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

AVALIAÇÃO DOS PADRÕES DE PROJETO EM DIFERENTES LINGUAGENS DE PROGRAMAÇÃO

RAUL COELHO CABRAL

Cajazeiras
Junho, 2021

RAUL COELHO CABRAL

AVALIAÇÃO DOS PADRÕES DE PROJETO EM DIFERENTES LINGUAGENS DE PROGRAMAÇÃO

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. Ricardo de Sousa Job.

Cajazeiras
Junho, 2021

Campus Cajazeiras
Coordenação de Biblioteca
Biblioteca Prof. Ribamar da Silva
Catalogação na fonte: Daniel Andrade CRB-15/593

C117a

Cabral, Raul Coelho

Avaliação dos padrões de projeto em diferentes linguagens de programação / Raul Coelho Cabral; orientador Ricardo de Sousa Job.- Cajazeiras, 2021.
46 f.: il.

Orientador: Ricardo de Sousa Job.
TCC (Curso de Análise e Desenvolvimento de Sistema) – Instituto Federal de Educação, Ciência e Tecnologia da Paraíba, Cajazeiras, 2021.

1. Padrões de projeto 2. Vocabulários de software 3. Detecção de Padrões - Desenvolvimento de software 4. Qualidade de software I. Título.

004.41(0.067)

SERVIÇO PÚBLICO FEDERAL
INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DA PARAÍBA
COORDENAÇÃO DE CURSOS - CAMPUS CAJAZEIRAS

Às **15:30** horas do dia **16** do mês de **junho** do ano de **2021**, via Google Meet, compareceu para defesa pública do **Trabalho de Conclusão de Curso**, requisito obrigatório para a obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas, o(a) aluno(a) **RAUL COÊLHO CABRAL**, matrícula **201622010256**, tendo como Título do Trabalho **AVALIAÇÃO DOS PADRÕES DE PROJETO EM DIFERENTES LINGUAGENS DE PROGRAMAÇÃO**. Constituíram a Banca Examinadora os professores **RICARDO DE SOUSA JOB** (orientador), **FÁBIO ABRANTES DINIZ** (examinador) e **JANDERSON FERREIRA DUTRA** (examinador).

Após a apresentação e as observações dos membros da Banca Examinadora, ficou definido que o trabalho foi considerado **APROVADO** com nota **95**, com a condição de que o (a) aluno (a) entregue, no prazo máximo de 30 dias, a versão final do trabalho, via processo eletrônico à coordenação de curso. A versão deve conter a ficha catalográfica e atender às sugestões feitas pelos membros da banca. O código fonte desenvolvido no trabalho (caso haja) deve ser enviado para o e-mail da coordenação do curso (cads.cz@ifpb.edu.br).

Cajazeiras-PB, 19 de junho de 2021.

Documento assinado eletronicamente por:

- **Raul Coêlho Cabral, ALUNO (201622010256) DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS - CAJAZEIRAS**, em 21/06/2021 14:02:29.
- **Fabio Abrantes Diniz, PROFESSOR ENS BASICO TECN TECNOLOGICO**, em 21/06/2021 08:39:45.
- **Janderson Ferreira Dutra, PROFESSOR ENS BASICO TECN TECNOLOGICO**, em 19/06/2021 14:37:55.
- **Ricardo de Sousa Job, PROFESSOR ENS BASICO TECN TECNOLOGICO**, em 19/06/2021 10:41:49.

Este documento foi emitido pelo SUAP em 18/06/2021. 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: 199009
Código de Autenticação: 84c6b6fd12

A minha Mãe Liana Coelho, minha Rainha.
Ao meu Pai Joaquim Cabral, minha Base de coração puro.

*Ao meu tio **Casemiro Coelho** (in memorian).*
*Ao meu tio **Joaquim Coelho** (in memorian).*

Esse foi o ultimo ciclo. Obrigado por serem as referencias de bons homens para todos.

AGRADECIMENTOS

A minha mãe **Liana** e meu pai **Joaquim** por todas as orações em momentos difíceis, por mais longe que eu estivesse. A minha irmã **Ruthe** por todo apoio.

A toda família, meus mais sinceros agradecimentos. Em especial a minha avó materna **Nescir**

A todos os professores do instituto, que me fizeram uma pessoa melhor. A **Ricardo Job** meu professor e orientador, por todos os conselhos e ensinamentos desse começo.

Dentre todos esses anos de percurso, muitas pessoas se tornaram importantes e o tempo e os acontecimentos deixaram de ser, quem sabe um dia nos encontraremos novamente.

Em especial ao grupo dos **"Paulo Renjes"** as melhores pessoas que eu já conheci. Juntos, conseguimos passar por todas as dificuldades desta jornada.

Em especial a meu amigo **Romulo**, que sempre esteve presente quando eu mais precisei. Obrigado irmão!.

“Sasageyo! Sasageyo! Shinzou Wo Sasageyo!!.”

Erwin, Shingeki no Kyojin

RESUMO

Boas práticas de programação como o uso de padrões de projeto estão cada vez mais comuns no ambiente de desenvolvimento, em busca da melhoria da qualidade de software. Portanto, a implementação falha dos padrões de projeto compromete a qualidade do software. Por sua vez, os padrões de projeto proporcionam aos desenvolvedores um vocabulário de software em comum, além de ajudar a se comunicar com outros desenvolvedores por meio de termos próprios. Sendo assim, é importante para o desenvolvimento desenvolvimento de software a identificação e detecção dos padrões de projeto para melhoria da qualidade, encontrando instâncias de padrões de projeto em um sistema de software que pode fornecer dicas úteis para a compreensão da arquitetura do mesmo e desenvolvendo uma abordagem baseada nas métricas recolhidas por meio do reconhecimento dos vocabulários de software, a fim de avaliar o uso dos padrões de projeto nas diferentes linguagens. O protótipo desenvolvido neste trabalho tem o intuito de testar as principais premissas sobre a abordagem da pesquisa. A extração das estruturas de classe por meio de análise estática para verificação sem a necessidade dos projetos compilados foi feita com a utilização da ferramenta de reconhecimento de texto estruturado **ANTLR4**. O reconhecimento dos padrões de projeto a partir do **.jar**, está sendo realizado com a ferramenta ***Design Pattern Detection (DPD)***.

Palavras-chave: Padrões de projeto. Vocabulários de software. Detecção de Padrões. Qualidade de software.

ABSTRACT

Good programming practices such as the use of design patterns are increasingly common in the development environment, seeking to improve software quality. Therefore, failure to implement design patterns compromises the quality of the software. Design patterns, in turn, provide developers with a common software vocabulary and help to communicate with other developers through these proprietary terms. Therefore, it is important for development that we know how to identify and detect these design patterns for quality improvement, finding instances of design patterns in a software system that can provide useful tips for understanding the architecture itself and developing an approach based on the metrics collected through recognition of software vocabularies in order to evaluate the use of design patterns in different languages. The prototype developed in this work is intended to test the main assumptions about the research approach. The extraction of class structures through static analysis for verification without the need of compiled projects was done through **ANTLR4**. Recognition of design patterns from the **.jar** is being performed with the **Design PatternDetection (DPD) tool**.

Keywords: Design patterns. Software vocabularies. Pattern Detection. Software quality.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

DP	Padrões de Projeto
DPD	Detecção de Padrões de Projeto
SSA	*Similarity Scoring Approach*
OSS	*Open Source Software*
AST	*Abstract Syntax Tree*
ASG	*Abstract Syntax Graphic*
GQM	*Goal Question Metric*

SUMÁRIO

1 INTRODUÇÃO

Este capítulo do trabalho descreve o contexto no qual a pesquisa foi escrita, detalhando os conceitos de qualidade de software e como está diretamente ligada com o uso de padrões de projeto no desenvolvimento de software. Neste contexto, destaca-se a importância na detecção de padrões de projeto para avaliação da qualidade de software. Em seguida detalhando a motivação a respeito de boas práticas de programação para manter a qualidade de software provendo um *overview* sobre padrões de projeto. Assim, apresenta-se a problemática deste trabalho, demonstrando falhas das ferramentas de detecção, além de más práticas relacionadas ao mau uso dos padrões de projeto no desenvolvimento. Seguidamente, são descritos os objetivos deste trabalho. Por fim, é descrito o conteúdo da estrutura do trabalho.

1.1 CONTEXTO

Na comunidade de desenvolvimento de software, deve-se dar a devida importância a garantia da qualidade de software. Escrever um código de qualidade e de fácil compreensão é o objetivo de quase todo desenvolvedor. Em um sentido amplo, um processo eficaz de software é aplicado de maneira a criar um produto útil que forneça valor mensurável para quem o produz e para quem o usa (PRESSMAN, 2005). Existem boas práticas e técnicas de programação conhecidas e difundidas que possuem o intuito de agilizar determinadas tarefas. Essas técnicas ajudam o desenvolvedor a codificar de forma mais legível, auxiliando a equipe de desenvolvimento, oferecendo maior qualidade e reduzindo custos. Dentre essas técnicas, destacam-se os padrões de projeto.

A história inicial dos padrões de software começa não com um cientista da computação, mas com um arquiteto de construção, Christopher Alexander, que reconheceu que um conjunto recorrente de problemas era encontrado sempre que uma construção era projetada. Ele caracterizou esses problemas recorrentes e suas soluções como padrões, descrevendo-os na forma de que cada padrão descreve um problema que ocorre repetidamente em nosso ambiente e, em seguida, descreve o núcleo da solução para esse problema, de forma que você possa usar a solução um milhão de vezes sem nunca fazê-lo da mesma maneira duas vezes (PRESSMAN, 2005).

As ideias de Alexander foram traduzidas pela primeira vez no mundo do software nos livros de Gamma (GAMMA et al., 1995), Buschmann (SCHMIDT et al., 2013) e seus muitos colegas. Atualmente, existem diversos catálogos e algumas dezenas de

repositórios de padrões. O *design* baseado em padrões pode ser aplicado em diversos domínios e sistemas diferentes.

