QUALITY ASSURANCE EVALUATION FOR PROGRAMS USING MATHEMATICAL MODELS

Murtadha M. Hamad and Shumos T. Hammadi
Faculty of Computers, Department of Computer Science, Al-Anbar University, Iraq

ABSTRACT

The purpose of this paper based on comprehensive quality standards that have been developed for program measurements. This paper adopted four measures to evaluate program performance (time complexity, reliability, modularity and documentary) evaluate on the basis of performance, these measures are based on mathematical models to evaluate the program . These measures applied on a sample of texts file that contain programs written in C++, so that was formed texts file that contain program to be evaluated . Analyzed the data obtained by using the algorithms proposed evaluation, which relied primarily on mathematical analysis, using mathematical functions to evaluate each program. C# was used as an environment in which software applied program evaluation. The results showed that the assessment depends on the structure and method of writing program.

KEYWORDS: *Quality Assurance, time complexity, reliability, modularity, documentary.*

I. INTRODUCTION

With increasing importance placed on standard quality assurance methodologies by large companies and government organizations, many software companies have implemented rigorous QA processes to ensure that these standards are met. The use of standard QA methodologies cuts maintenance costs, increases reliability, and reduces cycle time for new distributions. Modelling systems differ from most software systems in that a model may fail to solve to optimality without the modelling system being defective. This additional level of complexity requires specific QA activities. To make software quality assurance (SQA) more cost-effective, the focus is on reproducible and automated techniques [1].

In Software Quality, the definition should be as follows: software quality characterizes all attributes on the excellence of computer system such as reliability, maintainability and usability. In terms of practical application, software quality can be defined with three points on consistency: consistency with determined function and performance; consistency with documented development standard; consistency with the anticipated implied characteristics of all software specially developed [2].

Software quality is concerned with assuring that quality is built into the software products. Software quality assures creation of complete, correct, workable, consistent, and verifiable software plans, procedures, requirements, designs, and verification methods. Software quality assurance (SQA) adherence to those software requirements, plans, procedures, and standards to successive products.

The software quality discipline consists of product assurance and process assurance activities that are performed by the functions of SQA, software quality engineering, and software quality control [3]. Software quality assurance is that it is the systematic activities providing evidence of the fitness for use of the total software product. SQA is achieved through the use of established guidelines for quality control to ensure the integrity and prolonged life of software. SQA involves [4]:

- Establishing a Quality Assurance Group who has required independence.
- Participation of SQA in establishing the plans, standards and procedures for the project.

- Reviewing and auditing the software products and activities to ensure that they comply with the applicable procedures and standards.
- Escalating unresolved issues to an appropriate level of management.

In this paper will explain the affects of software Quality Evaluation and measurement approved to Software Performance Analysis. In the end, is discuss the results and the conclusions.

II. RELATED WORK

Several researches in the field of QA Evaluation have been done. There are a number of researchers and scientists used the methods of modelling technique based on the mathematical technique for evaluation to ensure the quality of the assessment. Some of these researches are summarized below:

Stefan Wagner, Florian Deissenboeck, and Sebastian Winter, This paper proposes that managing requirements on quality aspects is an important issue in the development of software systems. Difficulties arise from expressing them appropriately what in turn results from the difficulty of the concept of quality itself. Building and using quality models is an approach to handle the complexity of software quality. A novel kind of quality models uses the activities performed on and with the software as an explicit dimension. These quality models are a well-suited basis for managing quality requirements from elicitation over refinement to assurance. The paper proposes such an approach and shows its applicability in an automotive case study [5].

Manju Lata and Rajendra Kumar, This paper presented an approach to optimize the cost of SQA. It points out, how to optimize the investment into various SQA techniques and software quality. The detection and removal of defect is a software inspection providing technical support for the defect detection activity, and large volume of documentation are related to software inspection in the development of the SQA as a cost effective. The value of an inspection improves the quality and saves defect cost describe the optimization model for selecting the best commercial off-the-self (COTS) software product among alternatives for each module. As objective function of the models is to maximize quality within a budgetary constraint and standard quality assurance (QA) methodologies cuts maintenance costs. Increase reliability, and reduces cycle time for new distribution modelling system [6].

