Using McCabe Method to Compare the Complexity of Object Oriented Languages

Yas Alsultanny,
AArabian Gulf University (AGU),
Manama Bahrain, Manama, Bahrain 26671

Abstract
The absence of a standard regulatory mechanism in terms of quality control/quality assurance with respect to implementation and managing projects, particularly in the industrial sector has lead to an inconsistency among the various software systems.
The methodologies and tools relevant to the entire life cycle, from conceptualization to implementation, the quality assurance of software has to be visualized. Development of software for managing projects is an extremely complex affair.
The McCabe method of software complexity used to compare the complexity of several object oriented languages such as C++, Java and Visual Basic. The binary search algorithm written in three languages; C++, Java, and Visual Basic was used to measure the complexity of the algorithm, the results showed that the complexity of these languages are different, and the results showed that the languages that are easy in programming has more complexity, than that is not easy in programming. This means that the programs written in Visual Basic will be expected to have higher complexity than the same program written in C++ language.
A comparison between languages keywords showed that we can expected the complexity of programming languages with less keywords will be increased with longer access time.

Keywords: Software complexity, MacCabe method, object oriented languages, software programming.

1. Introduction

The concept of objects and instances in computing had its first major breakthrough with the PDP-1 system at MIT which was probably the earliest example of capability based architecture.
Another early *example* was Sketchpad made by Ivan Sutherland in 1963; however, this was an application and not a programming paradigm. Objects as programming entities were introduced in the 1960s in Simula 67, a programming language designed for making simulations. The idea occurred to group the different types of ships into different classes of objects, each class of objects being responsible for defining its *own* data and behavior. Such an approach was a simple extrapolation of concepts earlier used in *analog* programming [1].
Object-oriented programming developed as the dominant programming methodology during the mid-1990s, largely due to the influence of C++. Its dominance was further cemented by the rising popularity of graphical user interfaces, for which object-oriented programming is well-suited [2].
Java has emerged in wide use partially because of its similarity to C and to C++, but perhaps more importantly because of its implementation using a virtual machine that is intended to run code unchanged on many different platforms. This last feature has made it very attractive to larger development shops with heterogeneous environments. Microsoft's .NET initiative has a similar objective and includes/supports several new languages, or variants of older ones[3].

2. Object Oriented Keywords

The programming languages are affected by the keywords used in programming. Table (1) shows an example of comparison between the keywords of C# and Java languages.

Table (1) shows that there are (35) keywords with similar identification in both languages C# and Java. There are (15) keywords having different identification in both languages for example *base* in C# language is represented with *super* in Java language.
We can predicate that one of the reasons of software complexity is the absence of some keywords form the languages. From the programmer side view it is easier to learn the languages with less keywords, but from the software complexity side, it will be expected that the complexity increased and the access time will be longer.

Manuscript received March 5, 2009
Manuscript revised March 20, 2009

Table 1: C# and Java keywords

C# keyword	Java keyword	C# keyword	Java keyword	C# keyword	Java keyword	C# keyword	Java keyword
abstract	abstract	Explicit	N/A	object	N/A	this	This
as	N/A	Extern	native	operator	N/A	throw	Throw
base	Super	Finally	finally	out	N/A	true	True
bool	boolean	Fixed	N/A	override	N/A	try	try
break	break	Float	float	params	N/A	typeof	N/A
byte	N/A	For	for	private	private	unit	N/A
case	case	Foreach	N/A	protected	N/A	ulong	N/A
catch	catch	Get	N/A	public	public	unchecked	N/A
char	char	Goto	goto	readonly	N/A	unsafe	N/A
checked	N/A	If	if	ref	N/A	ushort	N/A
class	class	Implicit	N/A	return	return	using	import
const	const	In	N/A	sbyte	byte	value	N/A
continue	continue	Int	int	sealed	final	virtual	N/A
decimal	N/A	Interface	interface	set	N/A	void	void
default	default	Internal	protected	short	short	volatile	volatile
delegate	N/A	Is	instanceof	sizeof	N/A	while	while
do	do	Lock	synchronized	stackalloc	N/A	:	extends
double	double	Long	long	static	static	:	implements
else	else	namespace	package	string	N/A	N/A	strictfp
enum	N/A	New	new	struct	N/A	N/A	throws
event	N/A	Null	null	switch	switch	N/A	transient