No livro seminal sobre padrões de projeto (Padrões de Projetos - Soluções Reutilizáveis) (GAMMA et al., 1995), encontra-se uma descrição em que cada padrão de projeto descreve um problema no nosso ambiente e apresenta sua solução, de tal forma que você possa usar essa solução mais de uma vez em outros casos do mesmo problema, sem nunca repeti-la da mesma maneira. Resumidamente, padrões de projetos são soluções para problemas recorrentes e que mantém diretrizes sobre como lidar com determinados problemas.

Diante a relevância da qualidade de software, se as características não forem satisfeitas, como a implementação dos padrões de projeto for feitos de forma incorreta compromete a qualidade interna e externa do software, como a maioria dos projetos de software atuais lidam com produtos em evolução que consistem em um grande número de componentes, que pode tornar sua arquitetura complicada e bastante confusa (TSANTALIS et al., 2006). Os padrões de projeto podem impor estrutura ao sistema devido às abstrações que estão sendo usadas. Consequentemente, a identificação dos padrões de projeto implementados pode ser útil para a compreensão de um projeto existente e fornecer as bases para novas melhorias (VOKÁC, 2006).

É importante ao desenvolvimento que seja possível detectar e identificar os padrões de projeto para uma melhor compreensão do sistema. Para detecção automática de padrões existem basicamente duas técnicas: análise estática e dinâmica. A primeira busca avaliar os componentes e características, sem executar o software propriamente dito. A segunda se preocupa apenas com os dados inseridos e se a saída de dados inseridos é a esperada, qual é o tempo de resposta, qual o comportamento funcional e a performance da aplicação como um todo.

A falta de precisão de algumas ferramentas na detecção de padrões deixam a desejar quando há a necessidade de um projeto compilado para que a detecção ocorra limitando a quantidade de dados que poderiam ser recolhidos a uma população muito menor(FONTANA et al., 2012).

Apesar de existirem diversas técnicas para a detecção, é de alta relevância desenvolver uma abordagem de detecção dos padrões de projeto que traga uma forma diferente de se detectar. No geral algumas ferramentas de detecção recebem como dados de entrada projetos já compilados, a abordagem proposta sugere a utilização de projetos não compilados diretamente de repositórios *open-source* como no github[1]

[1] http://github.com/

como base para o experimento.

1.2 MOTIVAÇÃO

Uma forma de garantir que os produtos mantenham um bom nível de qualidade é ter um processo de desenvolvimento de software com qualidade, atentando sempre ao Processo de Desenvolvimento, buscando sempre a melhoria com boas práticas de programação tais como: a utilização de ferramentas, *frameworks* e metodologias que fazem com que a qualidade do Processo de Desenvolvimento seja algo natural no trabalho de todas as pessoas envolvidas e boas práticas de programação como o uso de padrões de projeto.

Entretanto, avaliar a qualidade de software é difícil, visto que é preciso definir uma boa métrica e uma boa forma de mensurá-la de tal forma que reflita o que é esperado como resultado do processo. A métrica puramente pode não prover respostas imediatas, mas os indicadores provindos de métricas sólidas afim de dar um indício do que está acontecendo e comparando com informações e experiências anteriores, dar uma visão do que pode vir a ocorrer no futuro. Mesmo que uma previsão tenha uma certa margem de erro, limitando um conjunto de possibilidades infinitas em um conjunto de resultados menor, e isso é o que se deseja ter para que se possa atuar dentro de um projeto.

Com a posse destes indicadores é possível criar uma ferramenta que auxilie, na prática, o desenvolvedor tanto a visualizar informações sobre o código, quanto para sugerir uma mudança em determinada classe afim de diminuir o acoplamento entre as classes e a complexidade do código, além de propiciar a estruturação explícita do padrão como conhecida, facilitando, assim, a documentação e comunicação do projeto.

1.3 PROBLEMÁTICA

O termo qualidade está de modo geral associado a produtos e serviços. Dentro do contexto do desenvolvimento onde está sendo cada vez mais relevante diante a situação que os sistemas se encontram hoje, onde o objetivo é atingir a satisfação total do cliente e reduzir as variações de processo e atingir a melhoria da qualidade contínua com os conceitos de qualidade, além da própria qualidade do sistema, tornaram-se cada vez mais prioridade.

Por sua vez, os padrões de projeto proporcionam aos desenvolvedores um vocabulário de software em comum que os permite se comunicar com outros desenvolvedores de forma mais coerente. O uso de vocabulário de software está diretamente

associado aos padrões de projeto, pois ao utilizar padrões de projeto fica evidente a incorporação dos nomes dos padrões de projeto ao vocabulário e que seu uso substitui longas descrições (ALUR et al., 2003).

Contudo, existem cenários em que os desenvolvedores usam os vocabulários de software para indicar que em um determinado conjunto de classes há um possível padrão de projeto implementado, sendo que na verdade não há. Essas ocorrências são denominados de falsos padrões de projeto.

Um falso padrão de projeto é uma possível tentativa de instância de um padrão nos quais as classes, interfaces, métodos e atributos envolvidos possuem terminologias referentes ao padrão, porém, não atendem as estruturas que descrevem o padrão de projeto (SEVERO; JOB, 2019). Ao indicar um falso padrão de projeto no código, o desenvolvedor pode ter tido a intenção de implementar o padrão de projeto, mas feito isso de forma incorreta.

Geralmente são adotados ferramentas que detectam padrões de projeto a fim de avaliar a qualidade de software em questão. A falta de precisão de algumas ferramentas de detecção de padrões de projeto em softwares de código aberto falham na detecção em sistemas muito grandes e nas diferentes linguagens de programação.

1.4 OBJETIVOS

Este trabalho tem como principal objetivo desenvolver a abordagem baseado nas métricas recolhidas por meio do reconhecimento dos vocabulários de software a fim de avaliar o uso dos padrões de projeto nas diferentes linguagens de programação. Para atingir este objetivo, propõe-se aqui os seguintes objetivos específicos:

- Estudar técnicas e aplicação de detecção de padrões de projeto;
- Estudar técnicas de leitura e reconhecimento de vocabulários de software por meio da gramática
- Identificar abordagens existentes dentro do contexto de detecção de padrões de projeto;
- Propor um experimento para avaliar os padrões de projeto em várias linguagens de programação
- Avaliar os padrões a partir das métricas coletadas

1.5 ORGANIZAÇÃO DO DOCUMENTO

Este documento está organizado em cinco seções para um melhor entendimento da proposta deste trabalho que está sendo apresentada. Na seção 3 consta o referencial teórico sobre o presente trabalho que está sendo submetido, além de apresentar alguns trabalhos relacionados. Na seção 4 apresenta a abordagem proposta com etapas utilizadas para o desenvolvimento do experimento e as considerações finais sobre. Por fim as conclusões do presente trabalho, trazendo as métricas trabalhadas, problemas enfrentados, resultados alcançados.

2 FUNDAMENTAÇÃO TEÓRICA

Na engenharia de software uma questão crucial é a manutenibilidade do código, pois, quando em 67% do custo total do software encontra-se na fase de desenvolvimento. A manutenção depende fortemente da qualidade do código desenvolvido. Por isso, alguns princípios e padrões são propostos para que se tenham um código com maior qualidade (GUO et al., 2011). Por fim, os critérios de avaliação a respeito das métricas de desempenho para avaliação da abordagem proposta.

Nesta seção primeiramente é analisado o uso de vocabulários de software para identificar especialistas em entidades de código. Logo após, o contexto de padrões de projeto e sua importância na qualidade de software. Adentrando assim a respeito de detecção de padrões de projeto, uma atividade útil na engenharia reversa para obter conhecimento sobre os problemas de design de um sistema existente, sobre sua arquitetura de software e qualidade de projeto, melhorando assim a compreensão do sistema e, portanto, sua manutenção e evolução.

2.1 VOCABULÁRIOS DE SOFTWARE

De acordo com a literatura, vocabulário de software é uma fonte valiosa de informação sobre o projeto (processo e produto) que deveria ser utilizada para auxiliar *stakeholders* durante as atividades de desenvolvimento e manutenção de sistemas (SANTOS et al., 2015). O vocabulário do software representa cerca de 70% do código fonte, onde seus identificadores em parte, são nomeados de acordo com conceitos, abstrações e funcionalidades que representam perante o sistema.

Estudos sobre identificadores e comentários, que constituem o vocabulário de software, apontam que capturam aspectos subjetivos e inerentes de quem os codifica ou os escreve, tais como: quantidade mínima e máxima de palavras para constituir um identificador, predileção por usar determinados termos ou palavras (SANTOS et al., 2015).

2.2 ANTLR4

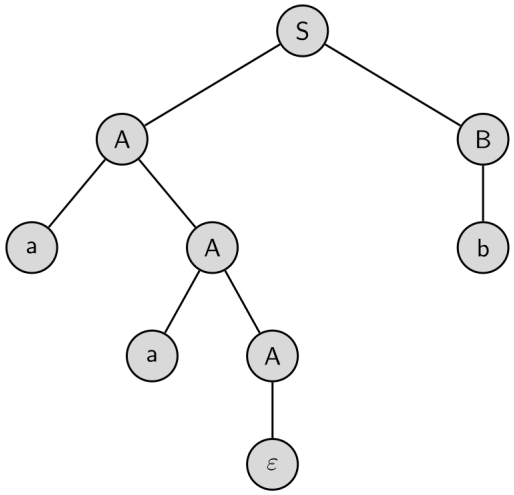
A **ANTLR4** foi a ferramenta escolhida para fazer o reconhecimento da gramática da linguagem, por ser um poderoso gerador de analisador para leitura, processamento, execução ou tradução de texto estruturado ou arquivos binários em C#, Python 2 e 3, JavaScript, Go, C ++ e Swift. Ou seja, um analisador também pode ser gerado para esses idiomas. É amplamente usado para criar linguagens, ferramentas e estruturas.

2.2.1 Etapas da análise do texto

Parece que entender que qualquer texto é um problema em aberto. Logo assume que com um subconjunto que cumpre várias regras específicas conhecidas como gramática.

