

# Software Metrics & Software Metrology

---

Alain Abran

## **Chapter 6** **Cyclomatic Complexity Number:** **Analysis of its Design**

# Part 2 of this book

- Part 2 of this book presents some of the most popular ‘software metrics’ and illustrates some of their weaknesses in their designs:
  - Cyclomatic complexity number in Chapter 6
  - Halstead’s metrics in Chapter 7
  - Function Points in Chapter 8
  - Usecase Point in Chapter 9
  - ISO 9126 in Chapter 10

# Agenda

This chapter covers:

- Introduction to the software Cyclomatic Complexity
- The definitions of the cyclomatic number in graph theory
- The transposition of the cyclomatic number to software and related measurement design issues
- Other measurement design issues related in particular to the entity and the attribute being measured

# Agenda

This chapter covers:



- **Introduction to the software Cyclomatic Complexity**
- The definitions of the cyclomatic number in graph theory
- The transposition of the cyclomatic number to software and related measurement design issues
- Other measurement design issues related in particular to the entity and the attribute being measured

# Introduction


- The software Cyclomatic Complexity number was proposed by [McCabe 1976].
  - It has since been referred to extensively in academia, as well as in industry, where it is often included in software tools that automatically collect measures from lines of code.
- While the rules for calculating the Cyclomatic Complexity number are simple, in that they can be automated, a clear definition (characterization and meta-model) of the complexity attribute has not yet been provided.
  - Without such a definition, the measurement results are difficult to understand, and analyzing McCabe's interpretation of software complexity is a challenge.
  - Not provided either are the properties of the numerical representation derived from the application of the algorithm of the cyclomatic number.

# Introduction

- This chapter presents an analysis of the measurement foundations of this number, and highlights some measurement design issues, such as:
  - measurement units
  - labeling of the cyclomatic number as a complexity concept.

# Agenda

This chapter covers:

- Introduction to the software Cyclomatic Complexity
- Definitions of **the cyclomatic number in graph theory** 
- The transposition of the cyclomatic number to software and related measurement design issues
- Other measurement design issues related in particular to the entity and the attribute being measured

# The Cyclomatic Number in Graph Theory

- McCabe's work is based on his analysis of some measurement concepts in graph theory and on his transposition of these concepts into the domain of software measurement.
- Understanding the cyclomatic number in graph theory is therefore important to understanding the input to its design.



# The Cyclomatic Number in Graph Theory

## ■ Definitions and Meta-model

- In graph theory, the cyclomatic number is defined in the following way: the cyclomatic number  $v(G)$  of a strongly connected directed graph is equal to the maximum number of linearly independent cycles where:
  - $v(G)$  is the symbol (with  $G$  representing a graph)
  - the numerical assignment rule is Equation 1:

$$v(G) = e - n + p \quad (1)$$

In this equation, there are:

- $e$  edges,
- $n$  vertices, and
- $p$  separate components.

# The Cyclomatic Number in Graph Theory

The entity being measured = a '**strongly connected directed graph**', the meta-model and characteristics of which are described through the following set of definitions:

- **Simple Graph**: a (usually finite) set of vertices  $V$  (or nodes) and set of unordered pairs of distinct elements of  $V$ , called edges.

## **A simple graph [Berge 2001]**

A simple graph,  $G$ , could be defined as a pair of sets  $(V, E)$ , where:

- $V$  is a finite non empty set of vertices, and
- $E$  is a set of pairs of vertices, called edges.

It is often represented by  $G = (V, E)$

# The Cyclomatic Number in Graph Theory

- **Chain**: In a simple graph  $G=(X,A)$ , a chain  $c$  is a finite sequence of vertices  $x_0, x_1, \dots, x_m$  such that, for all  $i$  with  $0 \leq i \leq m$ ,  $x_i, x_{i+1}$  is an element of  $A$ .
  - A chain is represented by  $c = [x_0, x_1, \dots, x_m]$ .
  - The length of the path  $c$  is the integer  $m$ , and it is represented by  $l = m(c)$  [Berge 2001].
- **Connected Graph**: A graph  $G$  is connected if, for all  $x$  and for all  $y$  [vertices], there exists a chain connecting  $x$  and  $y$  [Berge 01].
  - A graph that is not connected can be divided into connected components.
- **Cycle graph**: a path that begins and ends with the same vertex.

