Complejidad Ciclomática y Cognitiva

Edgar Sarmiento Calisaya

Universidad Nacional de San Agustín de Arequipa – Perú

Escuela Profesional de Ciencia de la Computación – EPCC

Esta Presentación hace uso de texto y figuras presentadas en documentos listados en la sección de referencias

Objetivos

- Comprender los conceptos básicos de complejidad ciclomática.
- Comprender los conceptos básicos de complejidad cognitiva.
- Comprender la importancia de complejidad ciclomática en las pruebas de software.
- Comprender la importancia de complejidad cognitiva en la mantenibilidad de software.

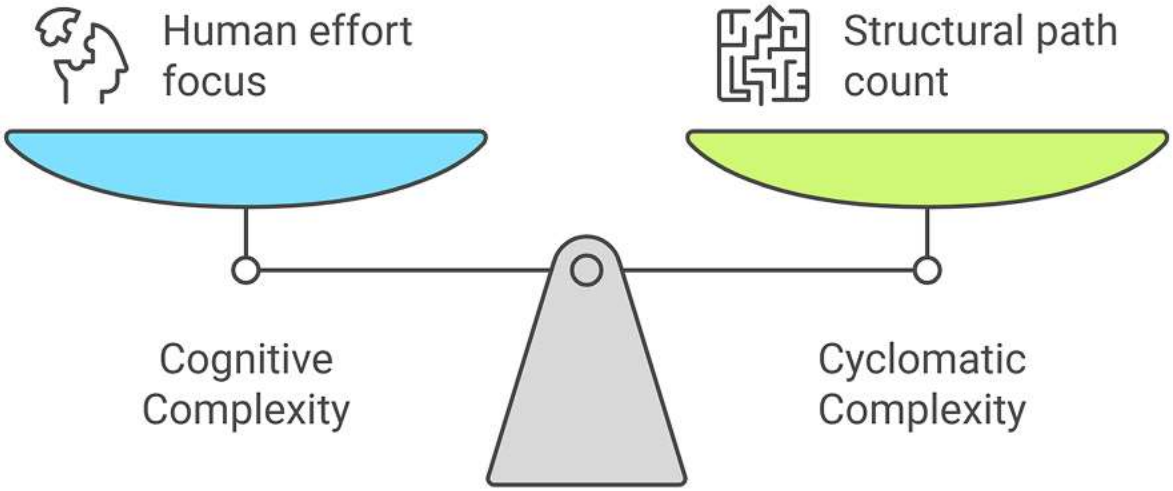
Agenda

Temas:

1. **Motivación**
2. **Complejidad Ciclomática**
3. **Complejidad Cognitiva**
4. **Conclusión**

Motivación

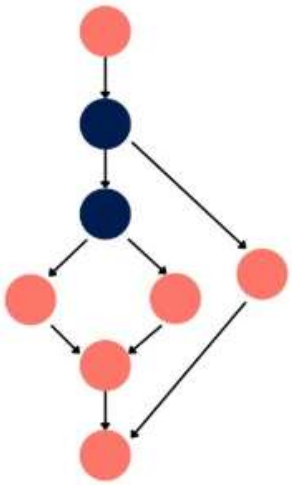
- Las **métricas** de software proporcionan una **medición cuantitativa** del grado en que una propiedad o característica está presente en un software.
- Utilizado generalmente para realizar **comparativas** o planificaciones.
- Una de las métricas más importantes con la que contamos en el desarrollo es la **complejidad**, que puede ser usada en las fases de **desarrollo, mantenimiento** o **pruebas**.



https://mstone.ai/glossary/cognitive-complexity/

Complejidad Ciclomática

- Dentro de la complejidad, en el mundo del software, la más usada, es la llamada complejidad ciclomática (McCabe , 1976).
- Muestra qué tan dificultosa es la lógica de un programa:
  - Qué tan eficaz/eficiente es un algoritmo.
- **Mide** el **número de caminos linealmente independientes** a través del **código**.
  - Valores altos suelen indicar la necesidad de simplificación.
- Se puede **CALCULAR** por:
  - **Análisis estático** de código fuente de un programa – **POR PALABRA CLAVE**
  - Un **grafo** que representa el **diagrama de flujo de control** que viene determinado por la representación de las estructuras de control de un determinado **programa**.