Inicialmente é levado o texto em pedaços ou constituintes chamados "lexemes". Ou seja é feito a análise lexical. Lexemes são as palavras em um texto. Permitindo ver se os sistemas cumprem as regras descritas. Lexemes são usados para formar estruturas mais complexas e, novamente, devemos verificar se eles cumprem as regras. Como resultado é obtido uma árvore de análise gramatical. Esta é a árvore em que a regra principal está na raiz, as regras entrelaçadas estão nos nós e os lexemas específicos são encontrados nas folhas como demonstrado na representação da árvore de análise visto na Figura 1.

Figura 1 – Árvore de Análise



Fonte: Elaborado pelo autorr, 2021

Também pode ser verificado se nosso texto está em conformidade com as regras que não podem ser cumpridas, mesmo usando a gramática. O que é chamado de analisador semântico. Se erros forem detectados, e feito o processamento do erro e demostrado. Se não houver erros ou os erros não forem críticos, analisa-se a árvore de análise.

Várias são as ferramentas e técnicas que buscam auxiliar o desenvolvedor, e demais envolvidos no processo de produção do software, trazendo mais produtividade à sua rotina de qualidades, bem como mais qualidade ao projeto em desenvolvimento. Uma destas técnicas consiste em análise estática, ou seja, sem a necessidade de que este seja executado. A partir desta técnica é possível identificar os padrões de

projeto sem a necessidade de um projeto compilado e extrair métricas que servirão como parâmetros para a análise do código do projeto sobre aspectos e a fim de avaliar a qualidade dos padrões implementados.

2.3 PADRÕES DE PROJETO

A manutenção e qualidade do software podem estar relacionados ao uso de padrões de projeto. Os padrões de projeto são descrições de objetos e classes comunicantes que precisam ser personalizadas para resolver um problema geral de projeto em um contexto específico (GAMMA et al., 1995).

Uma das principais motivações ao introduzir padrões de projeto é fornecer um vocabulário comum aos desenvolvedores, que correspondam a soluções de design reutilizáveis, flexíveis e de manutenção (AMPATZOGLOU et al., 2010). Além disso segundo (ARNOUT; MEYER, 2006), descrevem como os padrões de projeto orientado a objetos podem ser transformados em componentes reutilizáveis. Na literatura, uma grande variedade de estudos tentou avaliar o impacto da aplicação dos padrões de projeto na qualidade do software. Esses estudos, principalmente os empíricos, sugerem que os padrões de projeto orientado a objetos não são universalmente bons ou ruins (GAMMA et al., 1995).

Além disso, os padrões de projeto fornecem um mecanismo de reutilização, ajudando os desenvolvedores e os arquitetos a evitar uma re-implementação de uma solução existente. Contudo, analisar um projeto para identificar e entender os padrões que estão implementados sempre é uma tarefa árdua. Além disso, os desenvolvedores podem implementar variações dos padrões, tornando essa tarefa de identificação ainda mais complexa.

2.4 DETECÇÃO DE PADRÕES

A detecção de padrão de projeto (DPD) é um tópico que recebeu grande interesse nos últimos anos. Encontrar instâncias de padrão de projeto (DP) em um sistema de software pode fornecer dicas úteis para a compreensão de um sistema de software e sobre que tipo de problemas foram abordados durante o desenvolvimento do próprio sistema. Além disso, são importantes durante o processo de re-documentação, em particular quando a documentação é ruim, incompleta ou não está atualizada. Várias abordagens e ferramentas de DPD foram desenvolvidas, explorando diferentes técnicas para a detecção, como lógica nebulosa, técnicas de solução de restrições, provadores de teoremas, métodos de correspondência de modelos e técnicas de classificação (GUÉHÉNEUC; ANTONIOL, 2008; PRESSMAN, 2005).

Apesar das muitas abordagens propostas, os resultados obtidos costumam ser bastante insatisfatórios e diferentes de uma ferramenta para outra. Muitas ferramentas encontram muitos candidatos a padrões falsos positivos, mas outros corretos não são encontrados. Um problema comum no DPD é o chamado problema de variante: os DPs podem ser implementados de várias maneiras diferentes umas das outras. As principais variantes de cada padrão são descritas no catálogo de Gamma et al (GAMMA et al., 1995). Outros padrões são aplicados quando o contexto do aplicativo os exigem. Essas variantes causam a falha da maioria das ferramentas de reconhecimento de instância de padrões usando abordagens de detecção rígidas, baseadas apenas em definições de padrões canônicos. Por isso, é importante a comparação dos resultados fornecidos pelas diferentes ferramentas, para poder avaliar a melhor abordagem.

A pesquisa em detecção de padrões de projeto ainda enfrenta uma série de desafios importantes, como as abordagens de detecção atuais estão trabalhando independentemente umas das outras (FONTANA et al., 2012), tornando a avaliação de abordagens de detecção de padrão de projeto mais complexa, pois a maioria das ferramentas de detecção atuais não estão disponíveis aos desenvolvedores e pesquisadores.

As abordagens de detecção de padrões de projeto executam etapas importantes semelhantes para o reconhecimento de padrões. Essas etapas estão relacionadas à extração de informações do projeto e do código fonte. Esta pesquisa se concentra apenas nos padrões de projeto do Gang of Four (GAMMA et al., 1995) (GoF). Obviamente, isso não sugere que os padrões do GoF sejam melhores que outros tipos de padrões (por exemplo, padrões de arquitetura e padrões de idiomas). Algumas abordagens de detecção aplicaram os experimentos em programas de tamanho pequeno usando alguns padrões e atingiram alta precisão e taxas de recuperação.

Além disso, a maioria das abordagens de detecção depende do nível de código para detecção de padrões, que usa o código-fonte como entrada e o representa em um dos formatos de análise a Árvore de Sintaxe Abstrata (AST) ou Gráfico de sintaxe abstrata (ASG). A AST é usada em ferramentas de linguagem, como compiladores, tradutores e transformadores de linguagem, além de analisadores; para remover a sintaxe e, portanto, é uma construção ideal para uma ferramenta independente de idioma (Owens; Anderson, 2013).

2.4.1 Detecção baseada em Métricas

As abordagens de detecção de padrões baseadas em métricas calculam as métricas relacionadas ao programa, como agregações, associações e dependências

de diferentes representações de código-fonte. Outras diferentes técnicas são aplicadas para comparar valores, como a comparação de métricas de padrões identificados com métricas de código-fonte. As abordagens baseadas em métricas reduzem o espaço de pesquisa através da filtragem. Essas abordagens representam as informações estruturais e comportamentais do sistema de destino como estrutura, gráfico ou matriz UML. A maioria dessas abordagens tem boas taxas de precisão[1] e recall[2] mas não são capazes de lidar com as variantes de implementação dos padrões de projeto.

2.4.2 Representação intermediária do código fonte.

Sabe-se que todas as abordagens de detecção na literatura estão direcionadas ao código fonte do sistema em estudo e evitam direcionar o modelo de projeto do sistema para extrair as instâncias dos padrões de projeto. O modelo de projeto não fornece dados de tempo de execução, necessários para a extração de padrões de projeto (por exemplo, os relacionamentos de associação). Normalmente, os documentos de projeto são inconsistentes com o código fonte. Além disso, a maioria dos modelos de projeto não está disponível ao público (FONTANA et al., 2012).

Todos esses motivos fizeram do código fonte uma escolha melhor do que o modelo de projeto para extrair as instâncias dos padrões de projeto. A maioria das abordagens de detecção de padrão usa a representação *Abstract Syntax Tree* (AST) para gerar o modelo de código-fonte, já que é uma forma mais refinada da *Syntax Tree* com acesso as informações presentes nos nós folha. O modelo de código-fonte contém todas as informações necessárias para recuperar instâncias de padrão de projeto.

2.4.3 Critério de Avaliação

A escolha de uma métrica de desempenho geralmente depende do problema de negócios que está sendo resolvido. Diga-se que você tenha 100 exemplos em um conjunto de dados, e foi alimentado cada um com seu modelo e recebeu uma classificação. A classificação prevista vs. real pode ser representada em uma tabela como na Tabela 1 chamada matriz de confusão.

A Tabela 1 descreve um resultado negativo vs positivo. Esses dois resultados são as "classes"de cada exemplo. Como existem apenas duas classes, o modelo usado para gerar a matriz de confusão pode ser descrito como um classificador binário .

Métricas de *precision* e *recall* foram usadas pela maioria das abordagens para avaliar a precisão do processo de detecção. Algumas abordagens relataram o *F-Score*,

[1] https://medium.com/@shivangisareen/precision-recall-accuracy-6a214187f059
[2] https://towardsdatascience.com/precision-vs-recall-386cf9f89488

Tabela 1 – Representação em termos Previsto vs Real.

	Negativo (previsto)	**Positivo (Previsto)**
Negativo (Real)	Verdadeiro Negativo	Falso Positivo
Positivo (Real)	Falso Negativo	Verdadeiro Positivo

Fonte: Elaborado pelo autor, 2021

que fornece o equilíbrio entre *precision* e o *recall*.

A precisão pode ser calculada como mostrado na Equação 1, e difere de uma abordagem para outra, uma vez que algumas extraíram alguns padrões e alcançaram alta precisão. A precisão ajuda quando os custos de falsos positivos são altos. Então, vamos supor que o problema envolve a detecção de padrões de projeto. Se tivermos um modelo com precisão muito baixa, muitos dos padrões não serão identificados, e isso incluirá alguns erros de diagnóstico. Quando os **falsos positivos** são muito altos, quem monitora os resultados aprenderá a ignorá-los depois de ser bombardeado com alarmes falsos. O método de validação, definições de padrões também podem afetar a precisão da detecção.

$$Precision = \frac{VerdadeirosPositiovos(TP)}{VerdadeirosPositiovos(TP) + FalsosPositivos(FP)} \tag{1}$$

O *Recall* pode ser visto como a probabilidade de um documento relevante ser recuperado pela consulta. É trivial obter 100% de recuperação retornando nesse caso todos os padrões de projeto que foram implementados na aplicação em resposta a qualquer detecção. O cálculo do *recall* é mostrado na Equação 2

$$Recall = \frac{VerdadeirosPositiovos(TP)}{VerdadeirosPositiovos(TP) + FalsosNegativos(FN)} \tag{2}$$

F-score é uma medida geral da precisão de um modelo que combina *precision* e *recall*. Ou seja, um *F-score* é considerado perfeito quando é 1, significa que se tem baixos **falsos positivos** e **baixos falsos negativos**, portanto, está sendo identificado corretamente ameaças reais e não é perturbado por alarmes falsos. Enquanto o modelo é uma falha total quando é 0, o cálculo pode ser visto na Equação 3.

$$\textit{F1 Score} = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{3}$$

2.5 TRABALHOS RELACIONADOS

Durante o trabalho desenvolvido foram estudadas diversas tecnologias que têm objetivos semelhantes ao objetivo principal deste trabalho. Os trabalhos estudados apresentam modelos de detecção de padrões de projeto que adotam diferentes paradigmas à detecção, a relação de qualidade de software entre projetos *open-source* além da identificação de especialistas de código com base no vocabulário de software identificado.