3. Software Complexity

Software measurement is concerned with deriving a numeric value for an attribute of a software product, i.e. a measurement is a mapping from the empirical world to the formal world.
Software metrics have been found to be useful in reducing software maintenance costs by assigning a numeric value to reflect the ease or difficulty with which a program module may be understood. There are hundreds of software complexity measures that have been described and published for example, the most basic complexity measure, the number of lines of code (LOC), simply counts the lines of executable code, data declarations, comments, and so on. While this measure is extremely simple, it has been shown to be very useful and correlates well with the number of errors in programs [4].

4. McCabe Method

McCabe method “A complexity measure” that brought forward the idea of cyclomatic complexity for the first time, and it basically measures decision points or loops of the program. This method showed the intelligibility, testability, and maintainability. Cyclomatic complexity utilizes a graph, which is derived from code. The formula defined as; [5]

$$MC=V(G)=e-n+2p \quad \ldots(1)$$

Where;
e: is the number of edges
n: is the number of nodes
p: is the number of connected components
McCabe's cyclomatic complexity metrics measures software complexity in program's structure, but it neglects the fact that length of a program is a factor of complexity [6].
McCabe's complexity measurement calculates total number of *possible* control paths through a program, using a control graph. In some programs, it is possible to have an infinite number of control paths. In order to address this issue, a combination of the basic control paths in a program is used to produce all possible paths [7].

5. Binary Search Complexity

The binary search [8] was taken as a case study to measure the complexity of the programs written in C++, Visual Basic and Java languages. The listing of the main program (test program) of the binary search algorithm was written in C++, Visual Basic and Java languages will be listed in the following sections; the method of binarySearch was not used in the comparison for simplicity.

5.1 C++ binarySearch Program

Bellow is the Listing of the binarySearch (test program) that was taken from Detiel book (How to program C++) [8];

```
// C++
// BinarySearch test program.
#include <iostream>
using std::cin;
using std::cout;
using std::endl;

#include "BinarySearch.h" // class BinarySearch definition

int main()
{
  int searchInt; // search key
  int position; // location of search key in vector

  // create vector and output it
  BinarySearch searchVector ( 15 );
  searchVector.displayElements();

  // get input from user
  cout << "\nPlease enter an integer value (-1 to quit): ";
  cin >> searchInt; // read an int from user
  cout << endl;

  // repeatedly input an integer; -1 terminates the program
  while ( searchInt != -1 )
  {
    // use binary search to try to find integer
    position = searchVector.binarySearch( searchInt );

    // return value of -1 indicates integer was not found
    if ( position == -1 )
      cout << "The integer " << searchInt << " was not found.\n";
    else
      cout << "The integer " << searchInt
        << " was found in position " << position << ".\n";

    // get input from user
    cout << "\n\nPlease enter an integer value (-1 to quit): ";
    cin >> searchInt; // read an int from user
    cout << endl;
  } // end while

    return 0;
  } // end main
```

Figure (1) shows the C++ flow graph of the binarySearch algorithm (main program)

