos usuários, permitindo ajustes contínuos e melhorias baseadas em *feedback* e *insights*. Durante a aplicação, o CDD facilita a criação de soluções adaptativas que respondem dinamicamente às mudanças nas demandas, resultando em produtos mais intuitivos e ajustados ao contexto do usuário.

1.6 ORGANIZAÇÃO DO DOCUMENTO

Este documento está organizado em cinco capítulos: No Capítulo 1, Introdução, são apresentados a problemática, motivação, trabalhos relacionados, os objetivos deste trabalho e

a metodologia usada. O Capítulo 2 contém a fundamentação teórica relacionada a este trabalho, abordando temas como neurociência cognitiva, conceitos fundamentais do CDD e padrões de projetos.

No Capítulo 3, é apresentada a metodologia da pesquisa, incluindo uma breve revisão da literatura, análises de métricas de complexidade de códigos, a seleção de ferramentas e a aplicação do CDD. O Capítulo 4, Resultados de análises, neste capítulo analisou-se as questões de pesquisa e a aplicação do CDD. Por fim, o Capítulo 5 apresenta as considerações finais e a conclusão deste trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 O MÁGICO NÚMERO SETE DE MILLER

De acordo com a lei de Miller (1956), a capacidade do ser humano de armazenar grandes informações é limitada, exceto em casos excepcionais, como indivíduos com Q.I. elevado, ou seja, gênios. Fazendo analogias às teorias da informação, ele compara as informações assimiladas por um ser humano, com os dados que são armazenados por uma máquina, como um computador. Com base nisso, ele sugere que, para aprender de forma mais eficiente, devemos fragmentar as informações que desejamos armazenar, ou seja, organizá-las em unidades menores para facilitar a memorização.

> "[...] Miller retrata sobre a coincidência entre os limites do julgamento absoluto unidimensional e os limites da memória de curto prazo. Em tarefas nas quais o indivíduo necessita a ter aprender informações em grandes quantidades, mas sua memória não consegue assimilar tanta informação." (LIMA, 2003).

Ao compreender alguns limites em nossa capacidade de processar informações, Miller descreve que um ser humano médio pode manter na memória de curto prazo é de sete, mais ou menos dois objetos referentes ao seu aprendizado, este artigo foi uma publicação de 1956 na Psychological Review.

A segunda limitação cognitiva que Miller discute é a extensão da memória. A extensão da memória se refere à lista mais longa de itens (por exemplo, dígitos, letras, palavras) que uma pessoa pode repetir na ordem correta em 50% das tentativas imediatamente após a apresentação. Miller observou que a extensão da memória de adultos jovens é de aproximadamente sete itens (MILLER, 1956).

2.2 NEUROCIÊNCIA COGNITIVA

O conceito de aprendizagem é essencial para entender o processo pelo qual um indivíduo adquire conhecimento, habilidades e características necessárias para conviver e sobreviver. Esse processo envolve diversos fatores, como emoções, neurologia, ambiente e relacionamentos. Ao inicia a definição aprendizagem deve se compreender que há diversos autores e que cada um tem uma forma de descrever como tal formas diversas para expressar

uma única ideia que é, aprendizagem o processo pelo qual um indivíduo percorrer para obter habilidades, conhecimentos, valores, competências, comportamentos e tudo isso é algo cumulativo ao decorrer do tempo (SCHUNK, 2012).

Memória de longo prazo desempenha um papel crucial nesse processo de aprendizagem, pois é ela que retém de forma definitiva a informação, permitindo sua recuperação ou a evocação de tais memórias. Nela estão contidos todos os nossos dados autobiográficos e todo nosso conhecimento. O processo de formação da memória de longo prazo (ou de longa duração) envolve diversos mecanismos fisiológicos que visam a desenvolver condições favoráveis ao armazenamento de informações através de alterações anatômicas e fisiológicas (SILVÉRIO, 2006).

A neurociência cognitiva, por sua vez, é um campo de estudo que trata de um conjunto de aspectos a fim de explicar o funcionamento da cognição humana e como se dá o processo de aprendizagem. Ou seja, a neurociência cognitiva, além de tentar desvendar a cognição humana utilizando de abordagens da psicologia, também incorpora outras abordagens como neuropsicológica, cognitiva, comportamental, computacional e integrativa para entender a cognição e aprendizagem (SCHUNK, 2012).

2.3 COGNITIVE-DRIVEN DEVELOPMENT

A Teoria da Carga Cognitiva, proposta em 1988 pelo psicólogo australiano John Sweller, na Universidade de New South West, a teoria amplia a compreensão das limitações da mente humana ao sugerir que o *design* instrucional deve levar em conta as limitações da memória de trabalho para otimizar o aprendizado. Ela destaca a importância de minimizar a carga cognitiva extrínseca (o esforço mental decorrente da forma como a informação é apresentada) e de maximizar as cargas cognitivas intrínseca (complexidade inerente ao conteúdo) e germinativa (processos mentais que promovem a aprendizagem) para facilitar um processamento eficaz das informações (SWELLER, 1988).

- **Carga Cognitiva Extrínseca**: Esta se refere à quantidade de esforço mental imposto pela maneira como a informação é apresentada ao um indivíduo. Inclui elementos não

relacionados ao conteúdo que podem dificultar o aprendizado, como um *design* confuso ou informações irrelevantes (SWELLER, 1988).

- **Carga Cognitiva Intrínseca**: Esta se refere à complexidade do que um indivíduo está tentando aprender. Está relacionada à dificuldade intrínseca do conteúdo e a quantidade de novos conceitos e informações que precisam ser processados por um indivíduo (SWELLER, 1988).

- **Carga Cognitiva Germinativa**: Esta se relaciona a processos mentais que ajudam a aprender e compreender o conteúdo e/ou informações com mais eficácia. Envolve a construção e organização do conhecimento, bem como a integração de novas informações como o conhecimento prévio (SWELLER, 1988).

Esta teoria retrata a capacidade que um indivíduo possui para reter informações em memória de curto prazo, impondo restrições para o aprendizado, neste caso com base nos diversos aspectos presente na neurociência cognitiva é possível utilizar de técnicas para diminuir a complexidade e facilitar a aprendizagem.

Para um desenvolvedor, compreender os códigos de outros pode ser demorado, e o custo de tempo extra necessário para assimilar novas informações pode variar de acordo com o conhecimento prévio de cada indivíduo. Nas empresas, esse custo de tempo extra pode ser improdutivo, destacando a importância de práticas eficientes e a adoção de soluções como o CDD (SANTIAGO, 2021).

Baseando-se no CDD, é possível minimizar a complexidade no desenvolvimento de *software*, utilizando métricas propostas por essa abordagem, como atribuição de valores para métodos como exceções, laços, condições que já são determinadas pelo CDD, para realizar a verificação de complexidade intrínseca do código. Ao analisar todos os aspectos presentes no desenvolvimento de *software*, como condições, anotações, classes e métodos, é possível formular uma abordagem concreta e bem fundamentada (SANTIAGO et al., 2020).

Uma das propostas do CDD é fornecer informações sobre a complexidade que, por sua vez, refletem a experiência do desenvolvedor. Ao analisar dados muitas vezes negligenciados, como a complexidade de cada *commit*, é possível obter uma variável que permite analisar cada indivíduo de maneira única. É importante ressaltar que essa análise não visa diretamente

os aspectos psicológicos do desenvolvedor, mas sim oferecer *insights* sobre como os desenvolvedores produzem seus códigos.