Os principais trabalhos que foram estudados para apresentação desta seção, são o **SSA** *(Design Pattern Detection Using Similarity Scoring)* (TSANTALIS et al., 2006), como a representação de uma abordagem de detecção de padrões de projeto baseada na similaridade gráfica entre as vértices do padrão e do sistema em estudo. O **OSS** *(An Empirical Study on Design Pattern Usage on Open-Source Software)* (AMPATZOGLOU et al., 2010), que é um estudo empírico sobre como atualmente as comunidades de software de código aberto estão prosperando e o número de projetos disponíveis por meio de repositórios de códigos conhecidos está aumentando rapidamente. Mais especificamente, este estudo relata resultados empíricos com base no número e tipo de padrões de projeto recuperados de projetos de software de código aberto. O último trabalho *(Using Developers Contributions on Software Vocabularies to Identify Experts)*, se trata de analisar o uso de tais vocabulários de software para identificar especialistas em entidades de código.

2.5.1 *Design Pattern Detection Using Similarity Scoring* (SSA)

A detecção de padrões de projeto usando a *Similarity Scoring Approach* (**SSA**) é uma pesquisa desenvolvida em linguagem Java na Universidade da Macedônia para lidar com o problema de múltiplas variantes de padrões de projeto (TSANTALIS et al., 2006). O **SSA** descreve os padrões de projeto a serem detectados, bem como o sistema em estudo, como gráficos. Além disso, o SSA representa todas as informações estáticas do sistema como um conjunto de matrizes. O SSA usa um algoritmo de similaridade gráfica para detectar padrões de projeto, calculando a similaridade de vértices entre o padrão e o sistema em estudo. Para lidar com o problema de tamanho do sistema, O SSA divide o sistema em vários subsistemas e o algoritmo de similaridade é aplicado ao subsistemas em vez de todo o sistema. O SSA foi aplicado nas seguintes ferramentas: JHotDraw v5.1, uma estrutura gráfica bidimensional para editores de desenhos estruturados, JRefactory v2.6.24, uma ferramenta que permite realizar refatorações em projetos e JUnit v3.7, um framework open-source, que se assemelha ao raio de testes software java.

Além disso, o SSA usa matrizes para representar as relações entre as classes, que são gráficos direcionados que podem ser mapeados em uma matriz quadrada. Para preservar a validade dos resultados, o SSA pontuações de similaridade foram limitadas dentro do intervalo [0, 1]. Na verdade, o SSA tem várias limitações. Por exemplo, o SSA assume que não mais de uma característica para uma determinada instância de padrão de projeto é modificada. Por exemplo, um padrão possui duas caracteristicas, um limite igual a 1 e definido, o resultado da análise das métricas é 0,5 então esse padrão é adicionado aos resultados.

O resultado da abordagem deve ser interpretado de acordo com o número de padrões detectados corretamente no caso os verdadeiros positivos. Os falsos positivos são considerados instâncias do padrão que foram modificadas que não atendem a descrição do padrão que foi especificada. Falsos negativos são exemplos reais de padrões que não estão sendo detectados pela metodologia. A soma dos dos verdadeiros positivos e falsos negativos são iguais ao número total de instâncias dos padrões atuais do sistema, no entanto, o SSA mostra que o uso do algoritmo de similaridade produz resultados mais precisos do que o uso de correspondência exata/inexata de gráfico.

2.5.2 *An Empirical Study on Design Pattern Usage on Open-Source Software*

O desenvolvimento de um projeto *open-source* se baseia na colaboração. Um único desenvolvedor, ou um grupo de desenvolvedores, começa um projeto e lança uma versão que está disponível gratuitamente, por meio da internet, para uso e modificação. Este tipo de desenvolvimento tem vantagens e desvantagens. Uma desvantagem de desenvolvimento de software de código aberto é a falta de documentação e suporte técnico (AMPATZOGLOU et al., 2010). Considerando que, as principais vantagens do software de código aberto é o seu baixo custo, a sua fiabilidade e o fato de que fornecem seu código fonte para o usuário, para que seja capaz de personalizar o software de acordo com suas necessidades especiais.

O trabalho a ser referenciado tem como objetivo investigar o uso de padrões de projeto orientados a objetos em software open-source. Mais especificamente e empregado uma metodologia empírica, isto é, um estudo de caso, de modo a avaliar quais padrões são mais frequentemente aplicados em software de código aberto, que diferenças aparecem dentro de domínios de software e quais são os fatores mais influentes na aplicação padrão.

De acordo com (WOHLIN et al., 2012), existem três principais abordagens de investigação empírica, pesquisas, estudos de caso e experimentos. Considerando

a natureza e o assunto da pesquisa, foi selecionado para realizar um estudo de caso. A infinidade de projetos de código aberto consiste em estudos de caso como a abordagem ideal de pesquisa. Pelo contrário, as pesquisas não são adequadas para a pesquisa abordada, porque nesse caso o trabalho perderia os padrões empregados sem intenção pelos programadores. Finalmente, um experimento com programadores de código aberto diminuiria o número de sujeitos na pesquisa.

O estudo investiga empiricamente o uso de padrões de projeto orientado a objetos no desenvolvimento de software de código aberto. Por esse motivo, os autores exploraram 108 softwares de código aberto de três categorias, ou seja, ferramentas de desenvolvimento orientadas a objetos, aplicativos de comércio eletrônico e jogos de computador. Os resultados do estudo confirmam que padrões de projeto “fáceis de usar”, como Adapter, State e Singleton, são aplicados com mais frequência em código aberto. Padrões mais elaborados, como Visitor e Observer, são empregados com mais frequência por programadores de ferramentas de desenvolvimento orientados a objetos, provavelmente devido ao seu melhor entendimento e conhecimento sobre questões de engenharia de software. Em relação aos fatores que influenciam a aplicação do padrão, o tamanho e a atividade do projeto, ou seja, a duração, os downloads, o tamanho da equipe provou ser o mais importante. Pelo contrário, o número de versões não parece influenciar a atividade padrão. Por fim, a aplicação frequente do padrão Adapter em jogos de computador pode indicar níveis mais altos de reutilização nesse tipo de aplicativo.

2.5.3 Using Developers Contributions on Software Vocabularies to Identify Experts.

Um dos desafios diários no desenvolvimento de software é apoiar os líderes de equipe, gerentes de projeto a encontrar pontos eficientes no código-fonte (pontos rápidos e precisos), onde os desenvolvedores devem executar tarefas de manutenção. Porém, para que uma tarefa seja implementada com o mínimo de esforço, o menor custo possível e o menor espaço de tempo, também é necessário definir quem dentre os membros de uma equipe é o desenvolvedor mais apropriado para fazê-lo.

Os custos de manutenção representam entre 70% e 90% de um projeto de software (BENNETT; RAJLICH, 2000; BUSE; WEIMER, 2009). Durante as tarefas de manutenção, o entendimento do código leva de 50% (BENNETT; RAJLICH, 2000) a 78% do tempo do programador, enquanto 20% é para corrigir bugs e escrever código apenas 2%, (SANTOS et al., 2015). Identificar os membros certos de uma equipe de desenvolvimento que são especialistas em cada um dos projetos da entidade tem como objetivo a compreensão do código fonte, uma atividade que depende não apenas das

propriedades do código, mas também da experiência da pessoa (SANTOS et al., 2015).

Além disso, trechos de código criados por um desenvolvedor podem ser modificados por outro. As contribuições, feitas alterações, podem trocar o especialista em snippet de código do autor original para o maior desenvolvedor de contribuições (AMPATZOGLOU et al., 2010). As abordagens mais usadas para identificar especialistas são baseadas em registros de logs de mineração do VCS (Version Control System). Por exemplo, extrair o número de commits ou calcular a porcentagem de linhas de código modificadas (% LoC) (SANTOS et al., 2015).

As instruções do software são escritas de acordo com estritas regras gramaticais formais para evitar ambiguidade, e parte delas é formada por palavras reservadas e operadores de um idioma da linguagem a ser escrita. Os desenvolvedores são os autores das seqüências de caracteres, chamadas **identificadores**, que são usados para definir, referenciar e manipular os dois elementos estruturais básicos (por exemplo: atributos, variáveis locais, métodos e nomes de parâmetros) e os mais complexos. Por sua vez, comentários são textos também escritos por desenvolvedores usando linguagem natural com o objetivo de documentar explicitamente os elementos do código-fonte (SANTOS et al., 2015).

O estudo de caso do trabalho foi definido da forma: 1) seleção de projeto Java e amostras de entidades; 2) construção do oráculo de entidades especialistas; e 3) mensurar a expertise. Para isso, foi realizado um estudo de caso no projeto ePol3. A versão ePol (Sistema de Informações da Polícia Federal Brasileira) de acesso foi lançada em 18 de fevereiro de 2014, 0,5b. Naquela época, a equipe de desenvolvedores era composta por doze programadores e um líder de projeto. Em termos de tamanho, a ePol possui 907 entidades Java (classes e interfaces) em mais de 96K LOC(Linhas de Código), e seu vocabulário de software é composto por 2751 termos distintos. Como o trabalho está interessado no desenvolvedor mais adequado para realizar a manutenção de tarefas nas funcionalidades do sistema, foi descartado 436 entidades cuja função no código ePol é testar as outras 471 restantes (SANTOS et al., 2015).

