3. Software Testing Metrics [2,3,4]

Process indictors give insight into the dynamics of a given software process enabling project management to evaluate the efficacy of that process and the metrics represents a quantitative measure of the degree to which a system component or process possesses a given attribute.

The classifications of metrics are:-

1. Process / Project

- People.
- Effort expended (force).
- Schedule expended (distance).
- Budget expended (efficiency).
- Productivity rates(pages / hour, SLOC / day).
- Error rates and counts.
- Defect rates and counts.

2. Product Metrics

- Size (Function Point (FP), KSLOC, Modules, Subsystems, Pages and Documents).
- Error density(Errors/ KSLOC).
- Defect density (Defects / KSLOC or Defects / Function Point).
- Quality (Correctness ,Maintainability, Reliability , Efficiency, Integrity, Usability, Flexibility, Testability , Portability, Reusability , Availability, Complexity ,Understandability , Modifiability).

4. Software Testing Strategies[5,6]

The objective of testing is to reduce the risk inherent in computer systems.
The strategy must address the risk and present a process that can reduce these risks. The two components of the testing strategy are the test factors and the test phase, defined as follows :-

1. **Test Factor :-** The risk or issue that needs to be addressed as part of the test strategy . The strategy will select those factors that need to be addressed in the testing of a specific application system.

2. **Test Phase :-** The phase of the systems development life cycle in which testing will occur .

4.1. Test Factor [3]

In designing a test strategy, the risk factors become the basis or objective of testing. The risks associated with testing will be called "test factors" while the test factors themselves are not risks, they are attributes of the software. The test factors enable the test process to be logically constructed like other parts of information services. The following describes the test factors:-

4.1.1. Stress Test (Service level)[5,6]

Assurance that the deserved results will be available with in a time frame acceptable to the user. To achieve the desired service level it is necessary to match user requirements with available resources. Resource includes input / output capabilities, communication facilities, processing, and systems software capabilities. The metrics used to compute the stress testing is :-

1.Compute The Size Of Program (Size Metrics) [7,8]

There are many possibilities for representing , the size (or magnitude) of program including the amount of memory required to contain the compiled version of it. The principal size measures that have been found the most useful in characterizing software include such as lines of code, number of tokens , number of functions , and some other measures based on these basic ones. The size of a program is an important measure for primarily these reasons:-

1. It is easy to compute often the program is completed.
2. It is the most important factor for many models of software development.
3. Productivity is normally based on a size measure.

2.Complexity Metrics [2,9,10]

A variety of software metrics can be completed to determine the complexity of program control flow .Many of these are based on a representation called a flow graph, a graph is a representation composed of nodes and links (also called edges),When the links (edges) are directed, the flow graph is directed graph.

Complexity metrics can be used to predict critical information about reliability and maintainability of software system from automatic analysis of source code. The cyclomatic complexity may be used to provide a quantitative indication of maximum module size.

Cyclomatic complexity as a guide for the design of white-box .Complexity is computed in one of four ways :-

1. The number of regions of the flow graph corresponds to the cyclomatic complexity .

2. Cyclomatic complexity, V(G) for flow graph G is defined as

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

   where :-

   E : Is the number of flow graph edges .

   N : Is the number of flow graph nodes

3. Cyclomatic complexity V(G) for flow graph G is also defined as

   $V(G) = P + 1$

   where :-

   P: Number of predicate nodes contained in the flow graph G

4. Cyclomatic complexity V(G) for flow graph **G** is defined as

   $V(G) = m - n + p$

   where :-

   m : Number of branches in graph G .

   n : Number of nodes in graph G

   p : Number of separate parts in graph G

Note :-

Predicate node is the node that contains a condition and is characterized by two or more edges branches from it.

4.1.2. Coupling Testing [2,3,8,11]

Coupling can be defined as the degree of interaction between two modules but cohesion is the degree of interaction within a module. Module coupling provides an indication of the "connected" of a module to other modules by global data and the outside environment.

Module coupling measures the quality of the connection between modules. Dham has proposed a metrics for module coupling that encompasses data and control flow coupling , global coupling , and environmental coupling . The measures required to compute module coupling are defined in terms of each of the three coupling types noted above :-

For data and control flow coupling :-

d_i = Number of input data parameters .

c_i = Number of input control parameters .

d_o = Number of output data parameters .

c_o = Number of output control parameters .

For global coupling :-

g_d = Number of global variables used as data .

g_c = Number of global variables used as control .

For environmental coupling :-

w = Number of modules called (fan- out).

r = Number of modules calling the module under consideration (fan-in) .