2.4 PADRÕES DE PROJETO

Ao utilizar padrões de projeto para obter códigos mais limpos e aplicar o CDD para minimizar a complexidade, os desenvolvedores podem reduzir a carga cognitiva dos softwares que estão produzindo, o que é essencial para facilitar a compreensão do código por outros membros da equipe. Na comunidade de desenvolvedores, embora a produtividade seja frequentemente a prioridade, o uso de padrões de projeto, que padronizam a abordagem arquitetural dos projetos, é bastante comum. Esses padrões possuem singularidades que permitem que outros desenvolvedores compreendam o código, desde que estejam familiarizados com eles (GAMMA, 1994). Segundo Christopher Alexander (1994), cada padrão descreve um problema no ambiente e sua solução de forma que possa ser reutilizada inúmeras vezes, sem que seja aplicada da mesma maneira (GAMMA, 1994).

De acordo com GAMMA, E. et al. (1994), em geral, um padrão tem quatro elementos essenciais:

1. O nome do padrão é uma referência que podemos usar para descrever um problema de projeto, suas soluções e consequências em uma ou duas palavras. Dar nome a um padrão aumenta imediatamente o nosso vocabulário de projeto. Isso nos permite projetar em um nível mais alto de abstração. Ter um vocabulário para padrões permite-nos conversar sobre eles com nossos colegas, em nossa documentação e até com nós mesmos. O nome torna mais fácil pensar sobre projetos e a comunicá-los, bem como os custos e benefícios envolvidos, a outras pessoas. Encontrar bons nomes foi uma das partes mais difíceis do desenvolvimento do nosso catálogo.

2. O problema descreve em que situação aplicar o padrão. Ele explica o problema e seu contexto. Pode descrever problemas de projeto específicos, tais como representar algoritmos como objetos. Pode descrever estruturas de classe ou objeto sintomáticas de um projeto inflexível. Algumas vezes, o problema incluirá uma lista de condições que devem ser

satisfeitas para que faça sentido aplicar o padrão.

3. A solução descreve os elementos que compõem o padrão de projeto, seus relacionamentos, suas responsabilidades e colaborações. A solução não descreve um projeto concreto ou uma implementação em particular porque um padrão é como um gabarito que pode ser aplicado em muitas situações diferentes. Em vez disso, o padrão fornece uma descrição abstrata de um problema de projeto e de como um arranjo geral de elementos (classes e objetos, no nosso caso) o resolve.

4. As consequências são os resultados e análises das vantagens e desvantagens (*trade-offs*) da aplicação do padrão. Embora as consequências sejam raramente mencionadas quando descrevemos decisões de projeto, elas são críticas para a avaliação de alternativas de projetos e para a compreensão dos custos e benefícios da aplicação do padrão. As consequências para o *software* frequentemente envolvem balanceamento entre espaço e tempo. Elas também podem abordar aspectos sobre linguagens e implementação. Uma vez que a reutilização é frequentemente um fator no projeto orientado a objetos, as consequências de um padrão incluem o seu impacto sobre a flexibilidade, a extensibilidade ou a portabilidade de um sistema. Relacionar essas consequências explicitamente ajuda a compreendê-las e avaliá-las.

Os padrões de projeto são soluções reutilizáveis para problemas recorrentes no desenvolvimento de *software* e são divididos em três categorias:

1. Criacionais: Focam na criação de objetos, promovendo flexibilidade e independência na instanciação de classes. Exemplos: *Abstract Factory*, *Builder*, *Factory Method*, *Prototype*, *Singleton*.
2. Estruturais: Tratam da composição de classes e objetos para formar estruturas maiores. Exemplos: *Adapter*, *Bridge*, *Composite*, *Decorator*, *Facade*, *Flyweight*, *Proxy*.
3. Comportamentais: Lidam com as interações e responsabilidades entre os objetos. Exemplos: *Chain of Responsibility*, *Command*, *Iterator*, *Observer*, *State*, *Strategy*, *Template Method*, *Visitor*.

No quadro 1, é notada a distribuição das categorias e seus padrões de projetos.

Quadro 1 - Categorias e seus padrões

Categoria	**Padrões**
Criacionais	Abstract Factory
	Builder
	Factory Method
	Prototype
	Singleton
Estruturais	Adapter
	Bridge
	Composite
	Decorator
	Facade
	Flyweight
	Proxy
Comportamentais	Chain of Responsibility
	Command
	Interpreter
	Iterator
	Mediator
	Memento
	Observer
	State
	Strategy
	Template Method
	Visitor

Fonte: elaborada pelo autor

3 METODOLOGIA

3.1 REVISÃO DA LITERATURA

Neste trabalho, foi realizado um breve mapeamento sistemático da literatura a fim de selecionar trabalhos científicos que abordassem sobre o método CDD. Inicialmente, a pesquisa foi realizada por meio do buscador Google e no Google Acadêmico. Entretanto, foram encontrados poucos trabalhos científicos que abordassem sobre o tema desta pesquisa. Desta forma, foram analisadas as referências bibliográficas dos trabalhos inicialmente selecionados a fim de ampliar o escopo da pesquisa. Ao final deste processo, foram selecionados 9 trabalhos.

A análise dos trabalhos teve como objetivo responder as seguintes questões investigativas:

(Q1) Quais os principais objetivos dos trabalhos relacionados à utilização do método *Cognitive-Driven Development* (CDD)?

(Q2) Como utilizar o CDD de forma eficaz no desenvolvimento de *software*?

(Q3) Os trabalhos relacionados utilizam alguma ferramenta digital para facilitar a implementação do CDD?

(Q4) Quais os principais resultados alcançados a partir do uso do método de CDD?

(Q5) Quais as limitações levantadas pelos estudos analisados?

(Q6) Qual a relação do CDD com padrões de projeto?

(Q7) Quais as principais lacunas de pesquisa identificadas nos trabalhos analisados?

3.2 ANÁLISE DAS MÉTRICAS DE COMPLEXIDADE DE CÓDIGO

Neste subcapítulo é apresentada uma lista dos principais tipos de métricas de *software* que podem ser utilizadas no estudo de aplicação do CDD, com isso é possível compreender a

variação e quantidades de métricas, no mais vemos algumas categorias como, *Basic metrics*, *CK metrics*, *Complexity metrics* e *Coupling metrics*.