Para processar o vocabulário da **ePol**, foi usado o *VocabularyTools4* que, para este objetivo de estudo, foi configurado para: extrair identificadores que nomeiam classes, interfaces, enumerações, atributos, métodos, parâmetros e variáveis locais; coletar comentários e textos javadocs; dividir identificadores codificados tanto em camelcase quanto em sublinhado; extrair raiz de palavras escritas em língua portuguesa; e, descartar termos com menos de 3 caracteres.

A literatura sobre o vocabulário do software fornece evidências das habilidades

dos desenvolvedores, preferências pessoais, bem como o mapa mental de um projeto, e traz aspectos de projeto não capturados pelas métricas estruturais tradicionais. Os resultados alcançados neste estudo aumentam a lista de uso de vocabulário: identificação de especialistas em código.

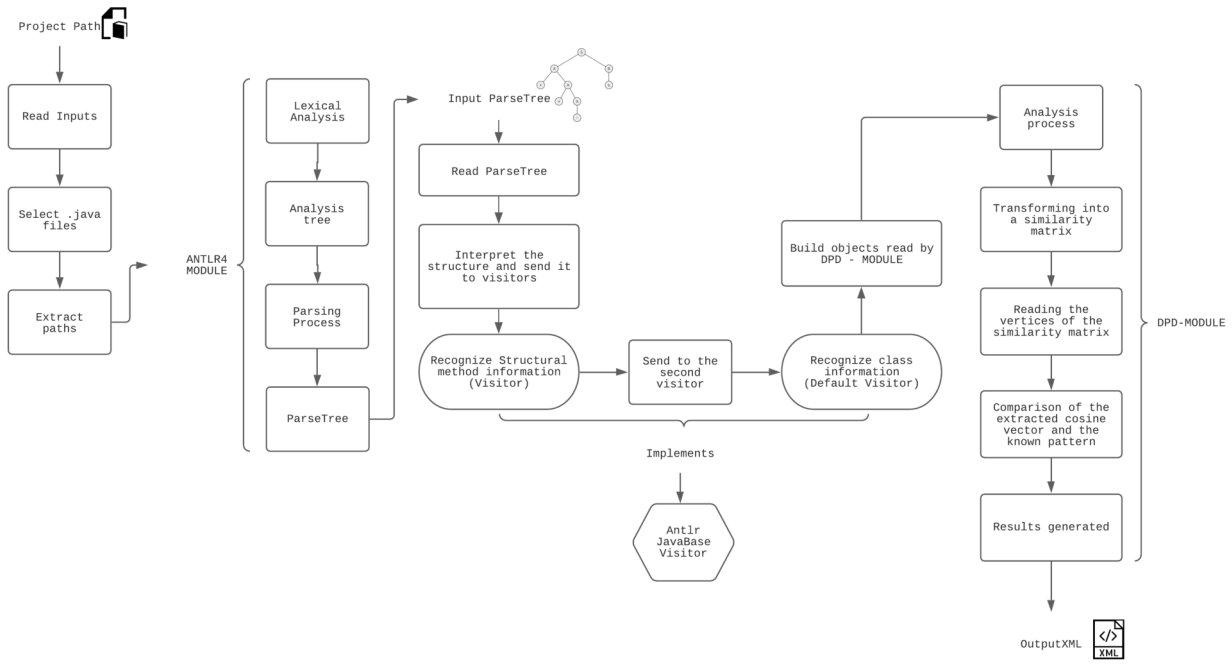
3 FERRAMENTA DESENVOLVIDA

A ferramenta desenvolvida é um protótipo preliminar para prova de conceito ou até mesmo como MVP (Produto Viável Mínimo). Durante a fase de testes e planejamento foi usada para aumentar a chance de sucesso deste trabalho, a **antlr-dpd**, realiza a análise estática sobre o código e a extração de informações sobre as estruturas de classe. O principal objetivo da ferramenta é a identificação e detecção acerca de padrões de projetos além de extrair informações sobre as chamadas de métodos e demais informações realizadas nas classes dos projetos Javas.

3.1 ANTLR-DPD

A antlr-dpd é uma ferramenta desenvolvida na linguagem Java e foi utilizada realizar todo o processo de identificação, detecção acerca de padrões de projeto definidos nos projetos acadêmicos selecionados. Gerando arquivos de *benchmarking* sobre as informações dos projetos indicando sobre a detecção de possíveis instâncias de padrões de projeto. A Figura 2 apresenta como a ferramenta está organizada.

Figura 2 – Representação dos módulos da ferramenta *Antlr-Dpd*



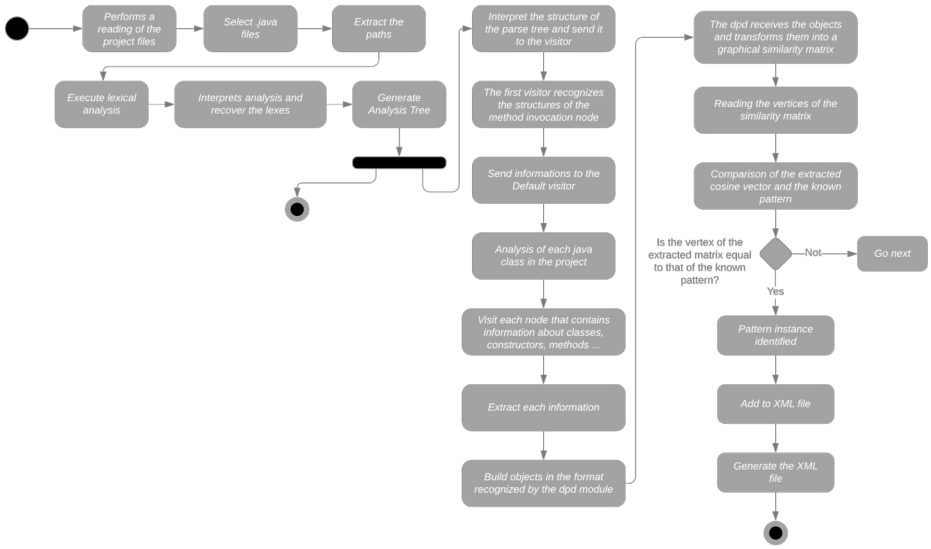
Fonte: Elaborado pelo autor, 2021

Para tal fim, dois *visitors* foram implementados com base no processo de

parser, conseguindo assim acesso às informações do processo de análise. O *visitor* percorre as árvores de análise chamando explicitamente o método de visita da interface ***ParseTreeVisitor*** nos nós filhos.

Assim dessa forma, um *visitor* consegue ter acesso aos nós visitantes da **AST**, podendo, assim, em tempo de execução do *visitor* obter informações extraídas da árvore que são necessárias para a criação dos objetos de modelo que posteriormente serão informados a **dpd-tool** para execução do segundo módulo. Uma primeira execução no projeto pelo primeiro *visitor* em busca de informações estruturais de métodos como sua invocação, tipo de retorno e demais contextos, para entregar ao segundo *visitor*, informações que não se conseguiria no método de visita para recuperar estas informações em tempo de visita do nó. Com as informações presentes, cria-se objetos modelos, em que essas informações estão presentes, o formato no qual a **dpd-tool** consegue reconhecer sua estrutura e aplicar o algoritmo de similaridade identificando então uma possível instância do padrão, A Figura 3 demonstra o diagrama de atividades da ferramenta *antlr-dpd*.

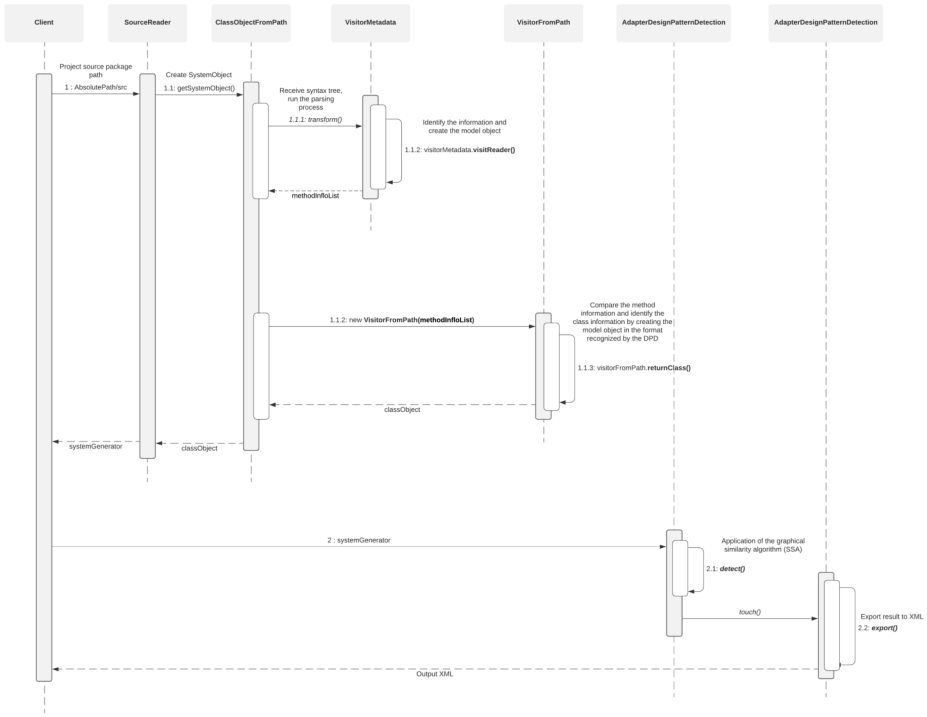
Figura 3 – Diagrama de atividade da ferramenta *Antlr-Dpd*



Fonte: Elaborado pelo autor, 2021

A ponto de validar a nossa ferramenta, utilizou-se de oráculos derivados da dpd conseguindo garantir a corretude da nossa ferramenta por meio de informações derivadas dos arquivos de *benchmarking*, como os *outputs* XML, extraídos das execuções do estudo de caso. Para um melhor entendimento do fluxo, a Figura 4 apresenta um diagrama de sequência.