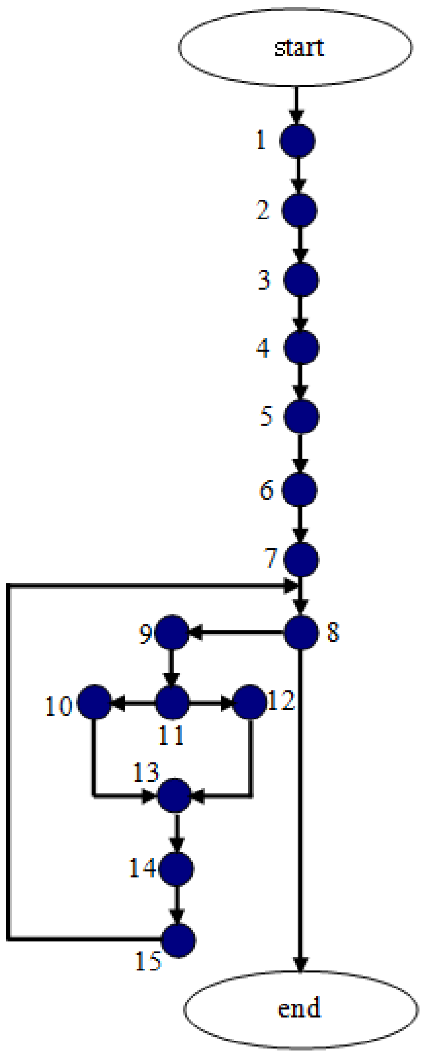

Figure 1: C++ languages-binarySearch flow graph

Table 2 shows the complexity of the main program binarySearch written in C++ language. With file size of this program is 1393 bytes.

Table 2: The complexity of the main program binary search written in C++

Complexity Type	Value
program length (in lines)	44
McCabe complexity	8
File size	1,393 bytes

5.2 Visual Basic binarySearch Program

Bellow is the Listing of the binarySearch (test program) that was written in Visual Basic taken from Detiel book (Visual Basic 2008 for Programmers) [9];

```
' Visual Basic
' Binary search of an array using Array.BinarySearch.
Imports System

Public Class FrmBinarySearchTest
   Dim array1 As Integer() = New Integer(19) {}

   ' create random data
   Private Sub btnCreate_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles btnCreate.Click

      Dim randomNumber As Random = New Random()
      Dim output As String = ("Index" & vbTab & "Value" & vbCrLf)

      ' create random array elements
      For i As Integer = 0 To array1.GetUpperBound(0)
         array1(i) = randomNumber.Next(1000)
      Next

      Array.Sort(array1) ' sort array to enable binary searching

      ' display sorted array elements
      For i As Integer = 0 To array1.GetUpperBound(0)
         output &= (i & vbTab & array1(i) & vbCrLf)
      Next
      txtData.Text = output ' displays numbers
      txtInput.Text = "" ' clear search key text box
      btnSearch.Enabled = True ' enable search button
   End Sub ' btnCreate_Click

   ' search array for search key
   Private Sub btnSearch_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles btnSearch.Click

      ' if search key text box is empty, display
      ' message and exit method
      If txtInput.Text = "" Then
         MessageBox.Show("You must enter a search key.", "Error", _
            MessageBoxButtons.OK, MessageBoxIcon.Error)
         Exit Sub
      End If

      Dim searchKey As Integer = Convert.ToInt32(txtInput.Text)
      Dim element As Integer = Array.BinarySearch(array1, searchKey)

      If element >= 0 Then
         lblResult.Text = "Found Value in index " & element
      Else
         lblResult.Text = "Value Not Found"
      End If
    End Sub ' btnSearch_Click
End Class ' FrmBinarySearchTest
```

Figure (2) shows the Visual Basic flow graph of the binarySearch algorithm (main program)

Table 2 shows the complexity of the main program binarySearch written in Visual Basic language. With file size of this program is 2026 bytes.