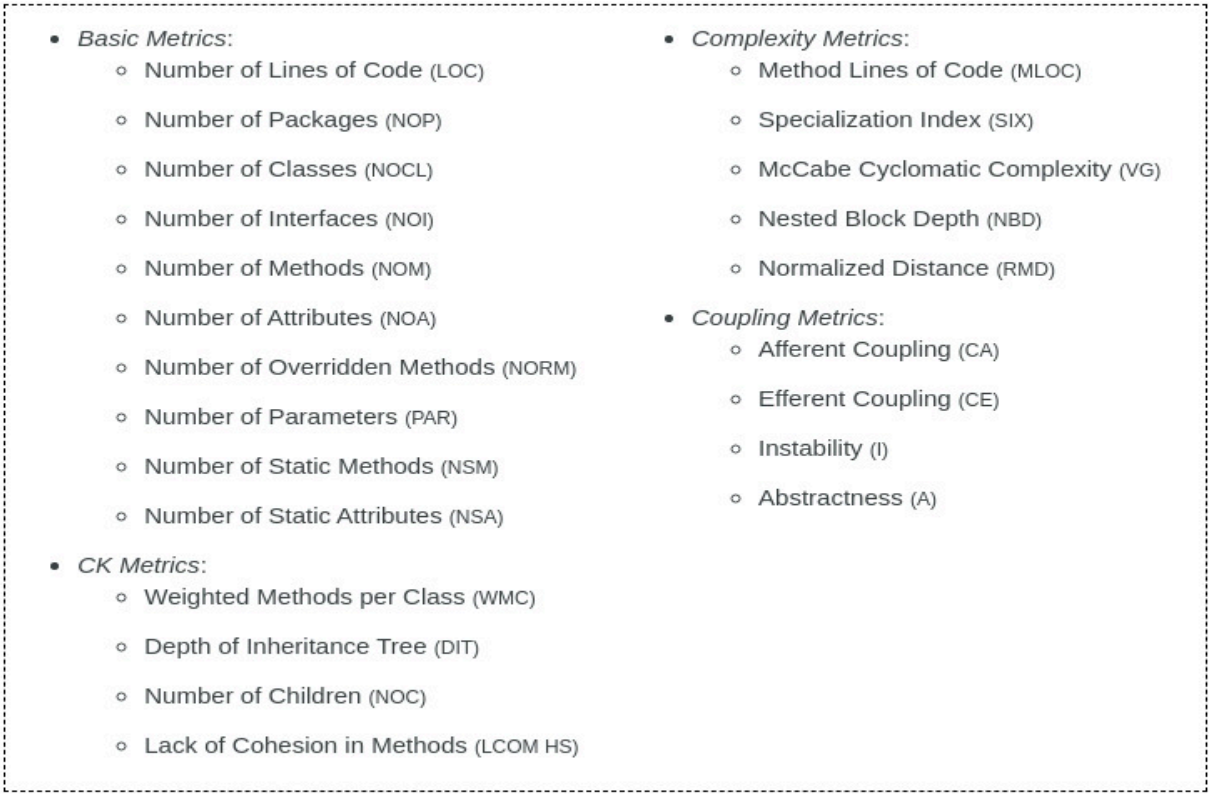
Basic Metrics: Medem atributos fundamentais do *software*, como linhas de código e número de métodos.

CK Metrics: Avaliam a qualidade da orientação a objetos, incluindo acoplamento e coesão.

Complexity Metrics: Medem a complexidade do código para identificar áreas de risco e manutenção.

Coupling Metrics: Avaliam a dependência entre módulos para melhorar modularidade e reutilização.

Figura 1 - Métricas Fornecidas pelo Qualitas.class

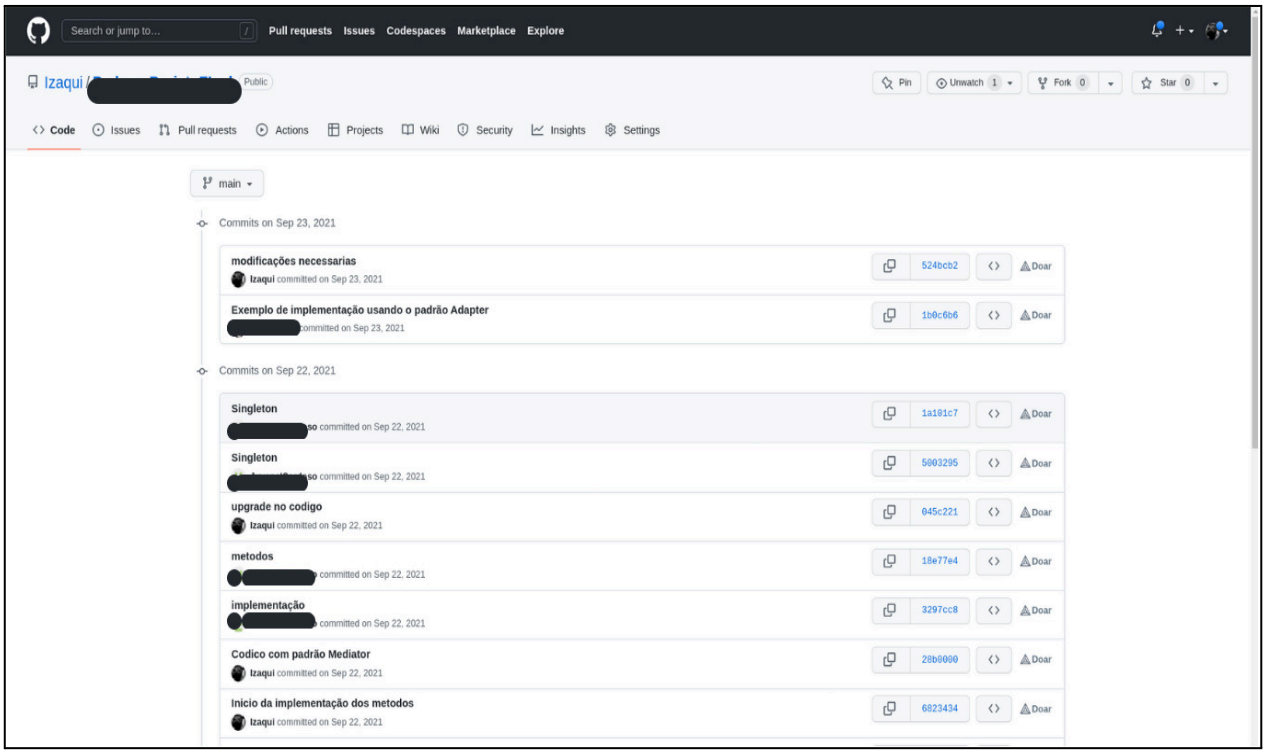


Fonte: http://java.labsoft.dcc.ufmg.br/qualitas.class/index.html

Para definir o grau de complexidade em um código utilizando o CDD, definimos valores no qual a somatória de tais nos dá como retorno nossa carga cognitiva.

Um ponto relevante será a análise da carga cognitiva gerada pelos *commits* de um desenvolvedor, com isso podemos obter valores que possam determinar um certo grau de experiência do indivíduo, entretanto, esse cálculo pode se tornar complexo. Devido a isso, os requisitos que podem ser levados em consideração para uma relevância na criação de uma artifício para analisar a experiência do indivíduo, que pode ser definido como: quantidades de *commits*, frequência de cada *commit* (isso inclui a relevância do que está sendo *commitado*, ou seja o valor que implica ao código) ou até mesmo o tempo de experiência do desenvolvedor. Muitos destes fatores podem ser irrelevantes de forma individual, no entanto o conjunto de tais fatores em uma análise um pouco mais detalhada nos remete aos graus de complexidades intrínseca, para que assim possamos determinar um nível de entendimento do desenvolvedor que está criando o código.

Figura 2 - Frequência dos *commits*



Fonte: elaborada pelo autor

Logo mais, na figura 4, é apresentado um exemplo de código com padrão mediator, que foi fornecido para a análise envolvendo o CDD, no qual já encontramos complexidades por falta de *COMMITS* e *ANNOTATION*.

Algoritmo 1- Exemplo de código com padrão Mediator

```
package CodigoIzaquiel;

import CodigoIzaquiel.components.*;

import CodigoIzaquiel.mediator.Mediator;

import javax.swing.*;

public class Main {

   public static void main(String[] args) {

        Mediator mediator = new Edit();



        mediator.registerComponent(new BoxTexto());

        mediator.registerComponent(new Button());



        mediator.createGUI();

   }

}
```

Fonte: elaborada pelo autor

3.3 SELEÇÃO DE FERRAMENTAS DE APOIO

Para este trabalho foi utilizado as seguintes ferramentas: IDE IntelliJ[2], CK calculates metrics[3], *Plugin* CDD[4] para detectar a complexidade intrínseca dos códigos, linguagem java[5], github[6].

