Ferramenta para qualificar a Complexidade Cognitiva do código utilizando a técnica Cognitive Driven Development

Ane Cristine Crispim
Instituto Federal Catarinense Campus Rio do Sul
SC - Brasil
anecrispim2@gmail.com

Marcela Leite
Instituto Federal Catarinense Campus Rio do Sul
SC - Brasil
marcela.leite@ifc.edu.br

ABSTRACT

This work proposed the development of an extension for Visual Studio Code, which supports the PHP programming language, using the Cognitive Driven Development (CDD) method to identify the cognitive complexity of the source code. The extension helps the developer to locate code snippets with unnecessary complexity, encouraging the application of refactoring techniques and good code design practices. For this purpose, research related to CDD was analyzed and the necessary parameters for the creation of a configuration file in JSON were defined. Due to the limitation of the VS Code API, which does not provide the AST (Abstract Syntax Tree) of the code, it was decided to integrate the Php-parser library, written in JavaScript, which proved to be compatible with the Typescript development environment. The tool was validated through tests on a real project, comparing its performance before and after refactoring. In the initial analysis, the extension correctly identified the critical points that needed improvement. After refactoring, a reduction in the identified complexity was observed, evidencing the effectiveness of the tool, even when applied to a small project. In summary, the extension proves to be an effective solution for automating the identification of cognitive complexity, contributing to more efficient software development and continuous maintenance of source code.

CCS CONCEPTS

• Software notations and tools • Development frameworks and environments • Design languages

KEYWORDS

Algorithm, Cognitive Complexity, Extension, VS Code, CDD.

1 Introdução

As definições de complexidade de software são diferentes dependendo do ponto de vista e elas podem ser divididas em três partes: a complexidade de implementação, complexidade de interface e complexidade de tamanho *[1]*. A complexidade de implementação é aquela que o programador possui ao ler um código e entendê-lo. A complexidade de interface é definida pela dificuldade que o cliente tem em usar um sistema, geralmente através da sua interface. Já a complexidade de tamanho é definida como o tamanho que um software possui em quantidade de linhas do código fonte.

As definições e princípios de um Código Limpo [2] citam a importância de desenvolver algoritmos que não gerem confusão àqueles que futuramente darão manutenção e continuidade ao código. Portanto, a preocupação com a complexidade de implementação que um Código terá ao final do seu desenvolvimento deve ficar evidente ao programador, para que assim, ele consiga decidir se essa complexidade é aceitável ou não considerando que terceiros venham a acessar e dar manutenção aquele código no futuro.

Com base nisso, o Cognitive Driven Development (Desenvolvimento orientado a cognição), abreviado como CDD, surgiu com o objetivo de contribuir como uma técnica de design de código que foca em reduzir sua complexidade, para isso o método utiliza formas de quantificar a complexidade do código escrito e definir limites para essa complexidade, tudo isso de uma forma parametrizável *[3]*.

O CDD lida com a complexidade cognitiva de um código, enquanto outras medidas de complexidade de algoritmos como, por exemplo, a Complexidade Ciclomática preocupa-se com a redução de casos necessários para cobrir um método, a complexidade cognitiva preocupa-se com a compreensibilidade do código e essa diferença se dá pelo fato de que a complexidade Ciclomática de um método não representa necessariamente a dificuldade de quem irá manter aquele algoritmo *[4]*.

Dentro deste contexto, este trabalho busca automatizar a detecção da complexidade cognitiva de um código-fonte e aproximar o programador da técnica de desenvolvimento orientado a cognição (CDD), para isso foi desenvolvida uma extensão para a IDE Visual Studio Code, que analisa a Abstract Syntax Tree (AST) de arquivos escritos em PHP por meio da biblioteca Php-parser, identificando pontos críticos que podem ser aprimorados. A validação em um projeto prático demonstrou a capacidade da ferramenta em apontar áreas que poderiam receber uma refatoração, contribuindo significativamente para a melhoria da qualidade e manutenção do código-fonte.

2 Cognitive Driven Development

O Cognitive Driven Development é uma técnica que busca reduzir a complexidade de código, adicionando limites no número de construções de programação. De acordo com [3] o CDD “em essência, ele se baseia na contagem das ocorrências de itens de complexidade em uma determinada unidade de código”.

O CDD baseia-se em teorias cognitivas da psicologia, sendo elas a teoria do “Número Mágico Sete” e a “Teoria de Carga Cognitiva (CLT) [3].

