Module 3 :
Testing Metrics for Monitoring and Controlling the Testing Process

Pooja Malhotra

Software Metrics

Metrics can be defined as “STANDARDS OF MEASUREMENT”.

Software Metrics are used to measure the quality of the project. Simply, Metric is a unit used for describing an attribute. Metric is a scale for measurement.

Suppose, in general, “Kilogram” is a metric for measuring the attribute “Weight”. Similarly, in software, “How many issues are found in thousand lines of code?”

Test metrics example:

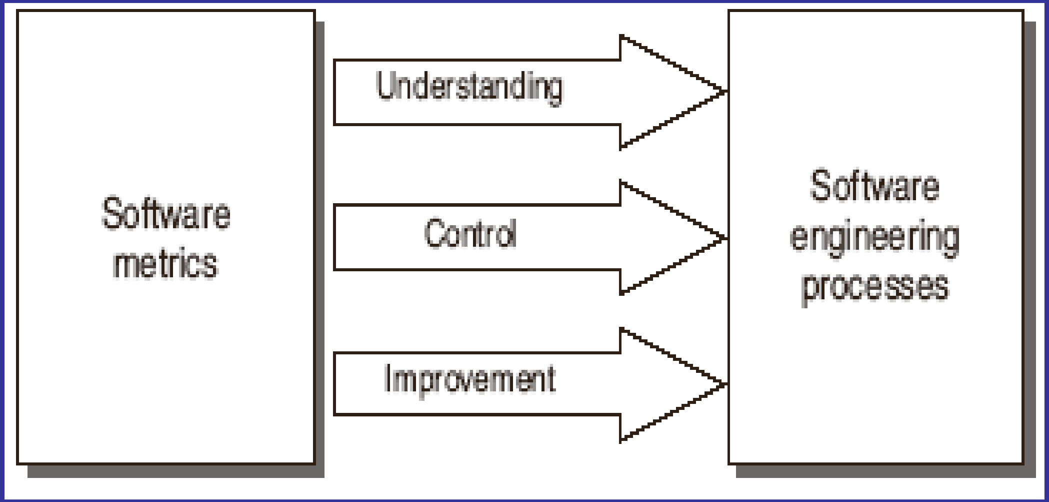
How many defects exist within the module?

How many test cases are executed per person?

What is the Test coverage %?

Need of Software Measurement

- Understanding
- Control
- Improvement



Software Metrics

Product Metrics

Measures of the software product at any stage of its development

From requirements to installed system.

- Complexity of S/W design and code
- Size of the final program
- Number of pages of documentation produced

Process Metrics

Measures of the S/W development process

- Overall development time
- Type of methodology used
- Average level of experience of programming staff

Measurement Objectives for Testing

The objectives for assessing a test process should be well defined.

GQM(Goal Question Metric) Framework:

- List the major goals of the test process.
- Drives from each goal, the questions that must be answered to determine if the goals are being met.
- Decides what must be measured in order to answer the questions adequately.

Attributes and Corresponding Metrics in Software Testing

Category	Attributes to be Measured
Progress	• Scope of testing • Test progress • Defect backlog • Staff productivity • Suspension criteria • Exit criteria
Cost	• Testing cost estimation • Duration of testing • Resource requirements • Training needs of testing group and tool requirement • Cost-effectiveness of automated tool
Quality	• Effectiveness of test cases • Effectiveness of smoke tests • Quality of test plan • Test completeness
Size	• Estimation of test cases • Number of regression tests • Tests to automate

Attributes : Progress

- Scope of testing: Overall amount of work involved
- Test Progress: Schudule , budget, resouces
  - Major milestones
  - NPT
  - Test case Escapes (TCE)
  - Planned versus Actual Execution (PAE) Rate
  - Execution Status of Test (EST) Cases(*Failed*, *Passed*, *Blocked*, *Invalid* and *Untested)*
- *Defect Backlog: Number of defects that are unresolved/outstanding*
- *Staff productivity: Time spent in test planning , designing, Number of test cases developed.(<u>useful to estimate the cost and duration for testing activities)</u>*

Attributes : Progress

Suspension criteria: It describe the circumstances under which testing would stop temporarily.

- Incomplete tasks on critical path, Large volume of bugs, critical bugs, incomplete test environment

Exit criteria: It indicates the conditions that move the testing activities forward from one level to the next.

- Rate of fault discovery in regression tests, frequency of failing fault fixes , fault detection rate.

Attributes: Cost

Cost	• Testing cost estimation • Duration of testing • Resource requirements • Training needs of testing group and tool requirement • Cost-effectiveness of automated tool

Attributes: Cost

Effectiveness of Test Cases

- Number of faults found in testing.
- Number of failures observed by the customer which can be used as a reflection of the effectiveness of the test cases.