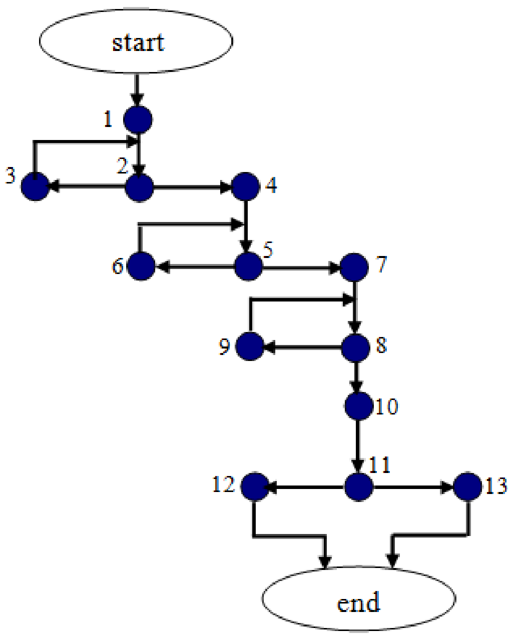

Figure 2: Visual Basic-binarySearch flow graph

Table 2: The complexity of the main program binary search written in C++

Complexity Type	Value
program length (in lines)	53
McCabe complexity	11
File size	2,026 bytes

5.3 Java binarySearch Program

Bellow is the Listing of the binarySearch (test program) that was written in Java taken from Detiel book (Java How to Program) [10];

```
// Java
// Sequentially Binary search an array for an item.
import java.util.Scanner;

public class BinarySearchTest
{
public static void main( String args[] )
{
    // create Scanner object to input data
     Scanner input = new Scanner( System.in );

      int searchInt; // search
      int position; // location of search key in array

     // create array and output it
     BinaryArray searchArray = new BinaryArray( 16 );
     System.out.println( searchArray );

     // get input from user
     System.out.print(
        "Please enter an integer value (-1 to quit): " );
     searchInt = input.nextInt(); // read an int from user

     // repeatedly input an integer; -1 will quit the program
     while ( searchInt != -1 )
     {
       // use binary search to try to find integer
       position = searchArray.binarySearch( searchInt );

      // return value of -1 indicates integer was not found
      if ( position == -1 )
        System.out.println( "The integer " + searchInt +
          " was not found.\n" );
      else
        System.out.println( "The integer " + searchInt +
          " was found in position " + position + ".\n" );

      // get input from user
      System.out.print(
          "Please enter an integer value (-1 to quit): " );
      searchInt = input.nextInt();
    } // end while
  } // end main
} // end class BinarySearchTest
```

Figure (3) shows the Java flow graph of the binarySearch algorithm (main program)

Table 3 shows the complexity of the main program binarySearch written in Java language. With file size of this program is 1,572 bytes.

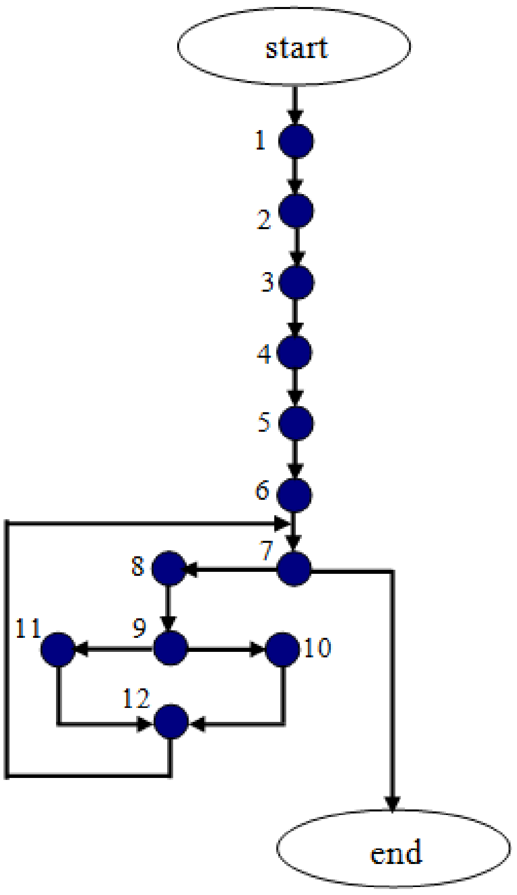

Figure 3: Java-binarySearch flow graph

Table 3: The complexity of the main program binary search written in C++

Complexity Type	Value
program length (in lines)	44
McCabe complexity	7
File size	1,572 bytes

6. Conclusion

Figure (4) shows the comparison between program length (in lines) and McCabe complexity of the three object oriented languages C++, Visual Basic and Java, by using the main program of the binarySearch algorithm that is used as a case study for comparison, the figure shows that the program length and complexity of C++ and Java are less than the Visual Basic, in this case we can predicate that the C++ and Java have less complexity than Visual Basic.