[2] https://www.jetbrains.com/pt-br/idea/
[3] https://github.com/mauricioaniche/ck
[4] https://github.com/gustavopintozup/plugin-cdd-java
[5] https://www.java.com/
[6] https://github.com/

IDE IntelliJ: *Integrated Development Environment* (IDE) popular para Java, com recursos avançados como refatoração e *debugging*, que aumentam a produtividade.

CK (Calculates Metrics): Ferramenta que avalia métricas de código como complexidade ciclomática, ajudando a identificar áreas problemáticas.

***Plugin* CDD**: Um *plugin* desenvolvido para detectar a complexidade intrínseca do código, facilitando sua simplificação e melhorando a manutenibilidade.

Linguagem Java: Linguagem de programação robusta e portável, foi a linguagem base para realizar as análises iniciais utilizando o *plugin* do CDD .

GitHub: Plataforma de hospedagem e versionamento de código, permitindo colaboração eficiente entre desenvolvedores, aqui o github é ferramenta utilizada para realizar a análise de um projeto como todo, proporcionando uma compreensão completa do projeto mesmo analisando trechos assim proporcionando métricas completas.

3.4 PLANEJAMENTO E APLICAÇÃO DO CDD

Por meio de uma análise dos códigos desenvolvidos por programadores distintos, os quais são comparados utilizando o *plugin* do CDD, é possível obter a carga cognitiva do código, ou seja, sua complexidade intrínseca. Essa análise, por sua vez, permite observar dados que levam ao entendimento de como o fator experiência pode afetar diretamente o desenvolvimento de um *software*. A priori, utiliza-se um grupo para a análise com as seguintes avaliações de requisitos propostos: validações, acoplamento, herança, commits, laços e desvios condicionais.

Na utilização de ferramentas como o CK, que calcula métricas de código em nível de classe e nível de método em projetos Java por meio de análise estática, o objetivo principal é o uso de outra ferramenta desenvolvida pelos pesquisadores Pinto et al. (2021). Nesse caso, trata-se de um *plugin* criado para a IDE IntelliJ, destinado à análise da carga cognitiva de projetos criados em Java. Ao utilizar essa ferramenta, o desenvolvedor terá à sua disposição os dados sobre a complexidade intrínseca do código.

A abordagem utilizando o CDD é realizada da seguinte forma: a análise da complexidade é feita por meio de pontuações, como pode ser observado no exemplo a seguir.

Algoritmo 2: Exemplo de json para pontuar complexidade:

```
{
 "limite": 10,
 "regras": [
  {
   "name": "IF_STATEMENT",
   "cost": 1
  },
  {
   "name": "TRY_CATCH_STATEMENT",
   "cost": 1
  },
  {
   "name": "SWITCH_STATEMENT",
   "cost": 1
  },
  {
   "name": "CONDITION",
   "cost": 1
  },
  {
   "name": "FOR_STATEMENT",
   "cost": 1
  },
  {
   "name": "FOREACH_STATEMENT",
   "cost": 1
  },
  {
   "name": "WHILE_STATEMENT",
   "cost": 1
  },
  {
   "name": "CONTEXT_COUPLING",
   "cost": 1,
   "parameters": "br.com.zup.lms"
  }
 ]
}
```

Neste JSON, é possível observar como a complexidade é pontuada, com um limite de dez pontos para uma complexidade aceitável. De acordo com as regras impostas pelo CDD, tem-se as seguintes categorias: IF_STATEMENT, TRY_CATCH_STATEMENT,

SWITCH_STATEMENT, CONDITION, FOR_STATEMENT, FOREACH_STATEMENT, WHILE_STATEMENT e CONTEXT_COUPLING.

No entanto, definir uma variável para representar a experiência de um indivíduo torna-se complexo, especialmente para uma ferramenta realizar essa análise. Contudo, ao considerar aspectos cognitivos de aprendizagem, é possível compreender uma formulação que gera uma pontuação, permitindo a criação de dados para analisar essa experiência de forma quantitativa.

A análise desses aspectos é realizada da seguinte maneira: cada *commit* relevante possui seus respectivos trechos de código, e cada trecho tem uma carga cognitiva que pode ser avaliada. Para criar uma referência de experiência, o seguinte cálculo é realizado: a quantidade de *commits* com relação cognitiva, ou seja, commits que possuem uma carga cognitiva pontuada, somada à pontuação de todos os commits, sendo esse valor dividido pela quantidade total de commits, para que se possa atribuir uma métrica geral aos dados.

Fórmula: A Carga Cognitiva por *commit* será chamada de Cc, variando de Cc1 até Ccn. O total, ICc (Individual CDD Commit), será a variável de comparação de complexidade.

Variáveis:

- CddCommit = (É a somatória dos valor de CDD do *commits*)
- Cc = (É a caga cognitiva do código *commitado*)
- ICc = (É a média geral do projeto)

$$(CddCommit) = \sum_{i=1}^{n} (Cc^{1} + Cc^{2} \dots Cc^{n})$$

$$ICc = \frac{(CddCommit)}{total}$$

O cálculo do *ICC* é uma média geral do projeto a ser identificado com complexidade ou não, não importando se trecho de seu código venha a possuir complexidades elevadas. Iniciasse realizando uma somatória das cargas cognitivas de cada *commit* que possui valores de carga de complexidade na qual a variável *Cc* recebe a carga de complexidade do *commit* em questão, e essa somatória é armazenada em uma variável denominada *CddCommit*.

Por fim determinar a média do projeto é utilizada uma variável de armazenamento denominada *ICc*, esta variável recebe a somatória e o total de *commits* que foram analisados, dividindo *CddCommit* por total.

Ao utilizar o ICC no controle de versão para manter uma média aceitável de complexidade, permitindo aos desenvolvedores refatorar códigos no seu desenvolvimento se necessário, além de manter a qualidade, legibilidade e sua manutenção aceitáveis.

4. RESULTADOS E ANÁLISES

4.1 ANÁLISE DAS QUESTÕES DE PESQUISA

(Q1) Quais os principais objetivos dos trabalhos relacionados à utilização do método Cognitive-Driven Development (CDD)?

Entre os trabalhos analisados, observou-se que o primeiro trabalho a tratar especificamente sobre o CDD foi o de Alberto de Souza e Victor Pinto (2020), intitulado "*Toward a Definition of Cognitive-Driven Development*". Esta pesquisa se baseia na análise de métricas de complexidade cognitiva e também na Teoria da Carga cognitiva. A abordagem proposta minimiza a sobrecarga cognitiva ao limitar os pontos de complexidade intrínseca do código-fonte no ambiente de desenvolvimento. Neste trabalho foram estabelecidas algumas métricas de complexidade, além de sugerir como os limites podem ser ajustados com base em critérios específicos de qualidade (SOUZA & PINTO, 2020).

Outras pesquisas tiveram como foco a realização de estudos experimentais, como é o caso de Pereira et al. (2021), no qual os estudos experimentais que realizados demonstram que o CDD tornou-se um método bastante promissor para orientar no desenvolvimento de *software*, com foco na compreensão e obtenção de código de alta qualidade no que diz respeito às métricas de qualidade (PEREIRA et al., 2021).