Figura 4 – Diagrama sequência do módulo da *Antlr-Dpd*



Fonte: Elaborado pelo autor, 2021

A anrlt-dpd tem como *output* um arquivo em XML contendo todos os padrões detectados no código-fonte. Assim como na DPD, para cada padrão, são listadas todas as suas instâncias e apresentadas as principais classes e métodos como pode ser visualizado na Figura 5.

Figura 5 – Representação do output XML da *AntIr-Dpd*

```
▼<system>
 ▼<pattern name="Factory Method">
  ▼<instance>
     <role name="Creator" element="FactoryCar"/>
     <role name="FactoryMethod()" element="FactoryCar::criar(String):Carro"/>
   </instance>
  </pattern>
  <pattern name="Prototype"/>
  <pattern name="Singleton"/>
  <pattern name="(Object)Adapter"/>
  <pattern name="Command"/>
  <pattern name="Composite"/>
  <pattern name="Decorator"/>
  <pattern name="Observer"/>
  <pattern name="State"/>
  <pattern name="Strategy"/>
  <pattern name="Bridge"/>
 ▼<pattern name="Template Method">
  ▼<instance>
     <role name="AbstractClass" element="GameMaker"/>
     <role name="TemplateMethod()" element="GameMaker::create():void"/>
   </instance>
  </pattern>
  <pattern name="Visitor"/>
  <pattern name="Proxy"/>
  <pattern name="Proxy2"/>
  <pattern name="Chain of Responsibility"/>
 </system>
```

Fonte: Elaborado pelo autor, 2021

3.2 MÓDULO DPD

Design Pattern Detection Tool, trabalha com o **SSA (*Similarity Scoring Approach*)** de forma que usa um algoritmo de similaridade gráfica para detectar padrões de projeto, calculando a similaridade de vértices entre o padrão e o sistema em estudo, a partir do bytecode das classes, representando através de uma matriz de similaridade as associações entre as classes. Abordagem do *score* de similaridade que mede a distancia do cosseno de similaridade a partir das vértices da matriz conhecida pela dpd e da matriz gerada com *input* das classes do projeto. Definindo uma possível instancia do padrão ou não. Para lidar com o problema de tamanho do sistema, o **SSA** divide o sistema em vários subsistemas e o algoritmo de similaridade é aplicado ao subsistemas em vez de todo o sistema. A DPD apresenta seus resultados em uma interface gráfica e tem como output um arquivo em XML contendo todos os padrões detectados no código-fonte. Para cada padrão, são listadas todas as suas instâncias e apresentadas as principais classes e métodos como pode ser visualizado na Figura 6

Figura 6 – Representação do output XML da *Design Pattern Detection*

```
<system>
  <pattern name="Factory Method">
    <instance>
      <role name="Creator" element="com.ifpb.model.FactoryCar"/>
      <role name="FactoryMethod()" element="com.ifpb.model.FactoryCar::criar(java.lang.String):com.ifpb.interfaces.Carro"/>
    </instance>
  </pattern>
  <pattern name="Prototype"/>
  <pattern name="Singleton"/>
  <pattern name="(Object)Adapter"/>
  <pattern name="Command"/>
  <pattern name="Composite"/>
  <pattern name="Decorator"/>
  <pattern name="Observer"/>
  <pattern name="State"/>
  <pattern name="Strategy"/>
  <pattern name="Bridge"/>
  <pattern name="Template Method">
    <instance>
      <role name="AbstractClass" element="com.ifpb.game.GameMaker"/>
      <role name="TemplateMethod()" element="com.ifpb.game.GameMaker::create():void"/>
    </instance>
  </pattern>
  <pattern name="Visitor"/>
  <pattern name="Proxy"/>
  <pattern name="Proxy2"/>
  <pattern name="Chain of Responsibility"/>
</system>
```

Fonte: Elaborado pelo autor, 2021

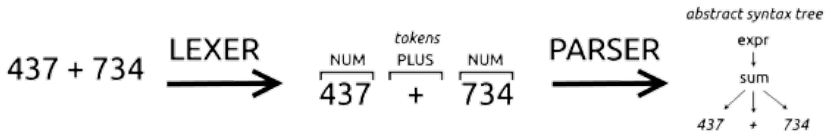
3.3 MÓDULO ANTLR

ANTLR é um gerador de analisadores, uma ferramenta que ajuda a criar analisadores. Um analisador pega um pedaço de texto e o transforma em uma estrutura organizada, o modelo de saída é uma árvore de análise em vez de uma ***Abstract Syntax Tree* (AST)**.

Pode-se pensar na AST como uma história que descreve o conteúdo do código, ou também como sua representação lógica, criada pela junção de várias partes. A diferença é que uma **árvore de análise** é exatamente o que sai do analisador, enquanto o AST é uma versão mais refinada da árvore de análise. Dessa forma é criado o AST manipulando a árvore de análise para obter as informações de todos os nós do projeto que está sendo análisado.

Inicialmente se define uma gramática *lexer* e *parser*, a antlr irá gerar um *lexer* e um analisador em sua linguagem de destino (por exemplo, Java, Python, C , JavaScript), um *lexer* pega os caracteres individuais e os transforma em *tokens*, os átomos que o analisador usa para criar a estrutura lógica. Após a invocação, se passa o código para reconhecer e é retornada uma árvore de análise, como visto na Figura 7.

Figura 7 – Representação do módulo da Antlr4



Fonte: Elaborado pelo autor, 2021

3.4 DIFICULDADES ENCONTRADAS

Durante todo o decorrer do desenvolvimento da ferramenta, uma enorme gama de dificuldades foram se expondo pelo caminho. Tais dificuldades acabaram por interferir bastante no resultado final deste experimento. Nesta seção, iremos abordar algumas destas dificuldades

3.4.1 Limitações do módulo da DPD

A escolha da implementação inicial para detecção de um padrão ***Toy***, casos reais fictícios com código fonte mais limpo e um numero de classes menor, mais fáceis de se trabalhar. De forma programática, escolhendo e adicionando suas informações como necessário, já que somente foi obtido o código fonte da dpd, uma ferramenta que tem um nível de complexabilidade extremamente grande, para antes de qualquer tipo de identificação foi necessário semanas de estudo de código fonte e a necessidade da implementação de um *factory method*, para que fosse possível a identificação do padrão *Toy*. Verificando dessa forma se o *insight* do trabalho teria fundamento para ser abordado.

3.4.2 Limitações da antlr4

A antlr é realmente duas coisas: uma ferramenta que traduz sua gramática para um analisador / *lexer* em Java (ou outra linguagem de destino). Então os analisadores consistem em um conjunto de regra ou "nós", então um aplicativo Java por exemplo chama uma dessas regras que são funções contendo informações da regra específica combinada.

3.4.2.1 Transformar a *Syntax Tree* em uma *Abstranct Syntax Tree*

Para pequenas tarefas de análise, às vezes é conveniente usar **antlr4** no modo interpretado, em vez de gerar um analisador em um destino específico, compilá-lo e executá-lo como parte de seu projeto. Criando objetos com a *Grammar lexer* e *parser* e, em seguida, cria-se interpretadores. Uma vez que temos um *ParserInterpreter*, pode-se

usá-lo para analisar a partir de qualquer regra que necessitar, dado um índice de regra (que a gramática + o analisador podem fornecer).

3.4.2.2 Importações

A limitação dentre os nós caía sobre uma serie de informações necessárias a identificação, porem inalcançáveis. Ao tentar obter as informações de importações dos pacotes, em determinado momento fez-se necessário comparar a nomenclatura de classes para adicionar suas informações de importação, porém, a **antlr4** não trazia identificadores para as importações gerais de pacote representadas por *****(asterisco) assim, impossibilitando adicionar importações de classes que estivessem nestes pacotes. Como consequências as informações de nome das classes ficaram de forma simples como : **Trator** e não de forma canonica com o nome do pacote como : **com.ifpb.package.Trator** ou **com.ifpb.package.***.

3.4.2.3 Informações de invocações de métodos

Devido ao fato desta área de desenvolvimento ser relativamente desconhecida, é compreensível que os desenvolvedores ainda estejam se adaptando as melhores maneiras de encapsular este tipo de implementação. Algumas informações não podiam ser recuperadas em tempo de execução do *visitor*. O **nó terminal** responsável por trazer as invocações de métodos não recuperava nativamente o seu **tipo de retorno**, então para a montagem do nosso objeto de modelo ficava faltando informações extremamente necessárias para a identificação.

4 AVALIAÇÃO

Nesta seção será apresentado um estudo de caso que foi utilizado como avaliação ao problema proposto na seção 1.3, contendo uma descrição, além de detalhar sobre as ferramentas em que o ambiente do estudo será realizado.

4.1 INSTRUMENTAÇÃO

Para o desenvolvimento da pesquisa acerca do estudo de caso, que será descrito no próximo item, foram utilizados os seguintes instrumentos: o Linux Ubuntu 20.04 LTS como sistema operacional; JDK com a versão 1.8.0. Além disso, foi utilizada a IDE IntelliJ IDEA em sua versão 2019.2 para o desenvolvimento da ferramenta, já para a execução da análise estática sobre o código e a extração de informações sobre as estruturas de classe foi utilizada a ferramenta **ANTLR4** com a gramática na versão ***grammars4***, que está disponível tanto no GitHub, quanto no Site[1] da Ferramenta.

Atualmente o trabalho conta com um protótipo exploratório o qual foi projetado para testar algumas principais premissas sobre o projeto; funcionalidade, tecnologia ou ambas. Na abordagem proposta, a extração das estruturas de classe foi feita por meio da **ANTLR4** de forma funcional. O reconhecimento dos padrões de projeto a partir do **.jar**, está sendo realizado com a ferramenta **DPD** utilizando da análise estática para verificação, sem a necessidade dos projetos compilados.

No estudo de caso a ferramenta utilizada para detecção dos padrões de projeto em projetos open source foi a **DPD**[2], com uma abordagem que se baseia no escore de similaridade (SSA) como visto na seção 2.5.1. O SSA usa um algoritmo de similaridade gráfica para detectar padrões de projeto calculando a similaridade de vértices entre o padrão e o sistema em estudo (TSANTALIS et al., 2006).