Cyclomatic complexity:

$$V(G)=E-N+2*P$$

Where:

- V(G) = Maximum number of linearly independent paths
- E = Total number of edges
- N = Total number of nodes
- P = Total number of predicate nodes (conditional nodes)

Complejidad Ciclomática

- **Complejidad Ciclomática - Calculo por Palabra Clave:**

  1. El valor comienza en "1" para el **método**, **función** o **rutina** (con o sin retorno)

  2. Agregue un punto más para cada elemento a continuación:

  2.1 Selección: IF, CASE O SWITCH

  2.2 Bucles: FOR, WHILE, DO, BREAK y CONTINUE

  2.3 Operadores: "&&", "||", "?"

  2.4 Excepciones: CATCH, THROW y THROWS

  2.5 Flujo – RETURN distinto del último

  Nota: ELSE, DEFAULT, FINALLY, ":" y el último RETURN no incrementan el conteo.

```
int sumOfPrimes(int max) {                    // +1
  int total = 0;
  OUT: for (int i = 1; i <= max; ++i) {       // +1
    for (int j = 2; j < i; ++j) {             // +1
      if (i % j == 0) {                       // +1
        continue OUT;
      }
    }
    total += i;
  }
  return total;
}                          // Cyclomatic Complexity 4
```

```
String getWords(int number) {    // +1
    switch (number) {
      case 1:                    // +1
        return "one";
      case 2:                    // +1
        return "a couple";
      default:                   // +1
        return "lots";
    }
  }          // Cyclomatic Complexity 4
```

https://docs.sonarsource.com/sonarqube-server/user-guide/code-metrics/metrics-definition

Complejidad Ciclomática

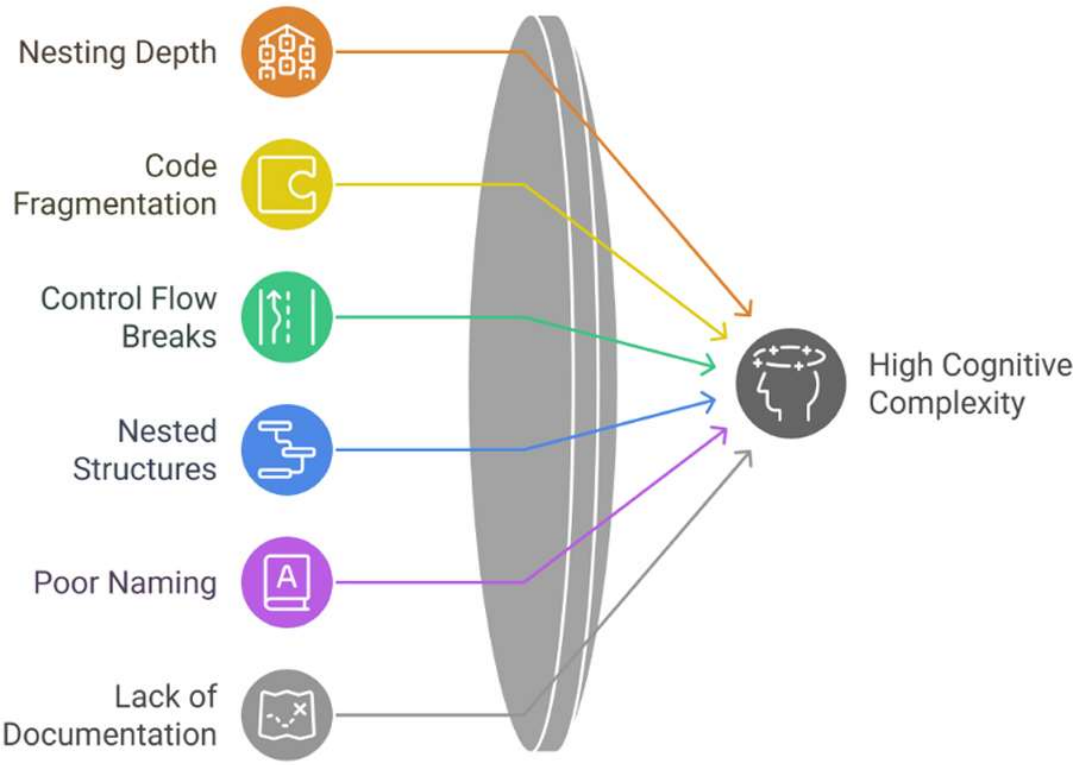
- Ambos códigos tienen un **complejidad ciclomática de 4**.
- Sin embargo, el **primero** de ellos tiene una **mayor dificultad para ser mantenido** debido al número de estructuras anidadas.
- Por lo que el **esfuerzo humano** destinado a su mantenimiento será **mayor que el segundo**.
- **Complejidad Ciclomática** nos **guía** en la implementación de algoritmos más **fáciles de probar (testability)**
  - Utilizado para la generación de casos de prueba.
- **Complejidad Ciclomática** no nos indica una medida de mantenibilidad
  - Esto es la **complejidad cognitiva**.

```
int sumOfPrimes(int max) {                    // +1
  int total = 0;
  OUT: for (int i = 1; i <= max; ++i) {       // +1
    for (int j = 2; j < i; ++j) {             // +1
      if (i % j == 0) {                       // +1
        continue OUT;
      }
    }
    total += i;
  }
  return total;
}                          // Cyclomatic Complexity 4
```

```
String getWords(int number) {    // +1
    switch (number) {
      case 1:                    // +1
        return "one";
      case 2:                    // +1
        return "a couple";
      default:                   // +1
        return "lots";
    }
  }          // Cyclomatic Complexity 4
```