Holmqvist and Karlsson, The purpose of this work to improve the quality of software testing in a large company developing real-time embedded system. Software testing is a very important part of software development. By performing comprehensive software testing, the quality and validity of a software system can be assured. One of the main issues with software testing is to be sure that the tests are correct. Knowing what to test, but also how to perform testing, is of utmost importance. This thesis explores different ways to increase the quality of real-time testing by introducing new techniques in several stage of the software development model. The proposed methods are validated by implementing them in an existing and completed project on a subset of the software development process [7].

III. SOFTWARE QUALITY EVALUATION

Software quality directly affects the application and maintenance of software, so how to objectively and scientifically evaluate software quality becomes the hot spot in software engineering field. Software quality evaluation involves the following tasks throughout software life cycle and based on software quality evaluation standard, which is implemented during software development process: continuously measure software quality throughout software development process, reveal current status of software, predict follow up development trend of software quality, and provide effective means for buyer, developer and evaluator. A set of evaluation activities may generally include review, appraisal, test, analysis and examination, etc. Performance of such activities is aimed to determine whether software products and process is consistent with technical demands, and finally determine products quality. Such activities will change the phase of development, and may be performed by several organizations. A set of evaluation activities may be generally defined in the software quality specifications of project plan, special project, as well as related software quality specifications [8].

IV. SOFTWARE PERFORMANCE ANALYSIS

For software qualification, it is highly desirable to have an estimate of the remaining errors in a software system. It is difficult to determine such an important finding without knowing what the initial errors are. Research activities in software reliability engineering have been studied over the past 30 years and many statistical models and various techniques have been developed for estimating and predicting reliability of software and numbers of residual errors in software. From historical data on programming errors, there are likely to be about 8 errors per 1000 program statements after the unit test. This, of course, is just an average and does not take into account any tests on the program [9].

4.1 Time complexity

Important factors in measuring the efficiency or effectiveness of any algorithm is the amount of (execution time), the time it takes for the implementation of the algorithm. There are no simple rules to determine the time, so let us go to (appreciation prior) for the execution time using some of the mathematical techniques after knowing a number of important factors relating to the issue addressed by the algorithm. Identify the function that determines the expected time for implementation, depending on some variables related to the steps of the algorithm, suppose that the algorithm includes the following statement [10].

X=X+1;

Here we must account for the amount of time required to execute this statement alone, and then must know the frequency of implementation of the so-called (frequency Count). It differs according to the sample data and by multiplying the amounts in (the time of the statement and the amount of frequency) we get the Total Execution Time expected.
That calculation time of implementation of all instruct with the required accuracy of the information is needed for:

- Type of computer hardware that implement the algorithm.
- The programming language used in the computer.
- Time of implementation of all instruct.
- Kind of translator or interpreter.

Possible to know that information to choose a machine (computer) fact or definition of a computer by default, and in both cases, the calculated time may not be accurate and appropriate for a number of computers or any computer, as the language interpreter may vary from one computer to another as well as other factors. These considerations make us focus our appreciation in advance of the execution time on the number of iterations of code phrases directives. Take the following three examples:

…….	for(i=1;i<=n; i++)	for(i=1;i<=n; i++)
…….	…….	……...
…….	…….	……
X=X+1	…….	for (J=1;J<=n; J++)
……	X=X+1;	……
……	…….	…….
……	…….	X=X+1;
(A)	(B)	(C)

In the example (A)

That is a combination(X=X+1) not contained within any iterative formula, that the number of times executed (frequency Count=1).

In the example (B): A combination of repeated (n) times.

In the example (C): A combination of repeated (n^2) times.