O trabalho de Pinto et al. (2021) demonstra, por meio de um estudo experimental realizado em contexto industrial voltado a avaliar refatorações de códigos usando o CDD, que o método tem como princípio, manter um desenvolvimento de *software* baseado na práticas de restrição congênita, para assim minimizar complexidade, como é apontado pelo CDD (PINTO

et al., 2021)

Já o de Barbosa et al. (2022) apresenta um estudo experimental que investiga os efeitos do CDD nos estágios iniciais do desenvolvimento de *software*, comparando-o com as práticas convencionais. O objetivo é avaliar se o uso do CDD melhora a legibilidade do código em relação aos métodos mais tradicionais (BARBOSA et al., 2022).

Além disso, dois artigos tratam do desenvolvimento de aplicações que apoiem o uso do CDD. No artigo de Pereira et al. (2021), é apresentada uma ferramenta denominada "Cognitive Load Analyzer", um *plugin* para IntelliJ IDEA e linguagem Java que visa suportar o CDD. Esta ferramenta é útil na adoção do CDD e tem como objetivo facilitar a introdução do CDD no desenvolvimento de *software*, introduzindo aspectos visuais com identificação de complexidade dentro dos códigos, visando superar o fatiamento do código e ajudar os desenvolvedores a reduzir a complexidade do código.

No artigo de Gustavo Pinto e Alberto Souza (2022) foram relatadas as experiências de uma equipe de desenvolvimento de *software* utilizando CDD, na construção de uma ferramenta de gestão de aprendizagem que foi desenvolvida do zero, este experimento foi aplicado dentro da Zup Innovation, uma empresa brasileira de tecnologia. Ao realizar a curadoria dos rastros de *commit* deixados nos repositórios, combinada com a percepção dos desenvolvedores, os autores organizaram um conjunto de descobertas e lições que podem ser úteis para quem deseja adotar o CDD.

Os dois últimos trabalhos analisados são focados na legibilidade e evolução do código. O artigo de Barbosa et al. (2022) tem como objetivo avaliar em que medida o CDD contribui para a melhoria da legibilidade do código. No artigo de Belém et al. (2023), os resultados iniciais de uma investigação dos *drivers* de carga cognitiva de atividades de evolução de *software*. Após analisar o conteúdo de entrevistas semiestruturadas conduzidas com membros de uma equipe de manutenção de *software*, foi encontrado um conjunto de problemas e dificuldades que eles vivenciam ao evoluir sistemas de *software*. Com base nesse conjunto, foi feito um mapeamento de uma versão preliminar dos *drivers* de carga cognitiva de atividades de evolução de *software*.

A partir dos trabalhos analisados, observou-se que o principal objetivo das pesquisas é apresentar os princípios do CDD e demonstrar suas aplicabilidade dentro do ambiente de

desenvolvimento de *software*, envolvendo as várias áreas de conhecimento que envolvem o processo de criação de um *software.* A ideia do CDD é trabalhar a fragmentação de conhecimento a fim de compreender os fragmentos de forma individual, a simplificação, boas práticas de programação, entendimento do código, análise de métricas. Todos esses objetivos são todos voltados para a criação de um ambiente de desenvolvimento de *software* que favoreça a eficiência, a clareza e a sustentabilidade ao longo do ciclo de vida do *software*.

(Q2) Como utilizar o CDD de forma eficaz no desenvolvimento de software?

Ao adotar as práticas do CDD, é possível criar um ambiente de desenvolvimento mais sustentável e eficaz, onde o código se torna mais fácil de entender, modificar e manter. O objetivo é sempre encontrar um equilíbrio entre a complexidade necessária para resolver o problema e a simplicidade desejada, a fim de reduzir a carga cognitiva dos desenvolvedores.

O CDD é uma abordagem voltada para a gestão da carga cognitiva dos desenvolvedores durante a construção de sistemas. Sua aplicação envolve aspectos como a divisão e simplificação de tarefas, ou seja, a modularização do código, mantendo funções curtas e específicas. Além disso, busca-se reduzir o uso de variáveis globais e o estado compartilhado, promover a imutabilidade, adotar boas práticas de nomenclatura, *design* orientado a eventos, fluxos de controle claros, e uma documentação inteligente (SOUZA & PINTO, 2020).

A abordagem também incentiva a refatoração contínua, revisões colaborativas, e o uso de ferramentas para visualização e métricas de complexidade e ferramenta como Cognitive Load Analyzer uma ferramenta projetada para avaliar e monitorar a carga cognitiva de usuários durante a interação com sistemas, interfaces ou tarefas específicas (PEREIRA et al., 2021).

O termo *Intrinsic Complex Point* (ICP) é utilizado para denominar a pontuação de complexidade intrínseca que se dá a um recurso de uma categoria dentro de um código. Ao analisar um código usando o ICP, torna-se possível determinar se o mesmo está com sua complexidade ultrapassando o limite ideia e assim minimizar a complexidade (SOUZA & PINTO, 2020).

Exemplo: Cada elemento do código-fonte é chamado de ICP. A figura abaixo apresenta algumas categorias de ICP de acordo com o trabalho de Souza e Pinto (2020).

Figura 4 - Medições para Pontos de Complexidade Intrínseca

Medições para Pontos de Complexidade Intrínseca		
Categoria	**BCS e Recursos**	**ICP**
Ramificação	if-else	2
	case	1
Tratamento de Exceções	Try-catch-finally	3
Acoplamento	Acoplamento contextual*	1
	Funções como argumento	1
Requisitos Transversais	Código relacionado à lógica de infraestrutura, frameworks e biblioteca	0

Fonte: (SOUZA & PINTO, 2020)

Determina-se um valor de limite, no qual o código seria considerado aceitável; caso o ICP seja acima do valor ele tornasse um código com sobrecarga de complexidade, ultrapassando o limite. Ao ser definida uma complexidade elevada, a melhor forma de manter o ICP ideal é a alteração do código, assim torna-se possível alterar o seu indicado e reduzir a complexidade (SOUZA & PINTO, 2020).

Dentro de um desenvolvimento de um código, sua complexidade só poderá ser alterada após refatoração do código, pois após análise e do ICP o valor determinado como limite de complexidade não torna-se mutável.

Todo o processo para pontuar e determinar cargas de complexidade podem ser feitos de forma manual, e de forma consideravelmente simples, pois a partir da determinação do que do valores de ICP dentro de um código, sua somatória é o valor de complexidade daquele código.

(Q3) Os trabalhos relacionados utilizam alguma ferramenta digital para facilitar a implementação do CDD?

De acordo com os trabalhos analisados, observou-se que a única ferramenta criada com o propósito específico de trabalhar o CDD é a Cognitive Load Analyzer, que é um *plugin* para IntelliJ IDEA e linguagem Java para dar suporte à adoção do CDD. O objetivo desta ferramenta é orientar os desenvolvedores durante a programação e reduzir a sobrecarga cognitiva em esforços futuros com a evolução do *software* (PEREIRA et al., 2021).

Além do Cognitive Load Analyzer, Pereira et al. (2021) cita outra ferramenta utilizada que é o IntelliJ IDEA, que está entre uma das IDEs Java mais adotados e eficientes já lançadas até hoje. Esse foi o principal motivo pelo qual os desenvolvedores da ferramenta escolheram a plataforma. Outro motivo foi o suporte avançado para análise estática de códigos-fonte Java por meio de AST (*Abstract Syntax Tree*), onde o SDK do IntelliJ já suporta vários identificadores. Isso é útil para desenvolver novos *plugins* e melhorar a experiência do programador. Essas ferramentas ajudam o desenvolvedor a reduzir a complexidade em tempo real, com a comparação do código a padrões conhecidos, como SOLID, DRY e KISS, além de oferecer *feedback* instantâneo sobre como melhorá-lo.