- Defect age

Defect age is used in another metric called defect spoilage to measure the effectiveness of defect removal activities.

- Spoilage = Sum of (Number of Defects x defect age) / Total number of defects

Effectiveness of Test Cases

- Defect Removal Efficiency (DRE) metric defined as follows:

$$\text{DRE} = \frac{\text{Number of Defects Found in Testing}}{\text{Number of Defects Found in Testing} + \text{Number of Detects Not Found}}$$

Spoilage Metric

- Defects are injected and removed at different phases of a software development cycle
- The cost of each defect injected in phase X and removed in phase Y increases with the increase in the distance between X and Y
- An effective testing method would find defects earlier than a less effective testing method .
- An useful measure of test effectiveness is defect age, called PhAge

Phase Injected	Phase Discovered							
	Requirements	High-Level Design	Detailed Design	Coding	Unit Testing	Integration Testing	System Testing	Acceptance Testing
Requirements	0	1	2	3	4	5	6	7
High-Level Design		0	1	2	3	4	5	6
Detailed Design			0	1	2	3	4	5
Coding				0	1	2	3	4

Table: Scale for defect age

Spoilage Metric

Phase Injected	Phase Discovered								Total Defects
	Requirements	High-Level Design	Detailed Design	Coding	Unit Testing	Integration Testing	System Testing	Acceptance Testing	
Requirements	0	7	3	1	0	0	2	4	17
High-Level Design		0	8	4	1	2	6	1	22
Detailed Design			0	13	3	4	5	0	25
Coding				0	63	24	37	12	136
Summary	0	7	11	18	67	30	50	17	200

Table: Defect injection and versus discovery on project *Boomerang*

A new metric called spoilage is defined as

$$\text{Spoliage} = \frac{\sum (\text{Number of Defects} \times \text{Discovered Phase})}{\text{Total Number of Defects}}$$

Spoilage Metric

Phase Injected	Phase Discovered								Weight	Total Defects	Spoilage = Weight/Total Defects
	Requirements	High-Level Design	Detailed Design	Coding	Unit Testing	Integration Testing	System Testing	Acceptance Testing			
Requirements	0	7	6	3	0	0	12	28	56	17	3.294117647
High-Level		0	8	8	3	8	30	6	63	22	2.863636364
Detailed Design			0	13	6	12	20	0	51	25	2.04
Coding				0	63	48	111	48	270	136	1.985294118
Summary	0	7	14	24	72	68	173	82	440	200	2.2

Table: Number of defects weighted by defect age on project *Boomerang*

Spoilage Metric

- The spoilage value for the Boomerang test project is 2.2
- A spoilage value close to 1 is an indication of a more effective defect discovery process
- This metric is useful in measuring the long-term trend of test effectiveness in an organization

Measuring Test completeness

Refer to how much of code and requirements are covered by the test set.

<u>The relationship between code coverage and the number of test cases:</u>

$$C(x) = 1 - e^{-(p/N)*x}$$

C(x) is coverage after executing x number of test cases, N is the number of blocks in the program and p is the average of number of blocks covered by a test case during the function test.

- Requirement traceability matrix

Quality

Effectiveness of smoke tests(establish confidence over stability of a system)

SMOKE TESTING, also known as “Build Verification Testing”, is a type of software testing that comprises of a non-exhaustive set of tests that aim at ensuring that the most important functions work. The result of this testing is used to decide if a build is stable enough to proceed with further testing.

Quality of Test Plan : The quality of test plan is measured in concern with the possible number of errors.

– Multidimensional qualitative method using rubrics

Attributes : Size

- Number of test cases reused
- Number of test cases added to test database
- Number of test cases rerun when changes are made to the S/W
- Number of planned regression tests executed
- Number of planned regression tests executed and passed

Size	• Estimation of test cases • Number of regression tests • Tests to automate

Size Metrics

- **Line of Code (LOC)**
- **Token Count (Halstead Product Metrics) : counting the number of operators and operands.**

Program Vocabulary

$n = n1 + n2$

where n = program vocabulary

n1 = number of unique operators

n2 = number of unique operands

Program Length

$N = N1 + N2$

Where N = program length

N1 = all operators appearing in the implementation

N2 = all operands appearing in the implementation

Token Count

Program Volume : Refers to the size of the program.