Using these measure, a module coupling indicator, m_c is defined in the following way :-

$$\mathbf{m_c = 1/M}$$

where :-

$$\mathbf{M = d_i + ac_i + d_o + bc_o + g_d + cg_c + w + r}$$

where :-

$$\mathbf{a = b = c = 2}$$

The higher value of mc , **the lower** overall module coupling. In order to have the coupling metrics move upward as the degree of coupling increases, a revised coupling metrics may be defined as :-

$$\mathbf{C = 1 - m_c}$$

Where the degree of coupling increases nonlinearly between a minimum value in the range 0.66 to a maximum value that approaches 1.0.

4.1.3. Performance Testing [3]

The amount of computing resources and code required by a system to perform its stated functions .

Performance includes both the manual and automated segments involved in fulfilling system functions .

The performance of program can be measured by using these metrics.

1. Run Time Metrics [12,13]

The running time of a program depends on factors as :-

1. The input to the program (running time is a function of input).

2. The quality of code generated by the compiler used to create the object program .

3. The nature and speed of instructions on the machine used to execute the program.

4. The time complexity of algorithm underlying the program, often the running time depends not on the execute input only but on the size of input .

The runtime can be computed by :-

***Big–Oh Notation Of Running Time [11]**

In this section we present some simple rules for determining a Big- Oh notation of the running time of the basic compound statements.

Rule 1 : For Loops

The running time of a for loops at most the running time of the statement inside the for loop (including test) times the number of iterations.

Rule 2 : Nested Loops

Analyze these inside out . The total running time of a statement inside a group of nested loops is the running time of the statement multiplied by the product of the sizes of all the loops .

Rule 3 : Consecutive Statements

These just add which means that the maximum is the one that counts, see this rule :-

If $T_1(N) = O(f(N))$ and
$T_2(N) = O(g(N))$ then :-

a - $T_1(N) + T_2(N) = \max(O(f(N)), O(g(N)))$
b - $T_1(N) * T_2(N) = O(f(N) * g(N))$

Rule 4 : If Else

For the fragment

If (Condition)
S_1
else
S_2

The running time of an if else statement is never more than the running time of the test plus the larger of the running times of S_1 and S_2.

Clearly this can be an overestimate in some cases, but it is never an underestimate.

Other rules are obvious , but a basis strategy of analyzing from the inside (or deepest part) out works . If there are function calls, these must be analyzed first, if there are recursive functions, there are several options. If the recursion is really just a thinly veiled for loop, the analysis is usually trivial.

2. Memory Requirements [9]

It is difficult to formulate any overall technique for storage analysis, since an analysis is so initially related to the problem details. The data structures and language chosen and the design approach, the approach taken here separately considers the problem, the program which the problem and the data storage associated with the program.

On analysis of the problem leads to an approach, a list of variables, possible algorithms and input and output data

requirements, on the basis of problem analysis. The analysis then considers the language. Computer system and storage media and:-

1 .Counts program length in terms of operations and operands into memory location.

2. Estimates the amount of input, intermediate and output memory storage required and convert this into memory locations.

The initial storage analysis is completed when the program and data storage needs are summed and operational among the various long and short term storage devices. The memory requirement can be computed by :-

Memory requirement = program length + variable and constant storage length + data structure storage length

The program length is computed by using the Halstead measure , this measure is used to compute the length of program by applying the equation :-

$$N = n_1 \text{Log}_2 n_1 + n_2 \text{Log}_2 n_2$$

where :-

n_1 = Number of distinct operators that appear in a program .

n_2 = Number of distinct operands that appear in a program

The estimate of data – structure storage length must come from analysis of the structures and algorithms to be used in the design.

The variable and constant storage length come from an analysis of input parameters and constants, the temporary storage required and the output parameters required, we can obtain the machine instructions length by dividing the number of takes.

4.1.4. Effort Test [2,9]

Testing effort can also be estimated using metrics derived from Halstead measure by using the definition for program volume , V , and program level , PL , software science effort , e , can be computed as : -

$$PL = 1 / [(n_1 / 2) * (N_2 / n_2)] \quad \text{……1}$$

$$V = N \text{Log}_2 (n_1 + n_2) \quad \text{……2}$$

$$e = V / PL \quad \text{……3}$$

where :-

$$N = n_1 \text{Log}_2 n_1 + n_2 \text{Log}_2 n_2 \quad \text{…4}$$

n_1 = Number of distinct operators that appear in a program .

n_2 = Number of distinct operands that appear in a program .

