Métricas de McCabe (Complejidad Ciclomática)

13 de Junio del 2024

1. Métricas de McCabe (Complejidad Ciclomática)

1.1. Definición y Tipos

La complejidad ciclomatica es una métrica introducida por Thomas J. McCabe en 1976 para medir la complejidad lógica de un programa. Esta métrica cuantifica el número de caminos linealmente independientes a través del código fuente de un programa, proporcionando una medida objetiva de la complejidad del software.

1.1.1. Tipos

Existen varias formas de calcular la complejidad ciclomatica:

- **Fórmula de Aristas y Nodos:** $V(G) = E - N + 2$, donde E es el número de aristas y N es el número de nodos en el grafo de flujo del programa (Scalabrino & Oliveto, 2019).
- **Fórmula de Nodos Predicados:** $V(G) = P + 1$, donde P es el número de nodos predicados, es decir, nodos que contienen condiciones (**liu2018evaluate**).
- **Fórmula de Regiones del Grafo:** $V(G) = R$, donde R es el número de regiones en el grafo de control de flujo (Odeh et al., 2023).

1.2. Aplicaciones y Limitaciones

1.2.1. Aplicaciones

La complejidad ciclomatica se utiliza principalmente para:

- **Planificación de Pruebas:** Determinar el número mínimo de casos de prueba necesarios para cubrir todos los caminos posibles a través del código (Peitek, 2020).
- **Evaluación de la Calidad del Código:** Proporciona una medida objetiva de la complejidad del código, facilitando la identificación de módulos complejos (Scalabrino & Oliveto, 2019).
- **Mantenimiento y Refactorización:** Ayuda a identificar partes del código que son excesivamente complejas y que podrían beneficiarse de la refactorización (Monteiro, 2019).

Ejemplo de Código en Python para Calcular la Complejidad Ciclomática

Listing 1: Calculando la Complejidad Ciclomática en Python

```
import ast

class CyclomaticComplexityVisitor(ast.NodeVisitor):
    def __init__(self):
        self.complexity = 1  # Start with a base complexity of 1

    def visit_If(self, node):
        self.complexity += 1
        self.generic_visit(node)

    def visit_For(self, node):
        self.complexity += 1
        self.generic_visit(node)

    def visit_While(self, node):
        self.complexity += 1
        self.generic_visit(node)

    def visit_With(self, node):
        self.complexity += 1
        self.generic_visit(node)

    def visit_ExceptHandler(self, node):
        self.complexity += 1
        self.generic_visit(node)

    def visit_BoolOp(self, node):
        self.complexity += 1
        self.generic_visit(node)

    def visit_And(self, node):
        self.complexity += 1
        self.generic_visit(node)

    def visit_Or(self, node):
        self.complexity += 1
        self.generic_visit(node)

def calculate_cyclomatic_complexity(code):
    tree = ast.parse(code)
    visitor = CyclomaticComplexityVisitor()
    visitor.visit(tree)
    return visitor.complexity

# Example function to analyze
code_example = """
def example_function(a, b):
    if a > b:
        return a - b
    elif a == b:
        return 0
    else:
        return b - a

    for i in range(10):
        print(i)

    while a < b:
        a += 1

    try:
        x = 1 / 0
    except ZeroDivisionError:
        print("Division by zero")
"""

complexity = calculate_cyclomatic_complexity(code_example)
print(f"Cyclomatic␣Complexity:␣{complexity}")
```

1.2.2. Limitaciones

A pesar de sus beneficios, la complejidad ciclomatica también tiene algunas limitaciones:

- **No Considera la Complejidad del Dominio:** La métrica solo evalúa la complejidad del código, sin tener en cuenta la complejidad del problema que el código está resolviendo.
- **Ignora el Rendimiento del Código:** No proporciona información sobre el rendimiento del código, como el tiempo de ejecución o el uso de memoria.
- **Dependencia de la Estructura del Código:** Puede ser engañosa si el código está estructurado de manera ineficiente, con muchas ramas condicionales innecesarias.