$V = N \log_2 n$

where V = Program volume

N = Program length

n = Program vocabulary

Function Point Analysis

Estimation models for estimating testing efforts

1) Halstead metrics for estimating testing effort

- PL = 1/ [(n1 / 2) * (N2 / n2)]
- e(effort) = V/PL
- V= describe number of volume of information in bits required to specify a program.
- PL= measure of software complexity
- Percentage of testing effort (k) for module k = e(k) / ∑e(i)

e(k) : effort required for module k

∑e(i): sum of halstead effort across all modules of the system

2) Development Ratio Method(No. of testing personal required is estimated on the basis of developer – tester ratio)

- Type and complexity of the software being developed
- Testing level
- Scope of testing
- Test effectiveness
- Error tolerance level for testing
- Available budget

- Heuristics Method : Developer –efficiency

Estimation models for estimating testing efforts

3)Project staff ratio method

Project type	Total number of project staff	Test team size %	Number of testers
Embedded system	100	23	23
Application development	100	8	8

4)Test procedure method

	Number of test procedures (NTP)	Number of person-hours consumed for testing (PH)	Number of hours per test procedure = PH/ NTP	Total period in which testing is to be done (TP)	Number of testers = PH/TP
Historical Average Record	840	6000	7.14	10 months (1600 hrs)	3.7
New Project Estimate	1000	7140	7.14	1856 hrs	3.8

Estimation models for estimating testing efforts

5)Task planning method

Number of Test Procedures (NTP)	Person-hours consumed for testing (PH)	Hours per test procedure = PH/NTP
840	6000	7.14
1000 (New project)	7140	7.14

Testing activity	Historical value	% time of the project consumed on the test activity	Preliminary estimate of person hours	Adjusted estimate of person-hours
Test planning	210	3.5	249	
Test design	150	2.5	178	
Test execution	180	3	214	
Project total	6000	100%	7140	6900

	NTP	PH (Adjusted estimate)	Number of hours per test procedure = PH/NTP	TP	Number of testers = PH/TP
New project estimate	1000	6900	6.9	1856 hrs	3.7

Architectural Design Metric used for Testing

Structural Complexity

$S(m) = f_{out}^{2}(m)$

where S is the structural complexity of a module m and fout(m) is the fan-out of module m.

This metric gives us the number of stubs required for unit testing of the module m.(Unit Testing)

Data Complexity

$D(m) = v(m) / [f_{out}(m) + 1]$

where v(m) is the number of input and output variables that are passed to and from module m.

This metric measures the complexity in the internal interface for a module m and indicates the probability of errors in module m.

System Complexity

$SC(m) = S(m) + D(m)$

It is defined as the sum of structural and data complexity.

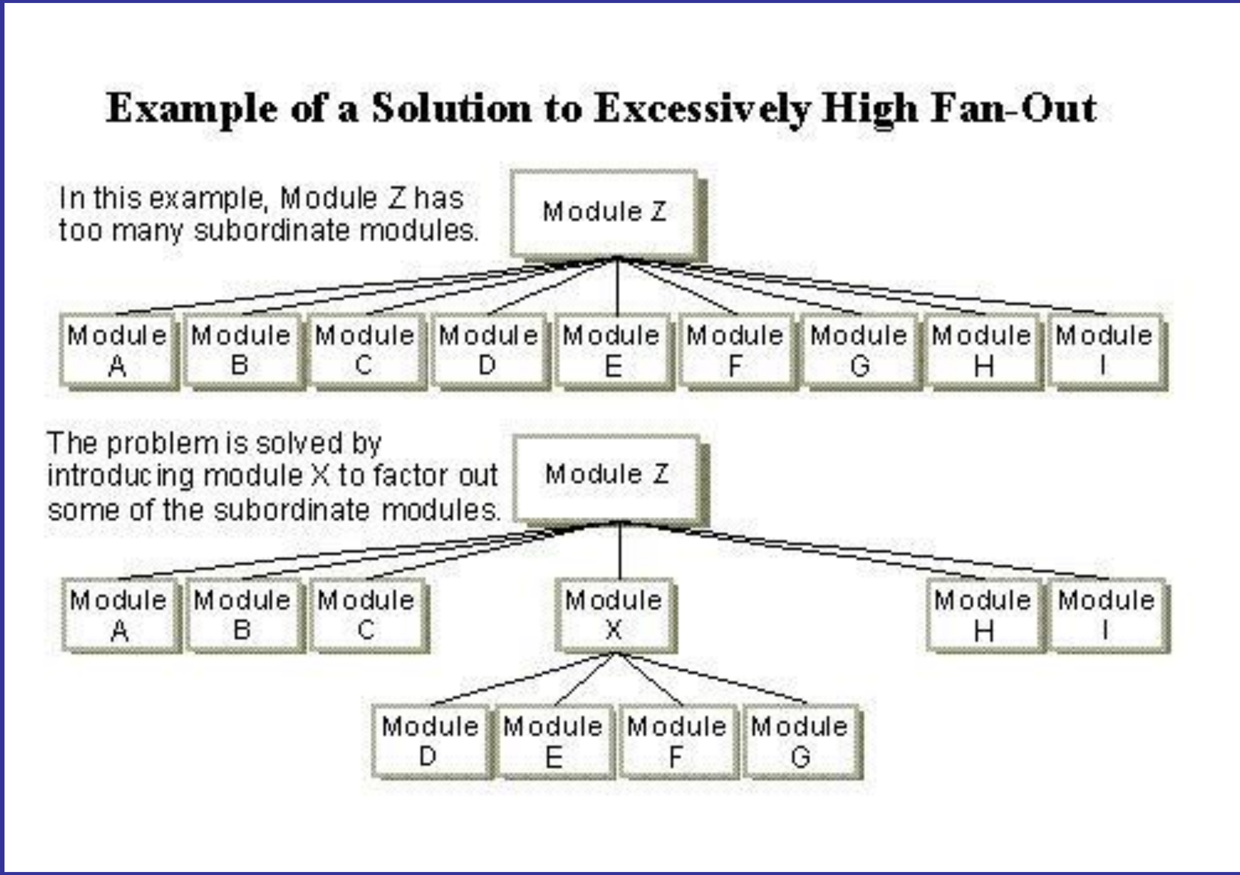
Overall architectural complexity of system is the sum total of system complexities of all the modules.