N_2 = Total number of operand occurrences .

The percentage of overall testing effort to be allocated to a module K can be estimated using the following relationship :-

Percentage of testing effort **(K)**=

$$e\ (K) / \sum e\ (i) \quad \text{……5}$$

where :-

e (K) is computed for module **K** using this equations (3) and the

summation in the nominator of equation (5) is the sum of software science effort across all modules of the system .

4.2. Execution Phase Test [3,6]

The objective of this phase is to determine whether the software system will perform correctly in an executable mode as it would be in an operational environment . Any deviation from the expected results should be recorded during this phase . Depending on the nature and severity of those problems , uncovered changes may need to be made to the software before it is placed in a production status . If the finding / problems are extensive it may be necessary to stop testing completely and return the software to the developers to make the needed changes prior to restoring testing .

The execution of this phase involves performing the following three tasks :-

Task 1 :- Build Test Data

Information services personnel and users have been using test data since the inception of computer programming. The concept of test data is simple one—creating representative processing conditions using test transactions .

Task 2 :- Execute Tests

There are many methods of testing an application system. The test term is concerned with that all of these forms of testing which occur so that the organization has the highest probability of success when installing a new application system .The following types of tests should be addressed by the test team during the test phase:-

1. **Service Testing.**
2. **Coupling Testing.**
3. **Performance Testing.**
4. **Effort Testing.**

Task 3 :- Record Test Result

The test report should indicate what works and what does not work. The test report should also give the tester's opinion regarding whether he or she believes the software is ready for operation at the conclusion of this test step.

The executing tests and recoding result are illustrated in Fig.(1) .

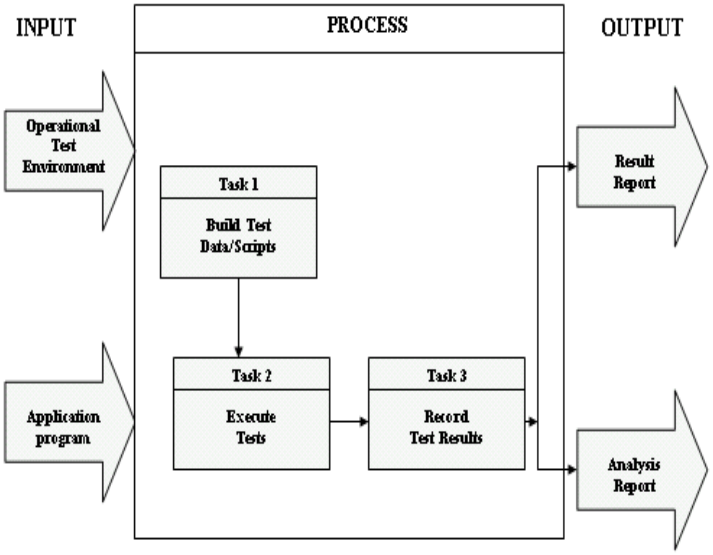


Fig.(1) Execute Dynamic Tests And Record Results

5. System Implementation

Systematic software testing strategies are test programming. This program aims to test software model such as Pascal, C, C++ program.

After the user completes the design of program and writes the code of program, the system begins to test the application program by applying the metrics of testing strategy until the operation is

complete. When the system completes the operation , the SSTS system displays the report to show the application program verification the system requirement and also to give an idea to the user about whether, the program is good or not , therefore , this application program can be used by all user .

An important advantage of applying the metrics of testing strategies over the application in execution step is that a guide to show the degree of complexity of application program , performance of application program , and computing the degree of coupling of program. These guides are easy to understand and help developers/ designers to specify the application program verification the system requirement and to perform the maintenance operation in an efficient way.

6. The Result Analysis

After complete the operation of testing the application program is shown in appendix , the analyze of the results of metrics :-

1. When the degree of complexity is high, then the number of test path is high because it is equal to complexity value and the need is to test all the path at least once, show Fig.(1). This value can be used to predict critical information about reliability and maintainability of application program.