Essas funcionalidades tornam a implementação do CDD mais prática, reduzindo a carga cognitiva diretamente na fase de desenvolvimento e permitindo que os desenvolvedores escrevam código mais limpo e fácil de manter. Diversas métricas de *software* são empregadas por meio de análise estática, como CBO (*Coupling Between Objects*), WMC (*Weighted Methods per Class*), RFC (*Response For a Class*), LCOM (*Lack of Cohesion of Methods*) e LOC (*Lines of Code*). Os resultados indicam que o CDD pode orientar o processo de reestruturação, promovendo uma separação de interesses coerente e equilibrada.

(Q4) Quais os principais resultados alcançados a partir do uso do método de CDD?

Observou-se alguns resultados conflitantes entre eles notou-se que que 70% dos desenvolvedores perceberam que refatorações baseadas em CDD possuía maior legibilidade do que suas versões originais, já em 20% dos desenvolvedores notaram que a refatorações orientadas por CDD, os deixaram indecisos, enquanto em apenas 10% dos desenvolvedores preferiram o código original. Em contrapartida foi possível notar que apenas ao utilizar

refatorações baseada em CDD apresenta melhor legibilidade e desempenho. Esses resultados fornecem evidências iniciais de que o CDD pode ser uma abordagem promissora para o *design* de *software* (BARBOSA et al., 2022).

(Q5) Quais as limitações levantadas pelos estudos analisados?

Entre os trabalhos, observam-se limitações, como a produção de códigos com alta complexidade, o que resulta em sobrecarga cognitiva para os desenvolvedores. A ideia principal por trás do CDD é manter as unidades de implementação dentro de uma restrição determinada, mesmo com a expansão contínua da escala do *software*, com tudo nota-se que existem determinados setores dos em um projeto que não devem ser completamente simplificados, ou seja não terão cargas cognitivas minimizadas. Ao decorrer dos estudos feitos nos trabalhos relacionados encontrasse tais limitações (PINTO & SOUZA, 2022).

(Q6) Qual a relação do CDD com padrões de projeto?

Neste estudo, observou-se que o CDD (Cognitive Driven Development) possui relações com as boas práticas de programação, incluindo as ferramentas usadas para sua análise. Assim, torna-se fundamental demonstrar a correlação entre os padrões de projeto e o CDD, uma vez que os padrões são amplamente considerados uma boa prática entre os desenvolvedores. O simples fato de sua utilização pode contribuir para a minimização da complexidade em um projeto, o que está diretamente associado ao CDD.

O CDD e os padrões de projeto se complementam, pois ambos têm como objetivo melhorar a qualidade e a compreensão do código. O CDD foca na redução da carga cognitiva, tornando o código mais simples e intuitivo. Já os padrões de projeto oferecem soluções reutilizáveis para problemas recorrentes, facilitando a organização e a legibilidade do código. No contexto do CDD, os padrões de projeto ajudam a abstrair a complexidade e a melhorar a clareza. No entanto, é importante usá-los com cautela para evitar o aumento desnecessário da complexidade.

No entanto, dentro dos trabalhos relacionados não foi possível encontrar referências na utilização dos padrões de projetos.

(Q7) Quais as principais lacunas de pesquisa identificadas nos trabalhos analisados?

Observou-se, entre os trabalhos, que existem singularidades em projetos complexos e robustos, nos quais a complexidade elevada é uma característica constante. Além disso, desenvolvedores iniciantes apresentam mais dificuldades na implementação do CDD nesses projetos, mesmo com o uso das ferramentas já citadas. Também se constatou que, entre os trabalhos, há divergências internas, como na utilização da linguagem de programação Java e suas variações, bem como nas ferramentas, que, por serem parte de um estudo recente, ainda são desconhecidas por grande parte da comunidade de desenvolvedores.

Embora os trabalhos ressaltem a criação de um plugin para o IntelliJ e seu uso, uma limitação identificada é que a ferramenta só pode ser utilizada em versões relativamente antigas do IntelliJ IDEA. Além disso, o projeto da ferramenta foi descontinuado, apresentando também limitações para outras linguagens de programação.

Por fim, observou-se que não foram encontradas referências nos trabalhos relacionados ao uso de padrões de projeto, evidenciando uma lacuna que pode ser investigada, assim como a análise da correlação entre o CDD e os padrões de projeto.

4.2 APLICAÇÃO DO CDD

Em uma análise em códigos com variação de tempo de experiência do desenvolvedor se pode notar uma evolução de na qualidade e complexidades reduzidas, ou seja devido o desenvolvido possuir o fato tempo de experiência a seu favor a compreensão mesmo que de forma involuntária pode vir a deixar o código menos complexo.

Figura 5 - Complexidade intrínseca de uma class exemplo 1

Fonte: elaborado pelo autor

Figura 6 - Complexidade Intrínseca de uma class exemplo 2

Fonte: elaborado pelo autor

Na Tabela 1 está classificado o grau de complexidade referente aos códigos descritos nas Figuras 5 e 6.

Tabela 1 - Complexidades

Projetos	**Complexidade por Class**
Delivery-BackEnd(class UsuarioController)	complexidade “5”
JMS-projeto(classe SimpleMessageClient)	complexidade “6”

Fonte: elaborado pelo autor

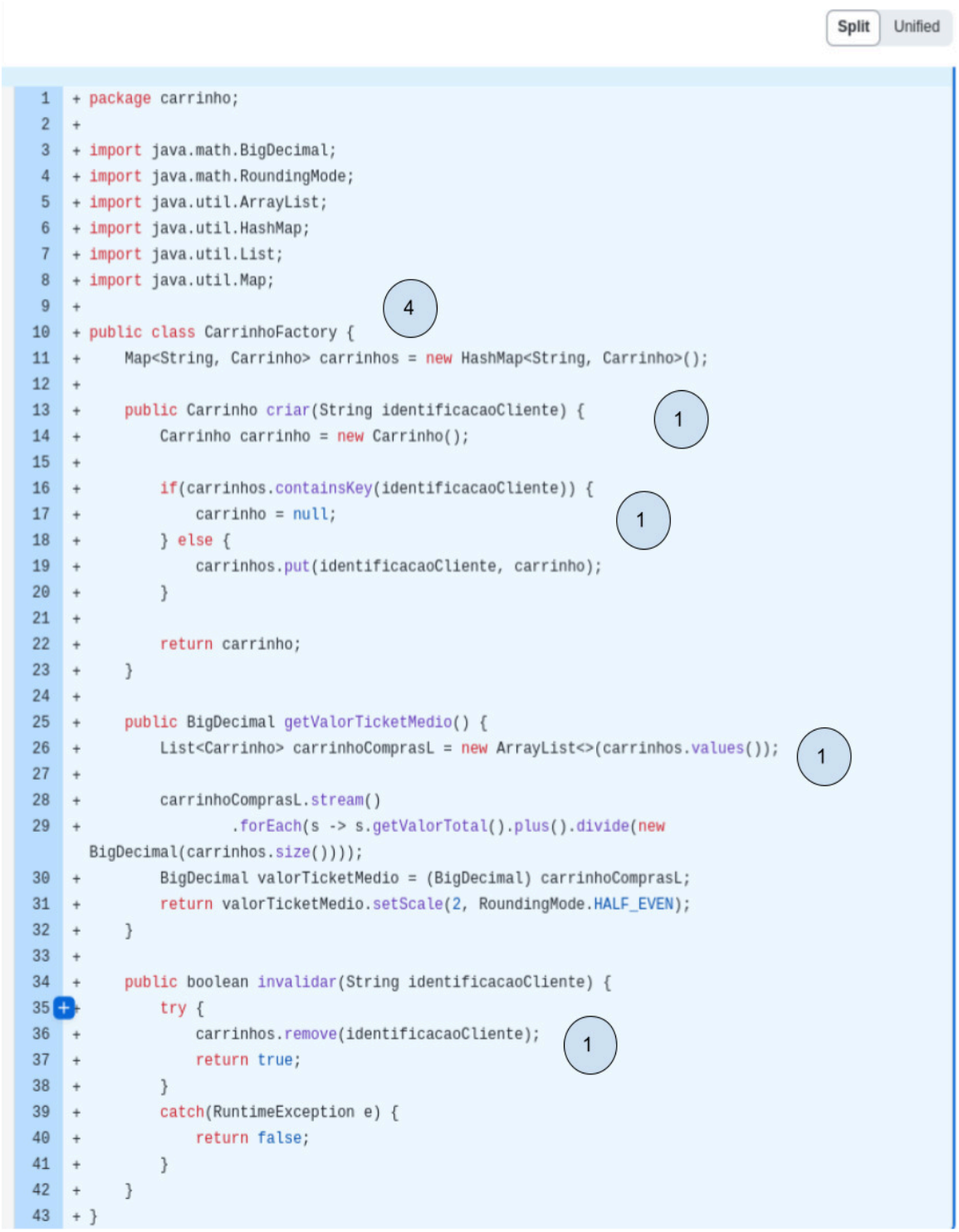
Ao analisar dados de complexidade utilizando a estratégia CDD com seus aspectos, um código se define com complexidade abaixo do limite, mas um dos fatores para ter uma complexidade reduzida provém de experiência e aprendizagem, entretanto ambos não constituem algo concreto, a experiência de um desenvolvedor se refere ao fato de compreender e assimilar conhecimento ao longo de sua carreira, com isso se pode compreender que quando mais trabalha em algo se pode ter uma melhor experiência e sua cognição se apura deixando de fato seus projetos menos complexos.