Efforts required for integration testing increases with the architectural complexity of the system.

Information Flow Metrics used for Testing

- **Local Direct Flow**
- **Local InDirect Flow**
- **Global Flow**
- **Fan-in** of a module
- **Fan-out** of a module

Information Flow Metrics used for Testing



Information Flow Metrics used for Testing :

Henry & Kafura Design Metric

Measure the total level of information flow between individual modules and rest of the system

IFC(m) = length(m) x $((fan - in(m)\ x\ fan - out(m))^2$

The higher the IF complexity for m, greater is the effort in integration and integration testing, thereby increasing the probability of errors in the module.

Cyclomatic Complexity Measures for Testing

- Since cyclomatic number measures the number of linearly independent paths through flow graphs, it can be used as the set of minimum number of test cases.

- McCabe has suggested that ideally, cyclomatic number should be less than equal to 10. This number provides a quantitative measure of testing difficulty. If cyclomatic number is more than 10, then testing effort increases due to :

  Number of errors increases.

  Time required to find and correct the errors increases

Pooja Malhotra

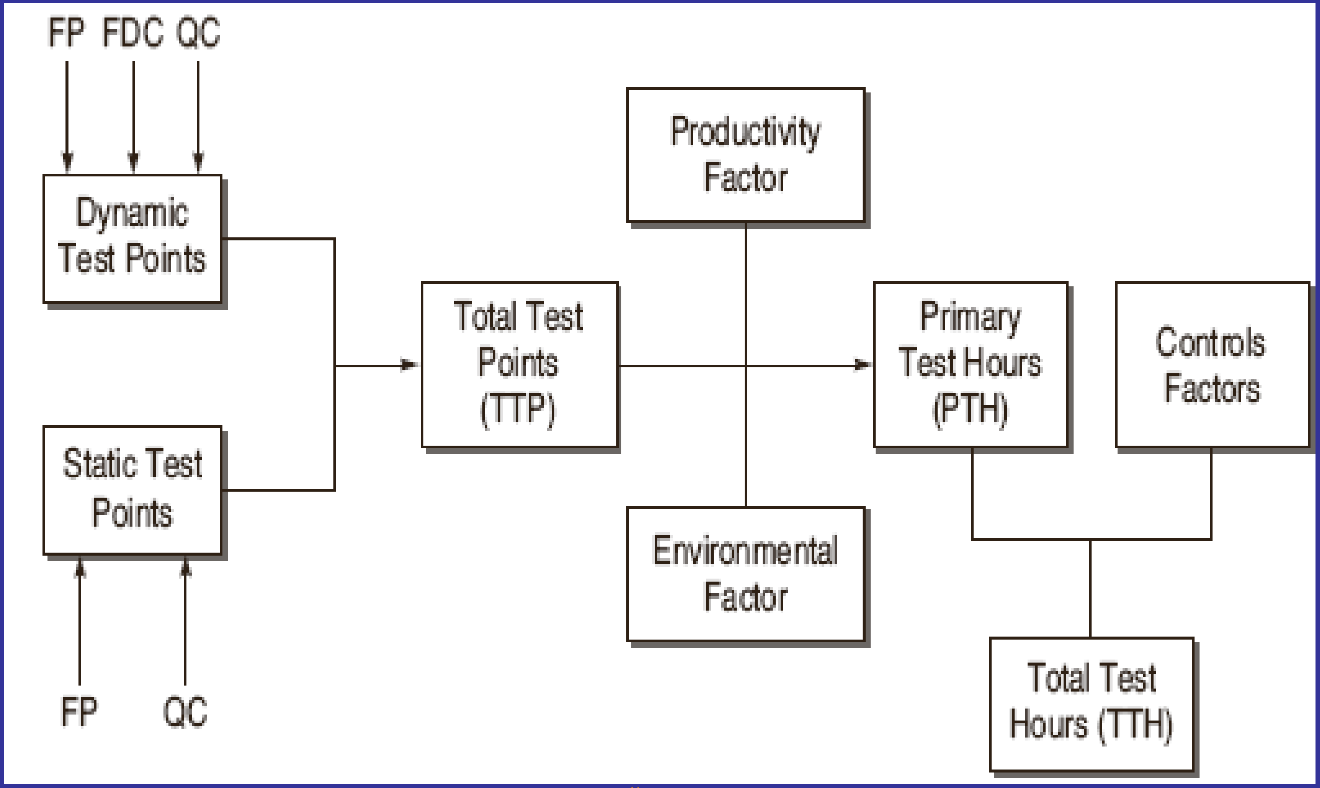
Function Point Metrics for Testing

Used for measuring size of a software, Testing effort

- Number of Hours required for testing per Function Point (FP)
- Number of FPs tested per person-months
- Total cost of testing per FP
- Defect density =*Number of defects (by phase or in total) / Total number of FPs*
- Test case Coverage =*Number of Test cases / Total number of FPs*
- Number of Test Cases = $(\text{Function Points})^{1.2}$
- Number of Acceptance Test Cases = (Function Points) x 1.2

Test Point Analysis:Black Box test effort estimaion

TPA calculates the test effort estimation in test points for important functions.



Test Point Analysis

- **Calculating Dynamic Test Points**

  DTP = FP x FDCw x QCdw

Where DTP = number of dynamic test points

FP = Function point assigned to function

FDCw = Function dependent factor wherein weights are assigned to function dependent factors