Complejidad Cognitiva

- A diferencia de la **Complejidad Ciclomática**, que determina qué **dificultad tiene probar el código**.
- La **Complejidad Cognitiva** es una **medida** de cómo es de difícil **entender** intuitivamente e **mantener** un bloque de código.
- Una **alta complejidad cognitiva** se **correlaciona** con mayores tasas de errores, deuda técnica y ciclos de desarrollo más lentos.



https://mstone.ai/glossary/cognitive-complexity/

Complejidad Cognitiva

- Si bien ambas **métricas** abordan la calidad del código, tienen **propósitos** distintos:

Aspect	Cognitive Complexity	Cyclomatic Complexity
Focus	Human readability & mental effort	Structural paths & test coverage
Measurement	Penalizes nesting, complex logic	Counts decision points (e.g., if, loops)
Use Case	Prioritize refactoring for clarity	Identify testing requirements

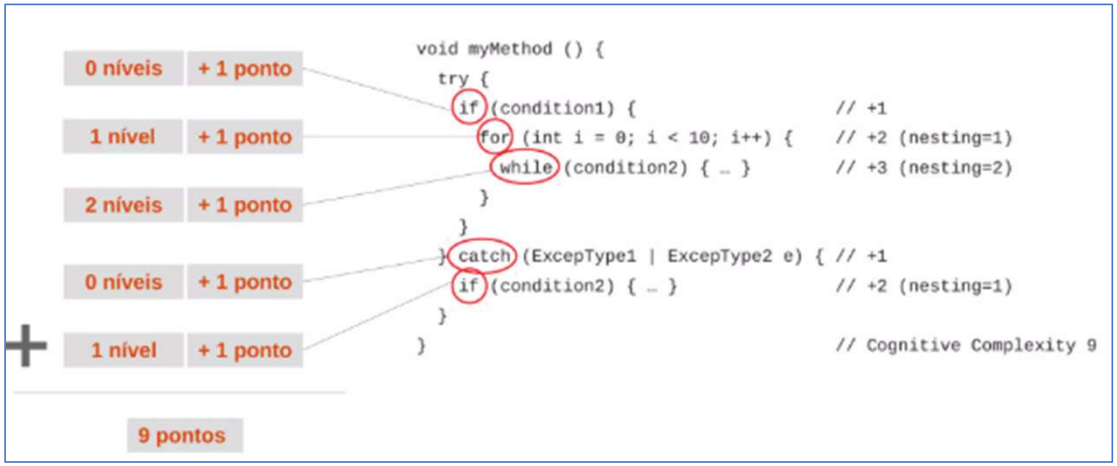
- Un **método** con varios **bucles anidados y condicionales** podría tener una complejidad ciclomática moderada, pero podría ser muy difícil de leer y comprender, lo que le otorga una alta calificación de complejidad cognitiva.
- **Por ejemplo**, una **sentencia switch con 10 casos** obtiene una puntuación de **10 en complejidad ciclomática**, pero puede tener una **baja complejidad cognitiva si la lógica es sencilla**. Por el contrario, un solo método con bucles anidados y condicionales podría tener una complejidad ciclomática moderada, pero una alta complejidad cognitiva debido a dificultades de legibilidad.