# The Cyclomatic Number in Graph Theory

- **Simple Cycle**: a cycle that has a length of at least 3, and in which:
  - the beginning vertex only appears once more, as the ending vertex, and
  - the other vertices appear only once.
- **Directed Graph**: (also called a digraph or quiver) a graph consisting of:
  - a set  $V$  of vertices,
  - a set  $E$  of edges, and
  - maps  $E \rightarrow V : e \rightarrow (s(e), t(e))$ , where:
    - $s(e)$  is the source and
    - $t(e)$  is the target of the directed edge  $e$ .
- **Strongly Connected Graph**: a directed graph that has a path from each vertex to every other vertex.

# The Cyclomatic Number in Graph Theory

## ■ Units of measurement

- A fundamental concept in measurement is the requirement for units of measurement.
- In looking at the units of measurement in Equation 1 of the cyclomatic number, we find the following:
  1. On the left-hand side of the equation:
    - the units of  $v(G)$  can be derived from the definition itself, that is, **a cycle**:
      - The characterization of this concept can be derived from the definition itself, that is, ‘linearly independent cycles in a strongly connected graph.’
      - A numerical assignment rule is also given in the definition, that is, the **‘maximum number’ of linearly independent paths**.

# The Cyclomatic Number in Graph Theory

## 2. On the right-hand side of the equation:

- There are 3 distinct types of units, that is:
  - edges,
  - vertices, and
  - 'separate components'.
- A numerical assignment rule is given for these numbers next, in the addition-subtraction format; however:
  - The addition or subtraction of different types of units is explicitly invalid in a numerical sense
    - (for instance, it does not make sense to add apples and oranges).
  - These operations are valid only if the units are transposed into a higher level of abstraction with another type of more generic attribute and with the same corresponding unit:
    - In graph theory, it is considered that cycles are being measured when counting nodes and edges.

# The Cyclomatic Number in Graph Theory


## **Example: Adding apples & oranges**

- 3 fruits (of the apple type) can be added to 2 fruits (of the orange type ) to give a total of 5 pieces of fruit.
- In this example, the term 'pieces of fruit' is a generalization of the apple type and the orange type.
- When the unit of the total number is 'pieces of fruit', information at the lower level of abstraction is lost, that is:
  - the term '5 pieces of fruit' does not provide information about how many apples and oranges are included in the total.

The way in which units are manipulated in graph theory has not been clearly documented, and this lack of clarity can make it a challenge to transpose these concepts into other fields.

# Agenda

This chapter covers:

- Introduction to the software Cyclomatic Complexity
- The definitions of the cyclomatic number in graph theory
- The **transposition of the cyclomatic number to software** and related measurement design issues 
- Other measurement design issues related in particular to the entity and the attribute being measured



# The Cyclomatic Number for Software

## ■ Transposition into software

- A key contribution of McCabe has been his transposition of the cyclomatic number from graph theory into software.
- In software, the program is modeled as a *control flow graph*, which is an abstract structure used in compilers:
  - specifically, an abstract representation of a procedure or program, maintained internally by a compiler:
    - Each vertex in the graph represents a basic block.
    - Directed edges are used to represent jumps in the control flow.
- There are 2 specially designated blocks:
  - the entry block: through which control flow enters the flow graph, and
  - the exit block: through which all control flow leaves.

# The Cyclomatic Number for Software

- *Program control flow graphs are not strongly connected, but they become so when a virtual edge is added, connecting the exit node to the entry node [Watson 1996].*
- Since the control flow graph in the case of software is then transformed into a strongly connected graph, the graph cyclomatic number can be applied to this representation of programs.

# The Cyclomatic Number for Software

- So, the cyclomatic number, when applied to software in this transposition, becomes:

$$v(G) = e - n + p + 1 \quad \text{Virtual edge} \quad (2)$$

- Note: 1 has been added to Equation 1 to integrate the supplementary virtual edge.
- Furthermore, in this transposition, only individual modules are taken into account, instead of the whole software entity. In the particular situation defined by McCabe, the number of connected components  $p$  is always equal to 1. McCabe then defines Equation 2 as Equation 3:

$$v(G) = e - n + 2 \quad (3)$$

# The Cyclomatic Number for Software

## Identification of measurement units

- When analyzed taking into consideration the units of each related element being added or subtracted, the software Cyclomatic Complexity number in Equation 3 is puzzling:
  - the number 2 in Equation 3 is derived from the addition of different units:
    - one 'connected component', plus
    - one 'virtual edge'.
  - Properly rewriting Equation 2 taking the units into consideration would lead to Equation 4 instead of Equation 3, that is:

$$v(G)^{\text{independent cycle}} = e^{\text{edges}} - n^{\text{nodes}} + p^{\text{connected components}} + 1^{\text{virtual edge}} \quad (2)$$

becomes  $v(G)^{\text{independent cycle}} = (e + 1)^{\text{edges + virtual edge}} - n^{\text{nodes}} + p^{\text{connected components}}$  (4)

Of course, adequate interpretation of the units in Equation 4 remains an issue as well.