Para que seja possível a detecção em várias linguagens de programação é necessário reconhecer os vocabulários de software como visto na seção 2.1. Existem algumas formas de se detectar os termos gramaticais da linguagem escolhida, um deles e a **grammars4**[3], a partir de um arquivo de gramática da linguagem com a extensão **.g4** que contém todos os termos do vocabulário de software.

[1] https://www.antlr.org/
[2] https://users.encs.concordia.ca/ nikolaos/pattern$_detection.html$
[3] https://github.com/antlr/grammars-v4

4.2 PLANEJAMENTO DO EXPERIMENTO

Na busca de verificar a validade dos métodos e conceitos utilizados e pressupostos durante o desenvolvimento da abordagem, é essencial obter resultados confiáveis e consistentes. Neste intuito, faz-se necessário um planejamento bem definido para os experimentos que suprem com dados, a pesquisa realizada. Foi utilizado então o modelo GQM (*Goal Question Metrics*) (WOHLIN et al., 2012). Um modelo GQM é uma estrutura hierárquica, começando com um objetivo, ou seja, especificando a finalidade da medição, o objeto a ser medido. O objetivo é então refinado em várias perguntas, que tentam caracterizar o objeto a ser mensurado. Cada pergunta é então refinada em métricas, algumas delas objetivas, algumas delas subjetivas.

É uma abordagem interessante, pois em algumas situações, é difícil visualizar os objetivos estão sendo alcançadas ou não, e as métricas auxiliam nisso. É importante ter algum tipo de critério de aceitação correspondente a esses objetivos. Esse critério de aceitação é definido na forma de perguntas na abordagem do GQM, e essas perguntas são respondidas usando as métricas.

4.2.1 Problema

- ***Business Problem***: É possível avaliar a qualidade dos padrões de projeto presente em projetos *open-source* com base nos padrões extraídos?
- ***Technical Problem***: Avaliar a qualidade dos padrões de projeto em diferentes linguagens de programação.

4.2.2 Objetivos

1. Analisar as estruturas de classes em busca de padrões de projeto
2. Com a intenção de avaliar em diferentes linguagens
3. Com respeito a quantidade de padrões de projeto utilizados
4. Do ponto de vista do desenvolvedor.
5. No contexto dos projetos *open-source* que utilizam os padrões de projeto.

4.2.3 Formulação das Hipóteses

Inicialmente, foi planejada uma pergunta sobre a questão principal (P1) e perguntas secundárias (P2, P3 e P4) foram definidas com intuito de viabilizar questões de contribuições a este trabalho.

P1: É possível a detecção em múltiplas linguagens?

Inicialmente foi verificado se existem instâncias de padrões de projeto, analisando os projetos *open-source*

H1: A metodologia para implementação na linguagem Java pode ser aplicada em todas as linguagens conhecidas pela **antlr4**.

Em relação a categorização dos padrões é necessário avaliar se a categorização imposta pelo **GoF**, tem alguma relação com a quantidade de padrões identificados.

P2: A quantidade de padrões utilizados para validar a pesquisa ?

H1: Foi verificado a cerca de 20 padrões dentro do catálogo de **Padrões GoF *('Gang of Four')***.

É importante analisar como a detecção de padrões de projeto como se comportam de acordo com o conjunto de métricas utilizados. Será levado em consideração a quantidade de indícios de padrões detectados utilizando o catalogo detectado pela DPD.

4.2.4 Avaliação em Três Etapas

Especialmente quando se pretende usar a avaliação para alguma abordagem, é preciso refletir desde a definição dos objetivos até a elaboração de instrumentos e critérios de análise do resultados. Para este trabalho foi definido em três etapas :

1. Processo manual
2. Caso real fictício (Projetos Toy)
3. Projetos Acadêmicos

Na primeira etapa da avaliação, o processo manual de identificação de um (*factory-toy*)foi feito no código fonte da DPD de forma programaticamente analisando e adicionando manualmente as informações referentes as classes do projeto em teste. Na segunda etapa do processo de avaliação, foi executado na **antlr-dpd** projetos *Toy*, casos reais fictícios porem mais simples de se trabalhar. Com um número de classes menor, código fonte mais limpo. Na terceira e ultima etapa, foram executados os projetos acadêmicos *open-source*.

4.2.5 Seleção dos Participantes

Para a realização deste experimento, foi utilizado um conjunto de projetos *open source* e um conjunto de métricas. Foram selecionados dezessete projetos e três métricas. As Tabelas 2 e 3 mostram, respectivamente, as métricas e os projetos selecionados.

Tabela 2 – Métricas a nível de classe selecionadas para as relações com os padrões.

Metricas	Siglas
Number of Classes	NOC
Number of Line Codes	LOC
Number of Java Files	NOJF

Fonte: Elaborado pelo autor, 2021

Os projetos escolhidos seguiram os seguintes critérios: (i) possuir o código fonte, (ii) a versão compilada de forma organizada, não quebrar a execução. Para as métricas, o Qualitas.class fornece um total de 23 sobre todos os seus projetos. Desse conjunto, as métricas foram selecionadas com base no seguinte critério: Métricas que remetem a características encontradas em classes, como seus atributos e métodos.

4.3 RESULTADO DA ANÁLISE

Após aplicar a abordagem em todos os projetos, foram constatados indícios e casos reais de padrões de projeto. A Tabela 3 mostra os valores agrupados por projeto. Optou-se por utilizar a precisão da métrica como indicador de qualidade da métrica desenvolvida.

Tabela 3 – Projetos *open-source* selecionados do Repositório do IFPB-Cajazeiras.

Projeto	#classes	#arquivos	#loc	#github.com
exemplo-abstract-factory	11	11	100	/ads-ifpb-padroes/exemplo-abstract-factory
exemplo-builder	5	4	116	/ads-ifpb-padroes/exemplo-builder
exemplo-prototype-gof	3	3	43	/ads-ifpb-padroes/exemplo-prototype-gof
exemplo-singleton	3	3	100	/ads-ifpb-padroes/exemplo-singleton
exemplo-bridge	7	7	91	/ads-ifpb-padroes/exemplo-bridge
exemplo-composite	10	10	119	/ads-ifpb-padroes/exemplo-composite
exemplo-decorator	7	7	100	/ads-ifpb-padroes/exemplo-decorator
exemplo-facade	5	5	87	/ads-ifpb-padroes/exemplo-facade
exemplo-flyweight	12	12	134	/ads-ifpb-padroes/exemplo-flyweight
exemplo-chain-of-responsability	5	4	99	/ads-ifpb-padroes/exemplo-chain-of-responsability
exemplo-command	10	10	160	/ads-ifpb-padroes/exemplo-command
exemplo-template-method	3	3	120	/ads-ifpb-padroes/exemplo-template-method
exemplo-state	4	4	135	/ads-ifpb-padroes/exemplo-state
exemplo-observer	5	5	99	/ads-ifpb-padroes/exemplo-observer

Fonte: Elaborado pelo autor, 2021

A Tabela 4 demonstra valores agrupados por projetos **Toy** criado e utilizado.

Tabela 4 – Projetos *TOY* manipulados para a abordagem.

Projeto	#classes	#arquivos	LOC
Factory Method - Toy	8	8	150
Template Method - Toy	3	3	52

Fonte: Elaborado pelo autor, 2021

Para validar a ferramenta proposta e observar se o processo de detecção ocorre corretamente, foi decidido a mensuração do *f-score (precision e recall)*. Para tanto, foi executado a seguinte metodologia: (i) detectou-se as instâncias dos padrões com a DPD; (ii) detectou-se as instâncias dos padrões com a antlr-dpd; (iii) mediu-se as métricas de *precision e recall*, observando os dados da **DPD** como fidedignos às instâncias existentes. Como pode-se observar na Tabela 5, para todos os padrões detectados e identificados, as instâncias detectadas entre a DPD e a antlr-dpd são as mesmas. Casos em que a DPD, não identificasse alguma instancia seriam por um de três motivos: (i) O padrão não estaria na lista de padrões de projeto conhecidos pela DPD, (ii) Não há nenhuma instancia daquele padrão, no projeto em si ou (iii) Falha na ferramenta. Dessa forma, se alguma dessas condições fosse satisfeita a a ferramenta não detectaria nenhuma instancia, se aplicando também para antlr-dpd, além de outros motivos especificados em **"Dificuldades encontradas"**.

Tabela 5 – Representação em métricas do *f-score*.

padrões	precision	recall	fscore
Abstract Factory	0	0	0%
Brigde	0	0	0%
Builder	0	0	0%
Chain Of Responsability	1	0	0%
Command	0	0	0%
Composite	0	0	0%
Decorator	0	0	0%
Facade	0	0	0%
Factory Method	1	1	100%
Flyweight	0	0	0%
Observer	0	0	0%
Prototype - GOF	0	0	0%
Singleton	1	1	100%
State	0	0	0%
Template Method	1	1	100%

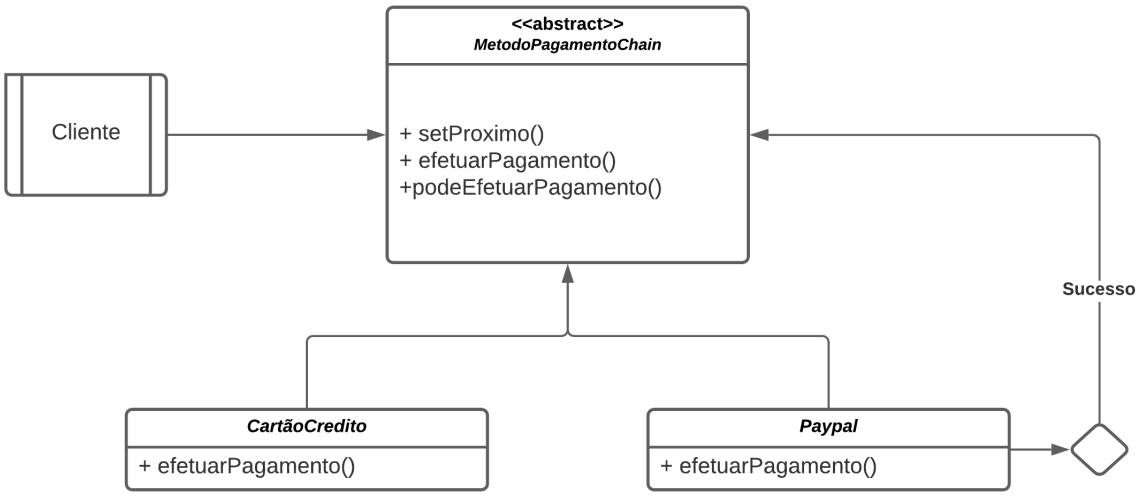
Fonte: Elaborado pelo autor, 2021

Durante a execução do estudo de caso buscando validar a antlr-dpd, foi notado que algumas instâncias de padrões de projeto não eram identificadas nem detectados pela ferramenta; Instancias, tais como *Builder*, *Facede* e *Flyweight* que não foram identificados por não estarem presentes na lista de padrões conhecidos pela DPD.