If we assume (n = 10), these frequencies are (1, 10, 100), and this corresponds to ride a bicycle, riding a car, boarding a plane compared to the distance that will be interrupted by each vehicle per unit time (hour, for example) and here we use the expression (order of magnitude of algorithm) means the frequency of implementation of the phrase. The term (order of magnitude of a statement) sum of all iterations terms under which the executive and the assessment pre-determined execution time.
The example above shows that the algorithm (A) is the fastest implementation of the algorithm (B) and in turn faster than (C).
Example: We have a matrix (A) dimensions (n * n) is required to sum each row and store it in the matrix to the other was (sum) and then calculate the sum total of the components of the matrix (A). Can be the solution in two ways.
The first way:

```
Grandtotal=0;
for(i=1;i<=n; i++)
{
   Sum[i]=0;
   for(j=1;j<=n; j++)
   {
        Sum[i]=Sum[i]+A[i][j];
        Grandtotal= Grandtotal+ A[i][j];
    }
 }
```

2N additions

This cycle is repeated (N) of times

Total collection=$2N*N=2N^2$

The second way:

```
Grandtotal=0;
for(i=1;i<=n; i++)
{
   Sum[i]=0;
   for(j=1;j<=n; j++)
   {
        Sum[i]=Sum[i]+A[i][j];
   }

        Grandtotal= Grandtotal+ Sum[i];
 }
```

N^2 additions

N additions

N additions

Total collection=$N+ N^2$

We note here that the number of the first algorithm ($2N^2$) is greater than the number of the second algorithm ($N^2+ N$), so the first take longer than the second.

The following discussion considers the various statement types that can appear in a program and state the complexity of each terms of the number of steps [10]:

- Declarative Statement: these count as zero steps as these are no executable.
- Comment: these count as zero steps as these are no executable.

- Expression and Assignment Statement: most expression has a step count of one. The exceptions are expressions that contain function calls. In this case, we need to determine the cost of invoking the function.
- Iteration statements: this class of statement includes the (for, while and Do….while) statement. We shall consider the step counts only for the control part of these statements. The step count for each execution of control part of a for statement is one.
- Switch statement: This statement consist of a header followed by one or more sets of condition and statement pairs. The cost of header switch expression is given a cost equal to that assignable to expression. The cost of the each following condition statement pair is the cost of this condition plus that of all preceding conditions plus that of this statement.
- If –Then–else Statement: It consists of three parts:

```
If (exp)
        Statement1 {block of statements}
else Statement2 { block of statements}
```

  Each part is assigned the number of steps corresponding to <exp>, <Statement2>, < Statement2 >, respectively. Note that if the else clause is absent, then no cost is assigned to it.

- Function invocation: All invocation of procedures and function count as one step unless the invocation involves value parameters whose size depend on the instance characteristics.
- Function statements: these count as zero step as their cost has already been assigned to the invoking statement.

4.2 Reliability

There is no doubt that the reliability of a computer program is an important element of its overall quality. If a program repeatedly and frequently fails to perform, it matters little whether other software quality factors are acceptable.

Software reliability, unlike many other quality factors, can be measured directed and estimated using historical and developmental data. *Software reliability* is defined in statistical terms as "the probability of failure-free operation of a computer program in a specified environment for a specified time". To illustrate, program X is estimated to have a reliability of 0.96 over eight elapsed processing hours. In other words, if program X were to be executed 100 times and require eight hours of elapsed processing time (execution time), it is likely to operate correctly (without failure) 96 times out of 100.

Whenever software reliability is discussed, a pivotal question arises: What is meant by the term *failure?* In the context of any discussion of software quality and reliability, failure is non-conformance to software requirements. Yet, even within this definition, there are gradations. Failures can be only annoying or catastrophic. One failure can be corrected within seconds while another requires weeks or even months to correct. Complicating the issue even further, the correction of one failure may in fact result in the introduction of other errors that ultimately result in other failures [11].