# The Cyclomatic Number for Software

- **The attribute measured – & its association with complexity**
  - McCabe has suggested that Equation 3 is, by association, a measure of ‘program complexity’, which he calls Cyclomatic Complexity and interprets as *the amount of decision logic in a single software module* [Watson 1996].
    - In other words, it is inferred that a software module is represented by a control flow graph and its *cyclomatic complexity is defined for each module to be  $e - n + 2$ , where  $e$  and  $n$  are the number of edges and nodes in the control flow graph respectively* [Watson 1996].
  - Note: no definition is provided for the term ‘complexity’, for the attribute itself, or for its direct characterization.

# The Cyclomatic Number for Software

## ■ Scale type(s)?

- Typically, a measurement method involves a single scale type. However, Zuse [1997] has identified 3 concurrent scale types for the software Cyclomatic Complexity number:
  - ordinal,
  - ratio, and
  - absolute!
- Zuse [1997] suggests, for example:
  - A specified operation of flowgraph concatenation in order to prove that the software Cyclomatic Complexity number is on the ratio scale, which is necessary in working with the additive operation.
  - Examples of other flowgraph operations which lead to ordinal or absolute scales are provided in Zuse [1997].

# The Cyclomatic Number for Software

- As a result, averages and standard deviations on the Cyclomatic number must be carefully computed:
  - We must ensure that the numerical addition corresponds to the empirical operations proposed by Zuse.
    - In other words, does the numerical addition means the same as the proposed empirical operation?
  
- This work of [Zuse 1997] is interesting, since it looks into the mathematical foundations for the analysis of scale types for the Cyclomatic Complexity number.
  - However, such an analysis does not take into account the presence of different measurement units in such equations.

# The Cyclomatic Number for Software

- The challenge remains for practitioners to consider the concurrent possibilities of multiple scale types and their correct use in mathematical operations:
  - identification,
  - interpretation, and
  - related uses in practice.
  
- There is also a long-running debate in the software measurement literature on the composition of graphs and corresponding numerical addition of cyclomatic complexity
  - see Henderson-Sellers & Tegarden [1994a and 1994b].



# The Cyclomatic Number for Software

## ■ ***Interpretation of the Cyclomatic Complexity number for software***

- When only the left-hand side of Equations 2 or 3 is taken into account, the cyclomatic number is defined in terms of the unit ‘number of cycles’, which clearly qualifies as a ‘count’:
  - this operation produces a numerical number of the *integer* type, the numbers of which are clearly on a *ratio* scale:
    - for instance: 10 independent cycles is clearly 5 times 2 independent cycles.

# The Cyclomatic Number for Software

- Now, when the term 'complexity' is added to label this measure, one would expect users of the number derived from the above equation to interpret it on a ratio scale;
  - They do not, but seem instead to *interpret* this number on some kind unspecified interval scale, which could, for example, be of the *exponential* scale type for testing time.

## Interpretation of the values of the software Cyclomatic Complexity number

Given two programs,  $P_a$  and  $P_b$ , with cyclomatic numbers of 10 and 5 respectively, we are inclined to interpret the testing time  $P_a$  as being 'much greater than twice' the testing time of  $P_b$ .

A possible explanation is an intuitive individual perception of the complexity of the graphical representation of the control flow graphs presented in conjunction with the McCabe Cyclomatic Complexity number:


- while the graph of a cyclomatic number of 5 seems easy to represent visually,
- the graph of a cyclomatic number of just double that (10) seems much harder to represent visually, and
- the graph of a cyclomatic number of 20 seems at times spaghetti-like visually!

# The Cyclomatic Number for Software

- It appears, therefore, that, using both the control flow graphs and the software Cyclomatic Complexity number, practitioners build their own interpretation scale and perhaps do so on a different scale type.
  - Of course, this is done implicitly, without clear rules and without measurement-related conventions.
  - This might explained partially why this cyclomatic is largely computed.