2. Evaluación de la Métrica

La evaluación de la métrica de complejidad ciclomatica se realiza principalmente a través del análisis del código fuente para identificar estructuras de control que incrementan la complejidad del software. Aquí se describen los pasos comunes y las herramientas utilizadas para evaluar esta métrica:

2.1. Pasos para Evaluar la Complejidad Ciclomática

1. **Construcción del Grafo de Flujo de Control:**
   - **Identificación de Nodos y Aristas:** Cada instrucción en el código se representa como un nodo y las transiciones entre estas instrucciones (flujo de control) se representan como aristas.
   - **Creación del Grafo:** Se construye un grafo dirigido donde los nodos son las instrucciones y las aristas representan las transiciones de control.
2. **Cálculo de la Complejidad Ciclomática:**
   - **Fórmula Básica:** $V(G) = E - N + 2$, donde E es el número de aristas, N es el número de nodos, y G es el número de componentes conectados del grafo.
   - **Método Alternativo:** También se puede calcular como el número de regiones en el grafo de control de flujo, o como $P + 1$, donde P es el número de nodos predicados.
3. **Interpretación de los Resultados:**
   - **Valores Bajos (1-10):** Indican código de baja complejidad, fácil de entender y mantener.
   - **Valores Moderados (11-20):** Indican código con complejidad moderada, puede requerir atención en términos de pruebas y mantenimiento.
   - **Valores Altos (¿20):** Indican alta complejidad, difícil de entender y mantener, y puede ser propenso a errores.

2.2. Herramientas para Evaluar la Complejidad Ciclomática

- **Visual Studio Code Metrics:** Visual Studio proporciona una funcionalidad integrada para calcular las métricas de código, incluida la complejidad ciclomatica. Esto se puede hacer seleccionando ‘Analyze ¿Calculate Code Metrics for Solution‘ (Studio, 2024).
- **MATLAB Code Analyzer:** MATLAB tiene una herramienta llamada Code Analyzer que puede medir la complejidad ciclomatica utilizando el comando ‘checkcode‘ con la opción ‘cyc« (Simulink, 2024).
- **Omni Calculator:** Omni Calculator ofrece una herramienta en línea para calcular la complejidad ciclomatica ingresando parámetros del código (Calculator, 2024).
- **SonarQube:** SonarQube es una plataforma de análisis de código que incluye la métrica de complejidad ciclomatica entre otras métricas de calidad de código (SonarQube, 2024).

Referencias

Calculator, O. (2024). Cyclomatic Complexity Calculator [https://www.omnicalculator.com/math/cyclomatic-complexity].

Monteiro, E., Sylvain y Sokolovas. (2019). Measuring code maintainability with deep neural networks. *Software Quality Journal*, *27*(4), 1235-1250.

Odeh, A. H., Odeh, M., & Odeh, H. (2023). Measuring Cyclomatic Complexity of Source Code Using Machine Learning. *International Journal of Intelligent Engineering and Systems*, *14*(1), 23-33.

Peitek, N. y. o. (2020). An Empirical Validation of Cognitive Complexity as a Measure of Source Code Understandability. *arXiv preprint arXiv:2007.12520*.

Scalabrino, S., & Oliveto, G., Rocco y Bavota. (2019). From Code Complexity Metrics to Program Comprehension. *Communications of the ACM*, *62*(5), 91-99.

Simulink, M.
bibinitperiod. (2024). Measure Code Complexity Using Cyclomatic Complexity [https://www.mathworks.co
code − complexity − using − cyclomatic − complexity.html].

SonarQube. (2024). SonarQube [https://www.sonarqube.org/].

Studio, V. (2024). Code metrics - Cyclomatic complexity [https://learn.microsoft.com/en-us/visualstudio/code-quality/code-metrics-cyclomatic-complexity?view=vs-2022].