4.3 Modularity

Modular programming is subdividing your program into separate subprograms such as functions and subroutines. For example, if your program needs initial and boundary conditions, use subroutines to set them. Then if someone else wants to compute a different solution using your program, only these subroutines need to be changed. This is a lot easier than having to read through a program line by line, trying to figure out what each line is supposed to do and whether it needs to be changed. And in ten years from now, you yourself will probably no longer remember how the program worked.

Subprograms make your actual program shorter, hence easier to read and understand. Further, the arguments show exactly what information a subprogram is using. That makes it easier to figure out whether it needs to be changed when you are modifying your program. Forgetting to change all occurrences of a variable is a very common source of errors. Subprograms make it simpler to figure out how the program operates. If the boundary conditions are implemented using a subroutine, your program can be searched for this subroutine to find all places where the boundary conditions are used.

This might include some unexpected places, such as in the output, or in performing a numerical check on the overall accuracy of the program.
Subprograms reduce the likelihood of bugs. Because subprograms can use local variables, there is less change that the code in the subroutine interferes with that of the program itself, or with that in other subprograms. The smaller size of the individual modules also makes it easier to understand the global effects of changing a variable [12].

4.4 Documentation

The system test also is concerned with the accuracy of the user documentation. The principle way of accomplishing this is to use the documentation to determine the representation of the prior system test cases. That is, once a particular stress case is devised, you would use the documentation as a guide for writing the actual test case. Also, the user documentation should be the subject of an inspection (similar to the concept of the code inspection), checking it for accuracy and clarity. Any examples illustrated in the documentation should be encoded into test cases and fed to the program [13].

V. PROPOSED ALGORITHMS FOR EVALUATION

To see if the programmatic product has a quality or not. There must be a standards assessment describes the programmatic product. In the software evaluation a mathematical models were used which are easy to measure and on that basis the values of four measures of the software is evaluated (*Time complexity*, *Reliability, Modularity*, *Documentation*). The next will explain each measure separately.

5.1. The Time complexity

Measurement of time is the time of performance, operating, or the so-called the execution time. Measuring the time adopted several measures to measure the execution time of software, as described in the chapter three, on which found the evaluation. The following algorithm describes the steps for finding the time:

Algorithm 1 *Time complexity measures of program.*
Input: *Text file of the program.*
Output: *Report of the Time complexity program.*

Step1: - Read Text file.
Step2: - Determine (Len ← Length of text file).

- Let t is two-dimension array
- k =0, is pointer on current state

Step3: - for (i =1; i < Len; i++).
Step4: - Determine (aa ← Token).
Step5: - Check aa

- Case aa= "}" then
  - if (t[0, k] == 1) then
  - t[1, k - 1] = t[1, k - 1] + t[1, k];
  - if (t[0, k] == 2) then
  - t[1, k - 1] = t[1, k - 1] + (t[1, k] * n);

  Else
  - k = k + 1;
  - t[1, k] = 0;
  - k = k - 1;

- Case aa = "for" OR aa= "while" OR aa= "do" then
  - k = k + 1;
  - t[0, k] = 2
  - while (aa != "{")
  - i = i + 1;
  - i = i - 1;

- Case aa = "{" then
  - if (t[0, k] != 2)
  - k = k + 1; t[0, k] = 1;
- Case aa = ";" then
  - t[1, k] = t[1, k] + 1

- End

Example:

```
#include <iostream.h>
// sequence is 0, 1, 1, 2, 3, 5, 8, 13, ...
int fib (int i)
{
  int pred, result, temp;
  pred = 1;
  result = 0;
  while (n > 0)
  {
    temp = pred + result;
    result = pred;
    pred = temp;
    n = n-1;
  }
  return(result);
}
int main ()
 {
  int n;
  cout << "Enter a natural number: ";
  cin >> n;
  while (n < 0)
  {
    cout << "Please re-enter: ";
    cin >> n;
  }
  cout << "fib(" << n << ") = " << fib(n) << endl;
  return(0);
}
```

It is easy to see that in the for loop the value of count will increase by a total of 6n. If count is zero to start with, then it will be 6n+9 on termination. So each invocation of sum execution a total of 6n+9 steps.