Complejidad Cognitiva

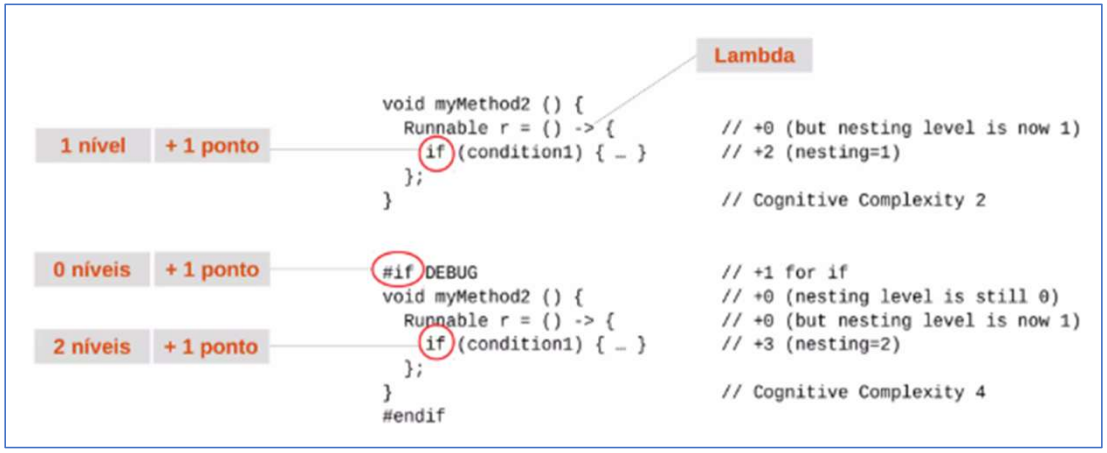
- **Criterios básicos y metodología**
  - Una **puntuación** de Complejidad Cognitiva se evalúa de acuerdo a estas **tres reglas básicas**:
    1. **Ignora** (no incrementa) las estructuras que permiten que varios enunciados sean legibles **abreviados** en **uno**.
       - swich, operador ternario.
    2. **Incrementa** (añade uno) por cada **ruptura en el flujo lineal** del código.
       - **+1**, if, else if, else, operador ternario, switch, for, foreach, while, do while, cacth, goto LABEL, continue LABEL, secuencia de operadores lógicos (&&, ||, ..) y recursividad
    3. **Incrementa** a cada nivel de **anidamiento** de una ruptura de de flujo (además del punto por ruptura – R2).
       - **+1**, if, else if, else, operador ternario, switch, for, foreach, while, do while, cacth, métodos anidados y métodos o estructuras tipo *lambda*.
       - **+1**, , if, operador ternario, switch, for, foreach, while, do while y cacth

https://www.sonarsource.com/docs/CognitiveComplexity.pdf

Complejidad Cognitiva



Estas **reglas** y los **principios** detrás de ellas se detallan más minuciosamente en:
https://www.sonarsource.com/docs/CognitiveComplexity.pdf



Complejidad Cognitiva

- **Criterios básicos y metodología**
  - Además, una **puntuación** de complejidad se compone de **cuatro tipos diferentes de incrementos**:
    - **A. Anidamiento**: evaluado en estructuras de flujo de control dentro de otras.
    - **B. Estructural**: evaluado en estructuras de flujo de control que están sujetas a un incremento por anidamiento (**1 punto + anidamiento**). Causan incremento por anidamiento.
      - If, #if, for, while, …
    - **C. Fundamental**: evaluado sobre declaraciones no sujetas a un incremento por anidamiento (**siempre cuentas 1**).
      - &&, ||, goto LABEL, recursión
    - **D. Híbrido**: evaluado en estructuras de flujo de control que no están sujetas a un incremento por anidamiento (**siempre cuentas 1**). Causan anidamiento.
      - Else, else if, elif, …

