Question: Exploring the Cyclomatic Complexity's Relevance Today
The Cyclomatic Complexity is commonly considered in modules on testing the validity of code design today. However, in your opinion, should it be? Does it remain relevant today? Specific to the focus of this module, is it relevant in our quest to develop secure software? Justify all opinions which support your argument and share your responses with your team.

The **software testing** activity is very **expensive, critical and complex**, and a software without testing is most dangerous and leads to more expensiveness. Software testing is, therefore, an **essential activity** before the release of any software (Madhavi, 2016; Syaikhuddin et al, 2018). To **evaluate the quality of the program**, it is important to **measure the complexity of the code** itself. Introduced by Thomos J. McCabe (1976), "Cyclomatic complexity" (CC) is a software **quality metric** that provides a **quantitative measure** of the **logical complexity** of a program (Madhavi, 2016; Sarwar et al, 2013) by measuring the number of linearly independent paths through a piece of code (Ebert & Cain, 2016). **Rejected by many academics** for its theoretical weaknesses, it is, however **used extensively by industry** professionals (Ebert & Cain, 2016).

The advantages of Calculating Cyclomatic Complexity are (Madhavi, 2016):

1) The Cyclomatic number is equivalent to the complexity.
2) Helps the developers and testers to find the independent paths through the program.
3) All the independent paths must have been tested at least once.
4) Improves the code coverage.
5) Evaluates and eliminates the more risk associated with the program.
6) Helps to focus on uncovered paths
7) Helps to achieve number of test cases to achieve both branch coverage (Upper bound) and number of paths (Lower bound). (Madhavi, 2016)

However, there are a few limitations of CC/Path Testing:

A time-consuming process, successful Path Testing does not reveal any missing functionality and wrong implementations. It can also become a difficult process if the project is too complex. Furthermore, Madhavi (2016) lists the following limitations:

1) Difficult to find errors in different areas and Interface errors and errors are not caught
2) Not all the initialization mistakes are covered.
3) Specification mistakes in requirements are not covered.
4) The data base and data flow errors are not caught
5) Mistakes in e Control Flow Graph (CFG) are not detected.

Another problem in cyclomatic complexity is the nesting problem (Sarwar et al, 2013).

References :

Ebert, C. & Cain, J. (2016). "Cyclomatic Complexity—40 Years Later", *Cyclomatic Complexity. IEEE Software, 33(6), 27–29.* doi:10.1109/MS.2016.147

Madhavi, D. (2016) A White Box Testing Technique in Software Testing: Basis Path Testing. Journal for Research | Volume 02 | Issue 04

Mohamed et al. (2013) "The Use of Cyclomatic Complexity Metrics in Programming Performance's Assessment", 6th International Conference on University Learning and Teaching, Procedia - Social and Behavioral Sciences 90 (2013) 497 – 503.

Sarwar et al (2013) Cyclomatic Complexity: The Nesting Problem DOI: 10.1109/ICDIM.2013.669398

Syaikhuddin et al (2018) Conventional Software Testing Using White Box Method, KINETIK, Vol. 3, No. 1, February 2018, Pp. 65-72