5.2. The Reliability Measure

To get on the reliable software must be reaching the number of errors in the programs to the lowest value as well as the loss the negative results which are resulting from them to the lowest level as possible. Where the first attempts to build the quality standards of the software went about the reliability of the programmatic product. The reason is the clarity of this attribute and easily measured as related to probability of failure for career and illnesses that occur in the software system during the operating effective for a long time. The reliability measuring was based on the two types of mathematical errors a division by zero and a negative value under the root, the following algorithm will show the reliability measurement:

Algorithm 2 *Reliability measures of program.*
Input: *Text file of the program.*
Output: *Report of the reliability program.*

Step1: - Read Text file.
Step2: - Determine (Len ← Length of text file).
Step3: - for (i =1; i < Len; i++).

Step4: - Determine (aa ← Mathematical expression).
Step5: - Check aa
- Case aa[i]= "/" or aa[i]= "%" then /* "/" is Division in C++ and "%" is Mod in C++
- n = i+1
- while (n != ";" OR n != ")") then
- If (aa[n] !="0") then
- n =n+1

endwhile
Step6: - " The program is not Reliability ".
Else
"The program is Reliability ".
Step7: - Case aa[i]= " sqrt " then // "sqrt" is root in C++
- n=i+1; while (n!=")") then
- if (aa[n] != "-") then
- n=n+1; endwhile
- Repeat Step6.

- End.

5.3. The Modularity Measure

Most programs consist of number of functions which are called when they are needed. The function is a set of instructions that can be called from anywhere in the main function to perform a specific task. The sub-functions (sub-programs) are characterized by have the same general structure of the main function in terms of defining variables and writing instructions. Among the benefits of the use of sub-functions, to simplify the problem to be solved and this by divided it in to a partial tasks (sub-functions). In some cases, the program will repeat a section or more the number of times, so the sub-programs (sub-functions) helps to reduce these repetitions by call this section each time by one step only. Evaluation has been adopted based on the number of existing functions, as explained in the following algorithm:

Algorithm 3 *Modularity measures of program.*
Input: *Text file of the program.*
Output: *Report of the modularity program.*

Step1: - Read Text file.
Step2: - Determine (Len ← Length of text file).
Step3: - for (i =1; i < Len; i++)
- Count=0,number to the Expressions reserved.

Step4: - Determine (aa ← Expressions reserved).
Step5: - Check aa
- if (aa= " void " or aa= " return ") then
- Count=Count + 1
- EndIf

Step5: - if (Count = 0) then
" The program is not modularity ".
Else
if (Count = 1) then
" The program is medium modularity ".
Else
if (Count >= 2) then
" The program is High modularity ".

- End.

5.4. The Documentation Measure

The Documentation is an important stage of building the software system. It is documents the internal construction of the program for the purpose of maintenance and development. Without documentation the stage of programs factory no longer able to follow-up their maintenance and development. Which

increases the financial cost and time for that program to the limits of unexpected or in other words, the failure to build software with high quality and long life cycle.
There is more than one way to documentation. For example, the programmer documentation is possibility to add comments within the software code. The analyst documentation during it the personal documents to explain the program cycle and the laboratory system documentation in which the points imbalance in the program are recorded. In this work the programmer documentation is adapted. Following algorithm describes the ratio of documentaries:

Algorithm 4 *Documentation measures of program.*
Input: *Text file of the program.*
Output: *Report of the Documentation program.*

Step1: - Read Text file.
Step2: - Determine (Len ← Length of text file).
Step3: - for (i =1; i < Len; i++).
- Count=0, number of Symbolic expressions.
Step4: - Determine (aa ← Symbolic expressions).
Step5: - if (aa [i] = "//" or aa [i] = "*/") then // "//" & "/*" is refer to Document in C++
Count=Count + 1
EndIf
Step6: - if (Count = 0) then
" The program is not Documentation ".
Else
if (Count = 1) then
" The program is medium Documentation".
Else
if (Count >= 2) then
" The program is High Documentation".