Ao estipular um limite dentro do CDD, os desenvolvedores devem sinalizar pontos de complexidade para determinados contextos dentro do código, como por exemplo um condicional, a ele será atribuído um valor de complexidade que deve ser incrementado na unidade de código final (arquivo com o código-fonte). Portanto, ao extrapolar o limite estipulado de complexidade para uma unidade de código, é preciso que o desenvolvedor faça uma refatoração, reanalisando a forma com que o código foi escrito, a fim de adaptá-lo de maneira a se adequar aos limites de complexidade estabelecidos.

Há também a determinação dos elementos que os desenvolvedores consideram mais complexos dentro de um código, podendo haver discordância naquilo que um considera mais complexo em relação a outro, entretanto a equipe de desenvolvimento deve elencar uma lista com alguns itens do código-fonte que em concordância tornam o código mais complexo.

Com base nisso são calculados os Pontos de complexidade intrínseca (do inglês Intrinsic Complexity Points ICPs) que de acordo com [3] “ICPs são elementos de código que podem afetar o entendimento dos desenvolvedores de acordo com sua frequência de uso”. Exemplos de itens a serem considerados são condicionais, laços de repetições, classes, métodos, funções, herança, manipuladores de exceção, etc.

O CDD é a técnica que a ferramenta proposta por este trabalho busca automatizar, portanto os conceitos do CDD tornam-se importantes no entendimento de como a extensão proposta precisará atuar no processo de identificação da complexidade de um código.

3 Solução proposta

Este trabalho propõe o desenvolvimento de uma extensão para a IDE Visual Studio Code, que será uma ferramenta de auxílio ao programador em identificar a complexidade de um código escrito na linguagem de programação PHP.

A complexidade do código será identificada pela ferramenta através da técnica CDD, uma técnica que baseia-se no desenvolvimento orientado à cognição, onde são consideradas as limitações cognitivas do ser humano para avaliação da complexidade de determinadas situações.

A extensão desenvolvida recebe um arquivo configurável para estipulação de parâmetros, esses parâmetros consistem nas limitações da complexidade que um arquivo de código fonte deve ter, bem como a definição dos ICPs para os elementos do código, como um bloco de “if”, de laço de repetição, um método, uma classe, etc. O arquivo em questão é apresentado na Figura 1.


```
{
  "totalFileComplexity": {
    "description": "Complexidade cognitiva total máxima permitida para um arquivo PHP.",
    "maxComplexity": 50,
    "indices": {
      "controlStructureComplexity": {
        "description": "Complexidade para estruturas de controle.",
        "weights": {
          "if": 1,
          "elseif": 1,
          "for": 1,
          "while": 1,
          "switch": 1,
          "foreach": 1
        }
      },
      "tryCatchComplexity": {
        "description": "Complexidade para blocos try-catch.",
        "weights": {
          "try": 1,
          "catch": 1
        }
      },
      "inheritanceComplexity": {
        "description": "Complexidade para herança de classes.",
        "weights": {
          "class": 1,
          "interface": 1
        }
      },
      "functionComplexity": {
        "description": "Complexidade para funções.",
        "weights": {
          "function": 1,
          "parameters": 0.5,
          "return": 0.5
        }
      },
      "methodComplexity": {
        "description": "Complexidade para métodos de classes.",
        "weights": {
          "method": 1,
          "parameters": 0.5,
          "return": 0.5
        }
      },
      "expressionComplexity": {
        "description": "Complexidade associada a expressões em variáveis.",
        "weights": {
          "ternary": 1,
          "logicalOperators": 0.5,
          "arithmeticOperators": 0.5
        }
      }
    }
  }
}
```


Figura 1: Arquivo de configuração da ferramenta elaborado pelos autores

O arquivo de configuração deve seguir o padrão estipulado na Figura 1 e a ferramenta é capaz de calcular a complexidade somente dos elementos definidos no arquivo padrão, caso o usuário opte por não calcular a complexidade para um elemento específico definido no arquivo, basta definir seu peso como zero.

A ferramenta deve ler o código escrito em um arquivo pelo programador, em blocos delimitados para a linguagem PHP e quando o autor do algoritmo executar a ferramenta em determinado arquivo, então ela identifica os índices de complexidade para cada bloco de código considerando a estrutura dos paradigmas de programação da linguagem. Desta forma, através dos parâmetros definidos como limites de complexidade, a ferramenta deve indicar no próprio código, em forma de comentário, qual a complexidade daquele algoritmo marcando em vermelho aqueles que extrapolaram o limite de complexidade.

Vale ressaltar que a ferramenta não irá tratar a complexidade do código, mas sim apenas mensurar a complexidade, pois justifica-se o desenvolvimento de uma extensão de IDE para que o desenvolvedor consiga utilizar a ferramenta em conjunto com demais recursos que o auxilie em melhorar o código escrito.