Casos como *State* e *Observer* a antlr-dpd não identificou nenhuma instancia de padrão presente, quanto na DPD foi identificado e detectado. A diferença entre as duas ferramentas no caso apresentado é que a DPD está em um estado de implementação muito maduro onde a gama de informações que é possível detectar é maior do que as obtidas pela antlr-dpd.

Na terceira etapa desse estudo de caso **Execução de projetos acadêmicos**, foi executado um projeto com uma instancia de padrão determinada. O projeto contia um ***Chain of Responsability*** padrão da categoria **Comportamental** representado pelo diagrama de classe na Figura8.

Figura 8 – Diagrama de Classe do Padrão Divergente



Fonte: Elaborado pelo autor, 2021

Esse padrão durante a execução da DPD foi indentificado e detectado um ***Chain of Responsability*** e um ***Template Method*** presentes no projeto. Enquanto na execução da antlr-dpd foi identificado um ***Factory Method*** e um ***Template Method***. Com isso podemos determinar que alguma propriedade que faz do ***Chain of Responsability*** uma instancia real, não estava presente da identificação da antlr-dpd.

Além da instancia de um **falso padrão** *(Factory Method)*, foi identificado um falso positivo que não estava listado como presente no projeto *(Template Method)*, porém presente nas duas execuções das ferramentas.

5 CONSIDERAÇÕES FINAIS

No decorrer deste trabalho foi apresentado uma contextualização a respeito de qualidade de software, suas aplicações e boas práticas para o desenvolvimento, assim como se relacionam com o uso de **padrões de projeto**. Também foi demonstrada o problema técnico e conceitual a respeito a como avaliar a abordagem proposta. Com isso foi então levantada a pergunta de pesquisa deste trabalho.

Logo após, foi apresentada a fundamentação teórica deste trabalho, na qual foi dissertado a respeito de conceitos e teorias necessárias para a total compreensão deste trabalho, como detecção de padrões de projeto e a relação com vocabulários de software, além do conceito empírico sobre a utilização de projetos *open source* para ser feito a detecção.

Então foi descrita como foi realizada a metodologia deste trabalho, apresentando o uso da revisão bibliográfica de maneira exploratória para o levantamento da problemática deste trabalho além de apresentar a abordagem criada como metodologia a ser utilizada na realização do experimento que está sendo desenvolvido.

Por fim foi apresentada a proposta para a solução da pergunta de pesquisa levantada na Problemática, que se dá pela utilização da ferramenta de reconhecimento de idiomas **ANTLR4**, como ferramenta para o desenvolvimento e aplicação do processo de reconhecimento de linguagem, e a ferramenta **DPD** como ferramenta para o desenvolvimento da análise a partir dos projetos não compilados.

Foi desenvolvido o protótipo de uma ferramenta, que realiza a análise estática do código fonte e recupera informações sobre as estruturas, assim como as devidas chamadas de métodos feitas.

Os resultados deste trabalho se iniciaram com os fichamentos produzidos após a leitura de artigos e textos relacionados ao tema abordado, este foi o levantamento bibliográfico, onde o foco estava em analisar como outros trabalhos utilizaram a análise estática em temas que envolviam identificação e detecção de padrões de projeto, buscando analisar as técnicas, ferramentas utilizadas, para que melhor guiasse o desenvolvimento deste trabalho. Após o desenvolvimento da ferramenta prevista e a análise dos resultados foi possível notar alguns pontos.

Na avaliação, foram encontrados sete instancias reais em projetos acadêmicos de padrões de projeto de um total de quatorze projetos analisados, criando assim uma

implementação possível para ser seguida em múltiplas linguagens. Isso caracteriza o cumprimento do objetivo proposto pelo trabalho.

Como principais contruibuições podemos citar: (i) o protótipo da ferramenta antlr-dpd, que realiza a análise estática do código e extrai todas as informações sobre os projetos executados a partir dos das classes **.java** sem necessitar da compilação de um projeto para execução da ferramenta.

Com a intenção de avaliar em múltiplas linguagens, o protótipo exploratório desenvolvido traz todo conhecimento necessário para aplicação em múltiplas linguagens, foi criado um protótipo exploratório na linguagem **C++** não finalizado, que conta com o reconhecimento de um projeto a partir do seu caminho de entrada, ate o processo de *parsing* da árvore de análise.

Por fim, como trabalhos futuros podemos desenvolver a ferramenta, uma vez que a ferramenta ainda apresenta algumas falhas em performance, dado alguns teste realizados com projetos de grande porte como *Junit5*, *Common Collections* do Quallitas.class, e até mesmo em alguns padrões já identificados e detectados pela dpd-tool como *State* e *Observer*. Além da implementação em outras linguagens a partir do *visitor* de cada uma delas.

REFERÊNCIAS

ALUR, D.; MALKS, D.; CRUPI, J.; BOOCH, G.; FOWLER, M. **Core J2EE Patterns (Core Design Series): Best Practices and Design Strategies**. [S.l.]: Sun Microsystems, Inc., 2003.

AMPATZOGLOU, A.; KYRIAKI, S.; STAMELOS, I.; CHARALAMPIDOU, S. An empirical study on design pattern usage on open-source software. In: **ENASE**. [S.l.: s.n.], 2010. p. 275–284.

ARNOUT, K.; MEYER, B. Pattern componentization: the factory example. **Innovations in Systems and Software Engineering**, Springer, v. 2, n. 2, p. 65–79, 2006.

BENNETT, K. H.; RAJLICH, V. T. Software maintenance and evolution: a roadmap. In: **Proceedings of the Conference on the Future of Software Engineering**. [S.l.: s.n.], 2000. p. 73–87.

BUSE, R. P.; WEIMER, W. R. Learning a metric for code readability. **IEEE Transactions on Software Engineering**, IEEE, v. 36, n. 4, p. 546–558, 2009.

FONTANA, F. A.; CARACCIOLO, A.; ZANONI, M. Dpb: A benchmark for design pattern detection tools. In: . [S.l.: s.n.], 2012. p. 235–244. ISBN 9781467309844.

GAMMA, E.; HELM, R.; JOHNSON, R.; VLISSIDES, J. Design patterns: Elements of reusable object-oriented software addison-wesley. **Reading, MA**, v. 2, p. 369–378, 1995.

GUÉHÉNEUC, Y.-G.; ANTONIOL, G. Demima: A multilayered approach for design pattern identification. **IEEE transactions on software engineering**, IEEE, v. 34, n. 5, p. 667–684, 2008.

GUO, Y.; WÜRSCH, M.; GIGER, E.; GALL, H. C. An empirical validation of the benefits of adhering to the law of demeter. In: IEEE. **2011 18th Working Conference on Reverse Engineering**. [S.l.], 2011. p. 239–243.

Owens, D.; Anderson, M. A generic framework for automated quality assurance of software models - application of an abstract syntax tree. In: **2013 Science and Information Conference**. [S.l.: s.n.], 2013. p. 207–211. ISSN null.

PRESSMAN, R. S. **Software engineering: a practitioner's approach**. [S.l.]: Palgrave macmillan, 2005.

SANTOS, K. d. F.; GUERRERO, D. D.; FIGUEIREDO, J. C. de. Using developers contributions on software vocabularies to identify experts. In: IEEE. **2015 12th International Conference on Information Technology-New Generations**. [S.l.], 2015. p. 451–456.

SCHMIDT, D. C.; STAL, M.; ROHNERT, H.; BUSCHMANN, F. **Pattern-Oriented Software Architecture, Patterns for Concurrent and Networked Objects**. [S.l.]: John Wiley & Sons, 2013. v. 2.

SEVERO, N. d. S.; JOB, R. d. S. An approach to detect false design patterns. In: **Proceedings of the XIII Brazilian Symposium on Software Components, Architectures, and Reuse**. [S.l.: s.n.], 2019. p. 63–72.

TSANTALIS, N.; CHATZIGEORGIOU, A.; STEPHANIDES, G.; HALKIDIS, S. T. Design pattern detection using similarity scoring. **IEEE transactions on software engineering**, IEEE, v. 32, n. 11, p. 896–909, 2006.

VOKÁC, M. An efficient tool for recovering design patterns from c++ code. **Journal of Object Technology**, v. 5, n. 1, p. 139–157, 2006.

WOHLIN, C.; RUNESON, P.; HÖST, M.; OHLSSON, M. C.; REGNELL, B.; WESSLÉN, A. **Experimentation in software engineering**. [S.l.]: Springer Science & Business Media, 2012.

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

Documento Digitalizado Restrito

Entrega de trabalho de conclusão de curso

Assunto: Entrega de trabalho de conclusão de curso
Assinado por: Raul Cabral
Tipo do Documento: Anexo
Situação: Finalizado
Nível de Acesso: Restrito
Hipótese Legal: Informação Pessoal (Art. 31 da Lei no 12.527/2011)
Tipo do Conferência: Cópia Simples

Documento assinado eletronicamente por:

- **Raul Coêlho Cabral**, **ALUNO (201622010256) DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS - CAJAZEIRAS**, em 28/06/2021 08:09:03.

Este documento foi armazenado no SUAP em 28/06/2021. 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: 262934
Código de Autenticação: 24fa3322ab