- End.

VI. EXPERIMENTAL RESULTS

Implementation of the proposed evaluation algorithm on program written in c++ language in text file Appendix A, using mathematical analysis of these program. In this paper, evaluated six program by using four measures (time complexity, reliability, modularity, documentary)
See appendix A include some sections of code used to implement the algorithm.
In the Table 1 notes the ratio of evaluations of software in accordance with QA standards adopted for each program: the time ***complexity, reliability, modularity*** *and* ***documentation***, the evaluation found using mathematical models.
The results after the implementation, the prog.2 was the highest rate of the time complexity, the prog.6 was the lowest time complexity. Clear that the programs (prog.4) from an arithmetic error and consequently appear that they are not reliability.

Table 1. Evaluate of Software

Name of program	*Time Complexity*	*Reliability*	*Modularity*	*Documentation*
prog.1	*333*	*The program is reliable*	*The program is high Modular*	*The program is highly documented*
Prog.2	*2872*	*The program is reliable*	*The program is high Modular*	*The program is highly documented*
prog.3	*49*	*The program is reliable*	*The program is high Modular*	*The program is medium documented*
prog.4	*21*	*The program is not reliable*	*The program is high Modular*	*The program is highly documented*

prog.5	*39*	*The program is reliable*	*The program is medium Modular*	*The program is highly documented*
prog.6	*14*	*The program is reliable*	*The program is medium Modular*	*The program is medium documented*

VII. CONCLUSIONS

The study aimed to shed light on the concept of TQM in the evaluation of software by discussing the different intellectual visions that dealt with the overall quality standards and models. Mathematical analysis was used for evaluation depending on the standard model to evaluate the programs adopted this model to four measures of evaluation. Also the time it takes for the implementation of the algorithm there are no simple rules to determine the time, so the execution time using some of the mathematical techniques after knowing a number of important factors.
Reliability depends on conceptual correctness of algorithms, and minimization of programming mistakes, such as logic errors (such as division by zero or off-by-one errors).Modular benefits of the use of sub-functions, to simplify the problem to be solved and this by divided it in to a partial tasks (sub-functions).Using documentation as a guide for writing the actual test case, checking it for accuracy and clarity. In the future we can use other linear models to evaluate the software and we can dealing with software to test those which are more complex..

Appendix A:

Multiplying a vector by a square matrix many times

```
#include <iostream<
#include <iomanip<
#include <fstream<
#include <cmath<
using namespace std;
void mat vec (int, double[][50], double[], double[])
int main()
}
int n, i, j, norm;
double b[50],c[50],a[50][50];
cout << endl;
cout << " Normalize the vector after each projection?" << endl;
cout << " Enter 1 for yes, 0 for no" << endl;
cout << " -------------------------" << endl;
cin >> norm;
 ---//Read the matrix and the vector:
ifstream input data;
input data.open("matrix v.dat");
input data >> n;
.
.
.
.
.
.
.
for (i=1;i<=n;i++)
}
b[i]=c[i];
{
if(norm == 1)
}
double rnorm = 0;
for (i=1;i<=n;i++)
```

```
}
rnorm = rnorm + b[i]*b[i];
{
rnorm = sqrt(rnorm);
for (i=1;i<=n;i++)
}
b[i]=b[i]/rnorm;
{
{
cout << " Projected vector at stage: " << icount;
cout << "\n\n";
for (i=1;i<=n;i++)
}
cout << setprecision(5) << setw(10);
cout << b[i] << endl;
{
icount = icount+1;
cout << " One more projection? "<< endl;
cin >> more;
{
return 0;
{
----------------------------------------------------*/
function mat vec performs matrix-vector
multiplication: c i = a ij b j
/*----------------------------------------------------
void mat vec (int n, double a[][50], double b[], double c([]
}
int i, j;
for (i=1;i<=n;i++)
}
c[i] = 0;
for (j=1;j<=n;j++)
}
c[i] = c[i] + a[i][j]*b[j];
{
{
{
```