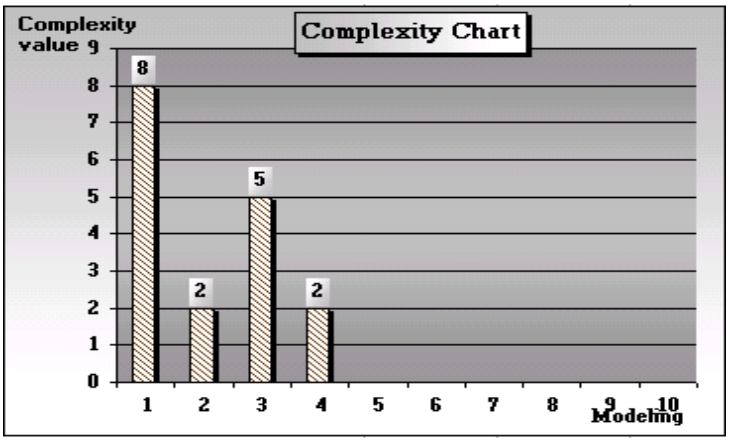


Fig.(1) Complexity Chart

2. The performance of program can be computed by a count of the total runtime of program and the memory space requirement to store and execute the application program. From this example it can be shown when the speed of application program is fast and the memory requirement is not for big spaces, then this application program is desired, but when the speed is not fast and memory requirement is for big space, then this application program is not desired and there is a need to redesign the application program, show Fig.(2) and Fig.(3).

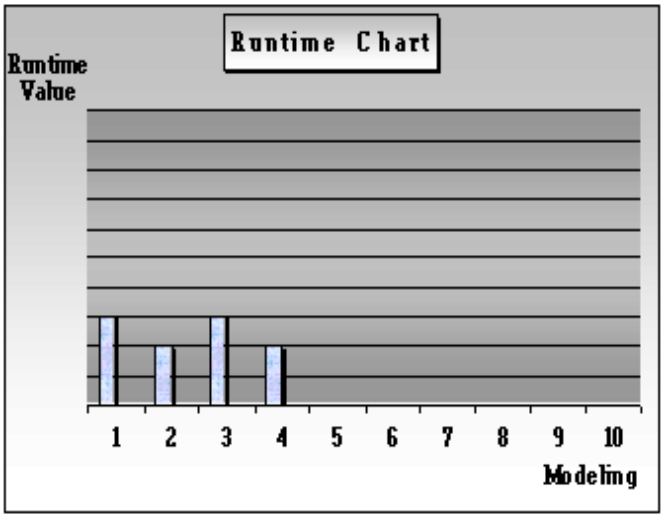


Fig.(2) Runtime Chart

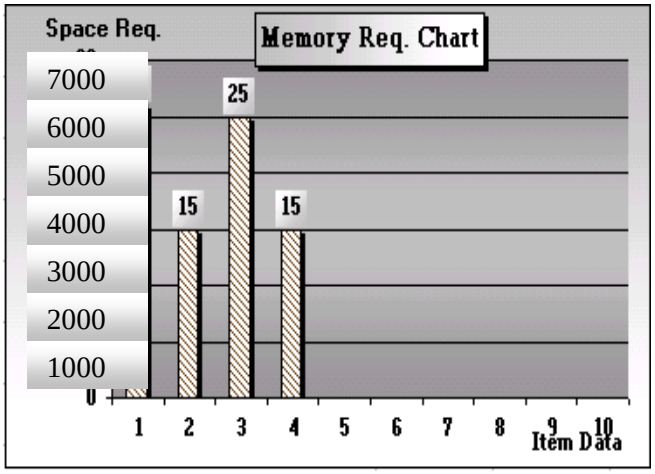


Fig.(3) Memory Req. Chart

3. The result of effort test of application program can be used to estimate the effort needed to test the program. When the effort test value increases then the effort is needed to test the application program increase, then software engineering aims to design program with the least effort. Show Fig.(4)