# Agenda

This chapter covers:

- Introduction to the software Cyclomatic Complexity
  - The definitions of the cyclomatic number in graph theory
  - The transposition of the cyclomatic number to software and related measurement design issues
  - **Other measurement design issues** related in particular to the **entity and the attribute being measured**
- 

# Other Design Issues : The Entity and Attribute Measured

## ■ The entity measured

- The entity measured by the Cyclomatic Complexity number is a control flow graph.
  - *Control flow graphs describe the logic structure of software modules. A module corresponds to a single function or subroutine in typical languages [Watson 1996].*
- The measured entity is the source code of a given module, which corresponds to a function or a subroutine.
  - So, the meta-model is made up of all the elements present in a graph, i.e. vertices and edges.
    - *the nodes represent computational statements or expressions, and the edges represent transfer of control between nodes [Watson 1996].*

## Other Design Issues : The Entity and Attribute Measured

- But, do graphs correctly represent the source code entity in order to measure its cyclomatic number?
  - In other words, has the assumption concerning the one-to-one relation of a given module's source code to its corresponding graph been verified?
- One module's source code is related to 1, and only 1, graph.
  - But, the reverse is not necessarily true; that is, 1 graph can be related to 1 or many source codes.
  - In [Zuse 1997], this set of source codes is considered as an *equivalence class* of source codes. So, it is not obvious that the final source code corresponds to the measured graph.
- In that sense, it could be interesting to test this assumption, which argues that there is a one-to-one relation between a measured graph and a final source code.

## *Other Design Issues : The Entity and Attribute Measured*

- Another point discussed in [Watson 1996] is the addition of a 'virtual edge' to the control flow graph in order to obtain a strongly connected graph.
  - The rationale of that 'work around' is to allow the use of the cyclomatic number of a strongly connected graph.
  
- But, adding a virtual edge does not modify the nature of the entity considered, i.e. the source program.
  
- The justification for the addition of a virtual edge is as follows:  
*This virtual edge is not just a numerical convenience. Intuitively, it represents the control flow through the rest of the programming in which the module is used [Watson 1996].*
  - So, it can be seen as the connection between the module and the rest of the program.

# Other Design Issues : The Entity and Attribute Measured

## ■ **The attribute measured**

- Referring to the flow graph of a program by its Cyclomatic Complexity number obviates the necessity to explicitly define the complexity of the source code, a definition which remains difficult to formulate and about which there is a lack of agreement in the software community
- By adding the term 'complexity' to the term 'cyclomatic number', an association is made between this number and a concept of complexity, but :
  - not explicitly described &
  - not associated with quantitative numbers for complexity !
- Moreover, no conditions are described under which it would hold for particular values of complexity.



# *Other Design Issues : The Entity and Attribute Measured*

- This lack of clarity has led practitioners, and researchers alike, to make associations such as:
  - *Complexity can be used directly to allocate testing effort by leveraging the connection between complexity and error to concentrate testing effort on the most error-prone software [Watson 1996].*
- with few proven quantitative properties, such as a generalizations like:
  - “The higher the Cyclomatic Complexity number, the greater the error.”
    - Of course: ‘the higher the error rate’ and ‘the most error prone’ clearly do not belong on a ratio scale, but on an ordinal scale at best.

# Other Design Issues : The Entity and Attribute Measured

- *[The] cyclomatic complexity measure correlates with errors in software modules [Watson 1996].*
  - Again this statement has led users of this Cyclomatic Complexity number to associate low numbers of errors with a low cyclomatic number.
- However, a coefficient of correlation ( $r$ ) between two given variables X and Y does not measure any causality relation between those variables:
  - A coefficient close to 1 does not mean that one variable implies the other, but simply expresses the fact that the two variables vary in the same direction.
- *Maintainers can keep maintenance changes from degrading the maintainability of software by limiting the cyclomatic complexity number during a modification [Watson 1996].*
  - The same comments as above apply.

# *Other Design Issues : The Entity and Attribute Measured*

- While the calculation rules for the Cyclomatic Complexity number are simple, and can therefore be automated,
  - a clear definition (characterization and meta-model) of the complexity attribute has not been provided.
- This does not facilitate an understanding of this measure and makes it challenging to analyze McCabe's interpretation of software complexity.
- Similarly, while defining a numerical assignment rule, the properties of the numerical representation derived from the application of the algorithm are not provided.

## *Other Design Issues : The Entity and Attribute Measured*

- Lopez and Habra [2005] have also noted the challenge of using the cyclomatic complexity number in Object Oriented
  - in particular the relevance of the cyclomatic complexity threshold for the java programming language.
- In summary, this chapter has presented an analysis of the measurement foundations of the cyclomatic number and highlighted several measurement design issues, such as:
  - measurement units
  - the artificial labeling of the cyclomatic number as a complexity concept.

# Summary

This chapter has covered:

- Introduction to the software Cyclomatic Complexity
- The definitions of the cyclomatic number in graph theory
- The transposition of the cyclomatic number to software and related measurement design issues
- Other measurement design issues related in particular to the entity and the attribute being measured