Time complexity	333
Reliability	*The program is Reliability*
Modularity	*The program is highly Modularity*
Documentation	*The program is highly documented*

```
for (int i = 0; i < len; i++)
        {
          if (aa.Substring(i, 1) == "}")
          {
            if (t[0, k] == 1)
              t[1, k - 1] = t[1, k - 1] + t[1, k];
            else
              if (t[0, k] == 2)
                t[1, k - 1] = t[1, k - 1] + (t[1, k] * n);
              else
                k = k + 1;

            t[1, k] = 0;
            k = k - 1;
          }
else
```

```
if ((aa.Substring(i, 3) == "for") || (aa.Substring(i, 5) == "while") || (aa.Substring(i, 2) == "do"))
{
        k = k + 1;
        t[0, k] = 2;
        while (aa.Substring(i, 1)!="{")
         { i = i + 1;
}
```

REFERENCES

[1] Michael R. Bussieck, Steven P. Dirkse, Alexander Meeraus and Armin Pruessner, "Software Quality Assurance for Mathematical Modeling system ", Springer 2005.

[2] Yang Aimin and Zhang Wenxiang, "Based on Quantification Software Quality Assessment Method", Computer and Information Technology College, Zhejiang Wanli University , Ningbo, CHINA, JOURNAL OF SOFTWARE, VOL. 4, NO. 10, DECEMBER 2009.

[3] Bryan O'Connor, "Software Assurance Standard Nasa Technical Standard", NASA-STD-8739.8 w/Change 1, July 28, 2004.

[4] Yujuan Dou, "Software Quality Assurance Framework (SQA)", 2008/11/28.

[5] Stefan Wagner, Florian Deissenboeck, and Sebastian Winter, " Managing Quality Requirements Using Activity Based Quality Models", Institute for Informatics Technische University Munchen Garching b. Munchen, Germany, ISBN: 978-1-60558-023-4, 2009.

[6] Manju Lata and Rajendra Kumar, "An Approach to Optimize the Cost of Software Quality Assurance Analysis", Dept. of Compute Science & Engg, International Journal of Computer Applications (0975 – 8887), Volume 5– No.8, August 2010.

[7] Holmqvist J. and Karlsson K., "Enhanced Automotive Real-TimeTesting through Increased Development Process Quality", (2010).

[8] Yang Aimin and Zhang Wenxiang, "Based on Quantification Software Quality Assessment Method", Computer and Information Technology College, Zhejiang Wanli University , Ningbo, CHINA, JOURNAL OF SOFTWARE, VOL. 4, NO. 10, DECEMBER 2009.

[9] Pham H, "Software Reliability", a chapter in Wiley Encyclopedia of Electrical and Electronic Engineering, Wiley: pp 565-578, 2000.

[10] Essam al-Saffar, "data structures", Faculty of Rafidain University, Department of Computer, Baghdad 2001.

[11] Roger S. Pressman, "Software Engineering", Software engineering: a practitioner's approach, Ph.D. thesis, ISBN 0-07-365578-3,2001.

[12] http://www.eng.fsu.edu/~dommelen/courses/cpm/notes/progreq/

[13] Glenford J. Myers, "The Art of Software Testing", John Wiley & Sons, Inc., 2004.Study ",ICGST-GVIP,ISSN 1687-398X,Volume (8),Issue (III),India, October 2008.

Authors

Murtadha Mohammad Hamad received his MSc degree in computer science from University of Baghdad, Iraq. , in 1991, received his PhD degree in computer science from University of Technology in 2004, and received the Assist Prof. title in 2005. Currently, he is a dean of College of Computer, University of Anbar. His research interested includes DataWarehouse, Software Engineering, and Distributed Database.

Shumos Taha Hammadi graduated from the College of Computer Department of Computer Science University of Anbar, Iraq. Currently, she is master student in the end of research phase.