Figura 8 - Complexidade por commit


```
1  + package dataBase;
2  +
3  + import java.sql.DriverManager;
4  +
5  + public class Connection {
6  +     //Ususario e senha do banco
7  +     private static final String USERNAME = "root";
8  +     private static final String PASSWORD = "";
9  +
10 +
11 +     private static final String DATABASE_URL = "jdbc:mysql://localhost:3306/P";
12 +
13 +
14 +     public static Connection createConnectionToMySQL() throws Exception {
15 +
16 +         Class.forName("com.mysql.jdbc.Driver");
17 +         Connection connection = (Connection) DriverManager.getConnection(DATABASE_URL, USERNAME,
   PASSWORD);
18 +
19 +         return connection;
20 +     }
21 +
22 +
23 +     public static void main(String[] args) throws Exception {
24 +
25 +         Connection con = createConnectionToMySQL();
26 +
27 +         if (con != null) {
28 +             System.out.println("Conexão obetida com sucesso!");
29 +             con.close();
30 +         }
31 +     }
32 +
33 +     private void close() {
34 +     }
35 + }
```

Fonte : elaborado pelo autor

Figura 9 - Complexidade por *commit*




```
+ package carrinho;
+
+ import java.math.BigDecimal;
+ import java.math.RoundingMode;
+ import java.util.ArrayList;
+ import java.util.HashMap;
+ import java.util.List;
+ import java.util.Map;
+
+ public class CarrinhoFactory {
+     Map<String, Carrinho> carrinhos = new HashMap<String, Carrinho>();
+
+     public Carrinho criar(String identificacaoCliente) {
+         Carrinho carrinho = new Carrinho();
+
+         if(carrinhos.containsKey(identificacaoCliente)) {
+             carrinho = null;
+         } else {
+             carrinhos.put(identificacaoCliente, carrinho);
+         }
+
+         return carrinho;
+     }
+
+     public BigDecimal getValorTicketMedio() {
+         List<Carrinho> carrinhoComprasL = new ArrayList<>(carrinhos.values());
+
+         carrinhoComprasL.stream()
+                 .forEach(s -> s.getValorTotal().plus().divide(new
BigDecimal(carrinhos.size())));
+         BigDecimal valorTicketMedio = (BigDecimal) carrinhoComprasL;
+         return valorTicketMedio.setScale(2, RoundingMode.HALF_EVEN);
+     }
+
+     public boolean invalidar(String identificacaoCliente) {
+         try {
+             carrinhos.remove(identificacaoCliente);
+             return true;
+         }
+         catch(RuntimeException e) {
+             return false;
+         }
+     }
+ }
```

Fonte: elaborado pelo autor

Na análise de *commits* isolados, é possível obter resultados significativos para que assim toda a comparação na hipótese seja válida.

Tabela 2 - *commit* e suas pontuações

Commit	Pontuação
Class connection	3
Class carrinhoFatory	4

Fonte: elaborado pelo autor

Algoritmo 3 - Código fonte connect DataBase com complexidade elevada

```
package dataBase;
import dominio.Cine;
import dominio.Filme;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.util.ArrayList;
import java.util.List;

// complexidade = 10
public class DataBase {
	public void save(Cine cine) {

	String sql = "INSERT INTO pessoa(nome, cpf, dependente,id) VALUES (?, ?, ?, ?)";

		Connection conn = null;
		PreparedStatement pstm = null;

		try { // 1
			conn = Connection.createConnectionToMySQL();
			pstm = (PreparedStatement) conn.prepareStatement(sql);
			pstm.setString(1, cine.getFilme());
			pstm.setString(2, String.valueOf(cine.getAtor()));
			pstm.setString(3, cine.getEvento());
			pstm.execute();

			System.out.println("salvo com sucesso!");
		} catch (Exception e) {
				e.printStackTrace();
		} finally {
```

```
            try { // 1
                if (pstm != null) { // 1
                        pstm.close();
                }
                if (conn != null) { // 1
                        conn.close();
                }
            } catch (Exception e) {
                    e.printStackTrace();
            }
            }
    }

    public List<Cine> getContatos(){

            String sql = "SELECT * FROM contatos";

            List<Cine> cines = new ArrayList<Cine>();

            Connection conn = null;
            PreparedStatement pstm = null;
            ResultSet rset = null;
            try { // 1
                    conn = Connection.createConnectionToMySQL();

                    pstm = (PreparedStatement) conn.prepareStatement(sql);

                    rset = pstm.executeQuery();

                    while (rset.next()) { // 1
                            Cine cine = new Cine();
                            cine.setFilme(rset.getObject());
                            cine.setAtor(rset.getObject());
                            cine.setEvento(rset.getObject());
                            cines.add(cine);
                    }
            }catch (Exception e) {
                    e.printStackTrace();
            }finally {
```

```
            try { //  1
                    if(rset!=null) { // 1
                            rset.close();
            }

            if(pstm!=null) { // 1
                    pstm.close();
            }

            if(conn!=null) { // 1
                    conn.close();
            }
            }catch(Exception e) {
                    e.printStackTrace();
            }
            }
            return cines;
    }
}
```

Fonte: elaborado pelo autor

No exemplo do algoritmo que vemos acima, notamos que a complexidade está no limite máximo, entretanto é possível notar que ao avaliar e comparar as complexidades para no final do projeto, possibilita ser recodificada para minimizar o impacto de complexidade no geral.