https://www.sonarsource.com/docs/CognitiveComplexity.pdf

Complejidad Cognitiva

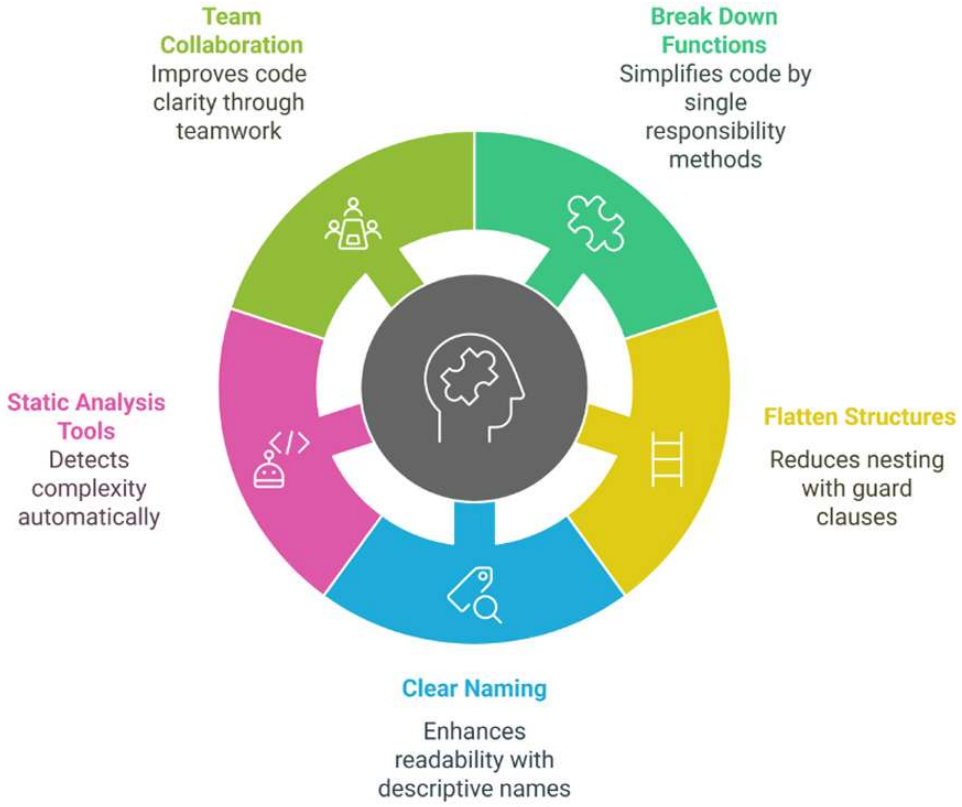
- Ambos códigos tienen un **complejidad ciclomática de 4**.
- Sin embargo, el **primero** de ellos tiene una **mayor dificultad para ser mantenido** debido al número de estructuras anidadas.
- La **complejidad cognitiva** en el caso del **switch** tiene valor 1, ya que es una estructura fácil de comprender y manejar.
- Sin embargo en el **flujo del primer algoritmo**, al tener una serie de elementos anidados hace más difícil la comprensión para el desarrollador, por lo que la **complejidad cognitiva aumenta**

```
                                  // Cyclomatic Complexity    Cognitive Complexity
int sumOfPrimes(int max) {              // +1
  int total = 0;
  OUT: for (int i = 1; i <= max; ++i) { // +1                      +1
    for (int j = 2; j < i; ++j) {       // +1                      +2 (nesting=1)
      if (i % j == 0) {                 // +1                      +3 (nesting=2)
        continue OUT;                   //                         +1
      }
    }
    total += i;
  }
  return total;
}                                 //         =4                    =7
```

```
                                  // Cyclomatic Complexity    Cognitive Complexity
String getWords(int number) {     //          +1
  switch (number) {               //                            +1
    case 1:                       //          +1
      return "one";
    case 2:                       //          +1
      return "a couple";
    default:                      //          +1
      return "lots";
  }
}                                 //          =4                =1
```