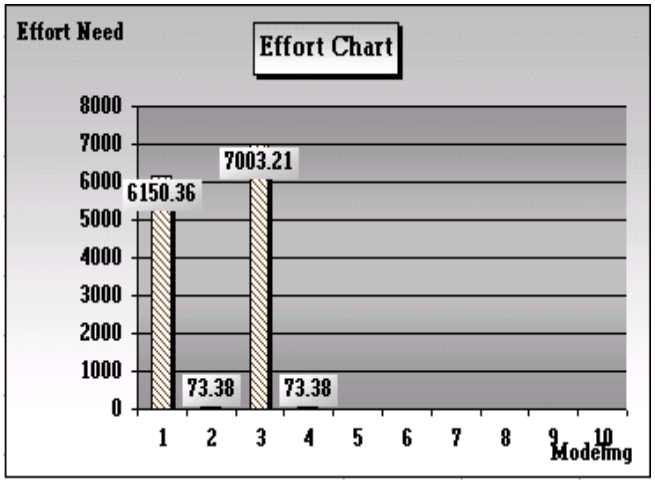


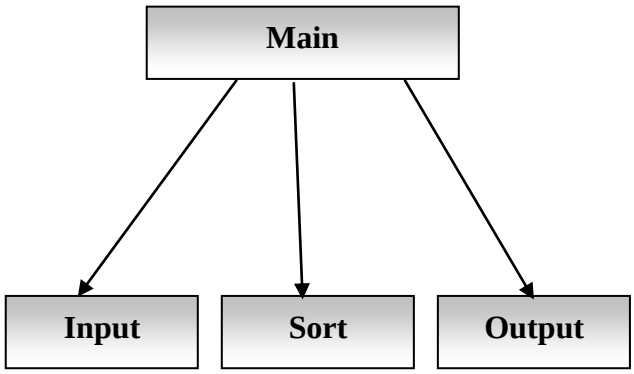
Fig.(4) Effort Chart

4. When the number of input and output of application program increases , then the number of resources requirement increases, In this example the number of resource req. = 4.

5. When the result of coupling test is high, then the degree of connection between the models are high. In this example the result of coupling metrics are represented in Fig.(5) and the sum number of fan-in = 3 and fan-out =3 .

Name	Fan-in	Fan-out
Main	**0**	**3**
Input	**1**	**0**
Sort	**1**	**0**
Output	**1**	**0**

(a) Fan-in & Fan-out Result



(b) Structure Coupling Of application Program

Fig.(5)

7. Conclusions

1. The value of complexity metrics can be used to specify the degree of complexity of program and the number of test path, therefore, the value of complexity can be used to predict critical information about reliability and maintainability of software model.

2. The performance of program can be computed by a count of he total runtime of program and the memory space requirement to store and execute the software model.

3. The result of effort test of application program can be used to estimate the effort needed to test the program.

4. When the number of input and output of application program increases , then the number of resources requirement increases.

5. When the result of coupling test is high, then the degree of connection between the models are high.

8. *Reference*

[1] McGregor , John D. , Sykes , David A. , " Particle Guide to Testing Object - Oriented software ", Addison - Wesley,Boston,2001 .

[2] Pressman , Roger S. , " Software Engineering " , McGraw – Hill , Singapore, 1997 .

[3] Mark , Kochanski , " Software engineering for Testing " , Paper by Internet on the site : www.software-engineer.org .2000

[4] Kemerer , Chris F. , " Software Process And Project Metrics " , on the site : www.ootesting.com . 2001

[5] Perrly , William E. , " Effective Methods For Software Testing " , John Wiley & Sons , Canada , 2000 .

[6] Pfleeger , Shari Lawernce , " Software Engineering theory and practice " , Second Edition , Prentice - Hall , Inc, New Jersey , 2001.

[7] Sunsmore , H.E. , Conte , S. D. , " Software Engineering Metrics And Models ",The Benjamin/ Cummings, 1986 .

[8] Jia , Xiaoping , " Object - Oriented Software Development Using Java " ,Addison - Wesley , USA , 2000 .

[9] Shooman , Martin L. , " Software Engineering Design , Reliability , And , Management " , Mcgraw - Hill , Singapore , 1988 .

[10] Zainab Mohammed , " Design Complexity Measurement and testing" , MSC , University of technology,1998.

[11] HawryZkiewycz , Igor , " Introduction to systems analysis and design " , Prentice Hall , Australia, 1998 .

[12] Wiess , Mark Allen, " Data structure & Algorithms Analysis in C++ " , Second edition , Addison Wesley , Inc , Canada , 2003 .

[13] Preiss , Bruno R. , " Data structures and Algorithms with Object — Oriented Design Patters In Java " , John Wiley & Sons , Inc , New York , 2000 .

"مكننـة ستراتيجيات اختبار البرمجيات"

الخـــلاصـــة

ان اهـم الاشـياء فـي هندسـة البرمجيـات هـي ملاحظة كيفيـة القيـام بعمليـة اختبـار اي برنـامج بواسـطة تطبيـق مجموعـة مـن سـتراتيجيات الاختبـار ثـم بعد ذلـك ملاحظـة النتـائج لكـي يـتم تحديـد مـدى كفـاءة وجـودة البرنـامج الـذي تـم اختباره .

في هذا البحث قد تم اختبار البرامج التي هي ام ان تكـون (Pascal, C, C++) مـن خـلال تطبيق مجموعة من المقاييس وهي:-

- حساب المساحـة المطلوبة.
- حسـاب مقـدار الجهـد.
- حساب درجـة الشـد.
- حساب درجـة التعقيـد.
- حساب درجـة الترابـط.
- حسـاب وقـت التنفيـذ .