Os resultados que a ferramenta apresentou são validados analisando um projeto de autoria própria que possui 7 amostras de códigos-fonte. As amostras de algoritmos têm padrões diferentes de quantidade de linhas de código e estruturação do algoritmo. Em um comparativo, a ferramenta será aplicada nesse mesmo projeto após uma refatoração, a fim de validar o comportamento da ferramenta e a qualidade de sua identificação de complexidade.

Na refatoração do projeto foram aplicadas técnicas de Design de código, como Design Pattern, Código Limpo e uma nova arquitetura do projeto, nesse caso foi usado a arquitetura MVC, modularizando melhor as funcionalidades do projeto, mas mantendo todas as suas funcionalidades originais.

4 Resultados

No primeiro projeto, sem refatoração, a ferramenta foi aplicada em 7 arquivos. Já no projeto com refatoração, a ferramenta foi aplicada em 8 arquivos. Nestes contextos, a extensão foi capaz de identificar que de 7 arquivos do projeto sem refatoração, 5 tiveram o limite de complexidade ultrapassado. E no projeto com refatoração, de 8 arquivos, 3 tiveram o limite de complexidade ultrapassado.

Mesmo que no projeto com refatoração ainda existam arquivos com o limite de complexidade atingido, ainda assim é possível notar uma redução de complexidade geral do projeto, onde a ferramenta identificou menor complexidade no projeto refatorado do que no projeto sem refatoração. Na Tabela 1 abaixo, é apresentada uma relação da complexidade obtida pela ferramenta nos dois projetos.

Tabela 1 - Tabela elaborada pela autora com a complexidade obtida pela ferramenta nos projetos com e sem refatoração

	Projeto Inicial	**Projeto Refatorado**
Número de arquivos	7	8
Soma das Complexidades	121	98
Média das Complexidades	17,28	12,25
Maior Complexidade	27,50	26,50
Menor Complexidade	4	2

Na tabela 1, são apresentados o número de arquivos em cada arquivo, a soma de complexidade de todos os arquivos obtida em cada projeto, a média dessas complexidades, a maior e menor pontuação de complexidade obtida em um arquivo dos projetos e ao lado a diferença desses valores entre os dois projetos. Onde é possível perceber que o projeto refatorado teve menos complexidade em relação ao projeto inicial.

5 Considerações finais

O CDD é uma técnica que surge com o propósito de auxiliar o desenvolvedor a diminuir a complexidade cognitiva do código, delimitando parâmetros para quantificar a complexidade das estruturas de uma unidade de código fonte, por esse motivo a técnica se torna um método interessante para ajudar o programador a escrever códigos menos complexos.

Ao aplicar a ferramenta no projeto refatorado, foi percebido que a ferramenta atendeu a expectativa de diminuição da complexidade após a refatoração do código-fonte. Em relação aos números obtidos, considerando a aplicação da ferramenta no projeto refatorado versus o projeto inicial, a extensão identificou no projeto uma queda na soma total da complexidade de 19,01% e uma queda na média geral da complexidade de 29,13% por arquivo.

É válido considerar que os resultados obtidos não refletem a totalidade do desempenho da ferramenta, uma vez que o projeto em que ela foi aplicada é pequeno e possui características específicas, bem como o tipo de refatoração feita para a comparação de resultados, porém inicialmente a ferramenta demonstrou resultados positivos dentro dos contextos dispostos neste trabalho.

Em conclusão, a ferramenta é capaz de identificar a complexidade de todos os elementos definidos em seu arquivo de configuração. Além disso, a ferramenta leva uma nova perspectiva ao desenvolvedor na análise do seu código, mostrando pontos a serem melhorados quando o limite de complexidade é atingido, dando o indicativo da possibilidade de realizar uma refatoração e aproveitar os benefícios que esse processo traz ao código, como a redução de bugs, melhoria na legibilidade do código, aumento do uso de padrões e aplicação de boas práticas de programação, levando a um código limpo e agilizando o tempo de manutenção futura dedicada a ele.

REFERENCES

[1] RAYMOND, E. S. The art of unix programming. System, 2003.

[2] MARTIN, Robert C. Código Limpo: Habilidades Práticas d o Agile Software. Rio de Janeiro: Alta Books, 2011.

[3] PINTO, G.; DE SOUZA, A. Cognitive Driven Development helps software teams to keep code units under the limit! Journal of Systems and Software, v. 206, 2023.

[4] CAMPBELL, G. A. Cognitive Complexity. A new way of measuring understandability. SonarSource S.A., v. 1.7 Whitepaper, 2023.