A partir da análise dos dados pode-se concluir que tornasse necessário a atribuição mínima de valor de cargas de complexidade, que compõe grandes projetos com *commits* que possuem certa relevância, assim minimizando o impacto da carga cognitiva intrínseca assim propondo a utilização de todos os aspectos recorrente neste trabalho, como a utilização de fragmentação para aprendizagem como vimos no artigo de Miller com seu número mágico, assim possuindo.

5 CONSIDERAÇÕES FINAIS

Neste trabalho analisou-se o CDD que é uma área relativamente nova, na qual não possui tanta exploração, o conceito surgiu em 2020, no qual alguns trabalhos foram publicados realizando estudos experimentais e testando a complexidade intrínseca de código com os ICPs. Os trabalhos publicados demonstraram resultados promissores, como no artigo de Alberto Luiz Oliveira Tavares de Souza que foi publicado em 2020, além do estudo que proporcionou a criação da ferramenta Cognitive Load Analyzer que foi descrita na publicação do artigo de Jherson Haryson A. Pereira em 2021.

Neste amplo estado para compreender o CDD foi feita uma análise em trechos de códigos dentro repositórios, assim com uma análise de *commit* a *commit* é possível acompanhar padrões de complexidade intrínseca de nos trechos de código e possibilita realizar uma comparação por complexidade notando que ao adotar o CDD a carga de complexidade intrínseca é reduzida, além de uma melhora significativa na legibilidade.

Neste estudo foi feita a aplicação do método CDD, e é possível analisar tanto a carga cognitiva de todo um projeto e também de membros da equipe de forma individual. Assim existindo dentro deste estudo uma variável conhecida como ICC(Individual CDD *commit*) para realizar o comparativo de complexidade com o próprio CDD.

Concluindo que as questões levantadas dentro deste estudo estão relacionadas ao próprio CDD, sua abordagem possui bastante relevância, sua compreensão e aplicação torna possível criação de código com com ICPs no limite ideal gerando complexidade aceitáveis. Através do estudo concluímos que o método possui importância impactante no ambiente de desenvolvimento, possibilitando a construção de código legíveis e com complexidade intrínseca arqueada ao entendimento dos desenvolvedores .

REFERÊNCIAS

BARBOSA, L. F., PINTO, V. H., SOUSA, A. L. O. T. d., & PINTO, G. (2022). **To What Extent Cognitive-Driven Development Improves Code Readability?**. ESEM '22: Proceedings of the 16th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement, 238 - 248. https://doi.org/10.1145/3544902.3546241

BARRETO, Márcio. **Percepção e realidade**. ClimaCom [online], Campinas, ano.4, n.9, Ago. 2017. Disponível em: http://climacom.mudancasclimaticas.net.br/?p=7288

FERREIRA, R., SOUZA, C. R. B., PINTO, V. H. S. C., & PINTO, G. (2024). **Assisting Novice Developers Learning in Flutter Through Cognitive-Driven Development**. SBES'24, 2024. 10.48550/arXiv.2408.11209

GAMMA, E.; HELM, R.; JOHNSON, R.; VLISSIDES, J. M. **Design Patterns: Elements of Reusable Object-Oriented Software**. 1. ed. Addison-Wesley Professional, 1994. ISBN 0201633612. Disponível em: https://www.amazon.com/Design-Patterns-Elements-Reusable-Object-Oriented/dp/0201633612/ref=ntt_at_ep_dpi_1

LIMA, Gercina & Â. Borém. **Interfaces between information science and cognitive science**. scielo, 2003. Disponível em: https://www.scielo.br/j/ci/a/QbYSX39CjbBrMLbqcqsfN6h/?lang=pt

MILLER, G. A. (1956). **The magical number seven, plus or minus two: Some limits on our capacity for processing information**. Psychological Review, 63(2), 81-97. https://doi.org/10.1037/h0043158

PEREIRA, J. H., SOUZA, A. L., & PINTO, V. H. (2021). **Cognitive Load Analyzer: A Support Tool for Cognitive-Driven Development**. SBES '21: Proceedings of the XXXV Brazilian Symposium on Software Engineering, 468 - 473. https://doi.org/10.1145/3474624.3476011

PINTO, G., & SOUZA, A. (2023). **Cognitive-Driven Development Helps Software Teams to Keep Code Units Under the Limit!** Journal of Systems and Software, 206(2023).

https://doi.org/10.1016/j.jss.2023.111830

PINTO, V. H. S. C., & SOUZA, A. L. O. T. (2022). **Effects of Cognitive-Driven Development in the Early Stages of the Software Development Live Cycle**. In Proceedings of the 24th International Conference on Enterprise Information Systems (ICEIS 2022), 2, 40 - 51. 10.5220/0011009000003179

PINTO, V. H. S. C., SOUZA, A. L. O. T., OLIVEIRA, Y. M. B., & RIBEIRO, D. M. (2021). **Cognitive-Driven Development: Preliminary Results on Software Refactoring**. Proceedings of the 16th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2021), 92 - 102. 10.5220/0010408100920102

SCHUNK, D. H. (2012). *Learning Theories: An Educational Perspective* (6th ed.). Pearson. Este livro fornece uma visão abrangente das diversas teorias de aprendizagem e é amplamente respeitado no campo da educação.

SILVÉRIO, G. Camargo; ROSAT, R. Menezes. **Memória de longo-prazo: mecanismos neurofisiológicos de formação**. RMMG, Disponivel em: http://rmmg.org/artigo/detalhes/577

SOUZA, A. (2021, July 8). **Cognitive-Driven Development (CDD): o que é e mais**. Zup. Retrieved May 28, 2024, from https://www.zup.com.br/blog/cognitive-driven-development-cdd

SOUZA, A. L. O. T., & PINTO, V. H. S. C. (2020). **Toward a Definition of Cognitive-Driven Development**. IEEE International Conference on Software Maintenance and Evolution (ICSME), 776-778. 10.1109/ICSME46990.2020.00087

STRINGFIXER. **Modelo de filtro de atenção de Broadbent**. Stringfixer,. Disponível em: https://stringfixer.com/pt/Broadbent's_Filter_Model_of_Attention.Acesso.em:09/03/2022.

ZUP. **Cognitive-Driven Development (CDD): proposta de design orientado ao entendimento,** Zup, 08 de julho de 2021. Disponível em:https://www.zup.com.br/blog/cognitive-driven-development-cdd . Acesso em: 03 de agosto de 2022.

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DA PARAÍBA
Campus Cajazeiras - Código INEP: 25008978
Rua José Antônio da Silva, 300, Jardim Oásis, CEP 58.900-000, Cajazeiras (PB)
CNPJ: 10.783.898/0005-07 - Telefone: (83) 3532-4100

Documento Digitalizado Ostensivo (Público)

TCC 2 Final

Assunto:	TCC 2 Final
Assinado por:	Izaquiel Silva
Tipo do Documento:	Avaliação
Situação:	Finalizado
Nível de Acesso:	Ostensivo (Público)
Tipo do Conferência:	Cópia Simples

Documento assinado eletronicamente por:

- **Izaquiel Canuto da Silva**, **ALUNO (201722010026) DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS - CAJAZEIRAS**, em 21/10/2024 16:32:15.

Este documento foi armazenado no SUAP em 21/10/2024. Para comprovar sua integridade, faça a leitura do QRCode ao lado ou acesse https://suap.ifpb.edu.br/verificar-documento-externo/ e forneça os dados abaixo:

Código Verificador: 1285988
Código de Autenticação: a0e799a3ec