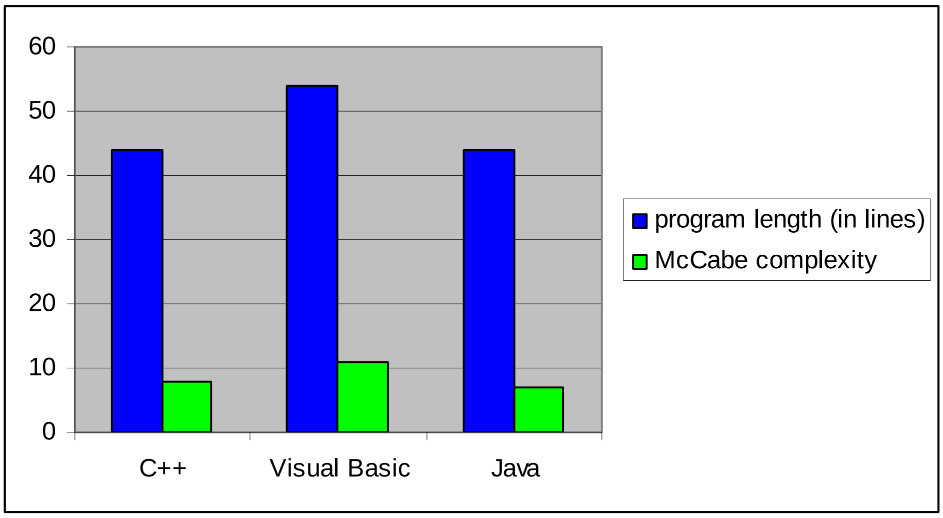

Figure 4: Comparison between program length and complexity of C++, Visual Basic and Java languages.

The languages with less number of keywords expected to be more complex and needs more access time compared with the languages that has more keywords. This can be used as a second pre-indictor for programs complexity.

References

[1] Benussi L., 1995, “Analysing the technological history of the Open Source Phenomenon”, FLOSS history, working paper, version 3.0, Department of Economics – University of Turin.

[2] Fatman R., 2000, “Software Fault Prevention by Language Choice: Why C is Not My Favorite Language”, Computer Science Division, University of California, Berkeley.

[3] Gopal N. et. al., 2004, “Distributed Parallel Virtual Machine: An Object-Oriented Approach”, Department of Computer Science, S.C.T College of Engineering.

[4] Cardoso, 2006, “Approaches to Compute Workflow Complexity”, Dagstuhl Seminar, The Role of Business Processes in Service Oriented Architectures, July 2006, Germany.

[5] McCabe 1976, “A Complexity Measure”, IEEE Transactions on Software Engineering, Vol. 2, No. 4, pp. 308-320, December 1976.

[6] Yanming C. et. al., 2007, “Exploration of Complexity in Software Reliability”, Tsinghua Science & Technology, Volume 12, Supplement 1, July 2007, Pages 266-269.

[7] Garcia 2008, “Software metrics through fault data from empirical evaluation using verification & validation tools”, Texas Tech University.

[8] Deitel H. et. al., 2004, "C++ How to Program", Pearson Prentice Hall.

[9] Deitel H. et. al., 2006, "Visual Basic 2008 for Programmers", Pearson Prentice Hall.

[10] Deitel H. et. al., 2005, "Java How to Program", Pearson Prentice Hall.