Conclusión

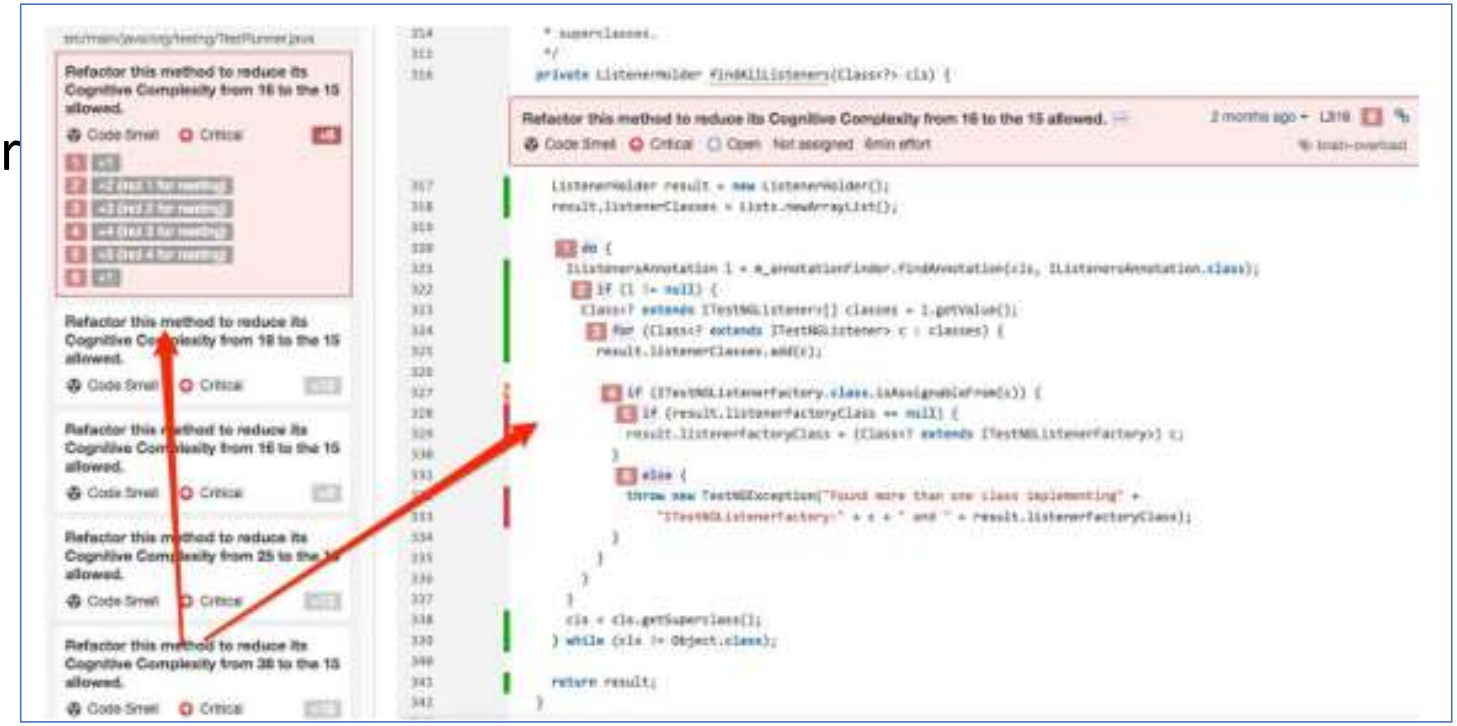
- Gestionar la complejidad del código es esencial para desarrollar software sostenible.
- Al centrarse en la legibilidad mediante **funciones más pequeñas, nombres claros y colaboración**, los equipos pueden reducir la deuda técnica y acelerar el desarrollo.
- Como dice el refrán: «El código se lee más de lo que se escribe».
- Priorizar prácticas de código intuitivas garantiza que su trabajo resista el paso del tiempo.



https://mstone.ai/glossary/cognitive-complexity/

Conclusión

- Herramientas como **SonarQube** priorizan la **complejidad cognitiva** para evaluar la **mantenibilidad**, reservando la **complejidad ciclomática** para el análisis de la **cobertura de pruebas**.



https://www.enmilocalfunciona.io/complejidad-cognitiva/

Referencias

- MCCABE, T. J. (1976). **A Complexity Measure**. IEEE Transactions on Software Engineering, SE-2(4), 308-320.
- CAMPBELL, A. **Cognitive complexity: A new way of measuring understandability**. SonarSource (2016).
- FOWLER, M. (2018). Refactoring: Improving the Design of Existing Code. Addison-Wesley.
- MARTIN, R. C. (2009). Clean Code: A Handbook of Agile Software Craftsmanship. Prentice Hall.
- **Complejidad Ciclomática**:
  - https://www.ndepend.com/docs/code-metrics#CC
  - https://docs.sonarcloud.io/digging-deeper/metric-definitions/
  - https://ieeexplore.ieee.org/document/1702388
  - https://artesoftware.com.br/2019/02/09/complexidade-ciclomatica/
  - **https://mstone.ai/glossary/code-complexity/**
- **Complejidad Cognitiva**:
  - **Ingles: https://www.sonarsource.com/docs/CognitiveComplexity.pdf**
  - Portugues: https://pt.slideshare.net/DouglasSiviotti/complexidade-cognitiva
  - Español: https://www.excentia.es/complejidad-cognitiva
  - https://www.enmilocalfunciona.io/complejidad-cognitiva/
  - **https://mstone.ai/glossary/cognitive-complexity/**