QCdw = Quality characteristic factor wherein weights are assigned to dynamic quality characteristics

Test Point Analysis

FDCw = ((FIw + UINw + I + C) /20) x U

Where FIw = Function importance rated by user

UINw = Weights given to Usage intensity of the function, i.e. how frequently function is being used

I = Weights given to the function for interfacing with other functions, i.e. if there is change in function, how many functions in the system will be affected

C = Weights given to complexity of function, i.e. how many conditions are in the algorithm of function
U = Uniformity factor

Test Point Analysis

QCdw = ∑ (rating of QC / 4) x weight factor of QC

Table 11.8 Ratings for FDC_w factors

Factor/ Rating	Function Importance (FI)	Function usage Intensity (UIN)	Interfacing (I)	Complexity (C)	Uniformity Factor
Low	3	2	2	3	0.6 for the function, wherein test specifications are largely re-used such as in clone function or dummy function. Otherwise, it is 1.
Normal	6	4	4	6	
High	12	12	8	12	

Table 11.9 Ratings and weights for QC_{dw}

Characteristic/ Rating	Not Important (0)	Relatively Unimportant (3)	Medium Importance (4)	Very Important (5)	Extremely Important (6)
Suitability	0.75	0.75	0.75	0.75	0.75
Security	0.05	0.05	0.05	0.05	0.05
Usability	0.10	0.10	0.10	0.10	0.10
Efficiency	0.10	0.10	0.10	0.10	0.10

Test Point Analysis

- **Calculating Static Test Points**
  - **STP = FP x ∑QCsw / 500**

  Where STP = Static Test Point

  FP = Total function point assigned to system
- QCsw = Quality characteristic factor wherein weights are assigned to static quality characteristics
- Static quality characteristic is what can be tested with a checklist.
- QCsw is assigned the value 16 for each quality characteristic which can be tested statically using the check list.
- **Total Test Points (TTP) = DTP + STP**

Test Point Analysis

Calculating Primary Test Hours

- **Primary Test Hours (PTH) = TTP x Productivity factor x Environmental factor**

- **Productivity Factor** ranges between 0.7 to 2.0.
  - Skill set,knowledge,experience of tester

- **Environmental factor = weights of (Test Tools +**
- **Development Testing +**
- **Test Basis +**
- **Development Environment +**
- **Testing Environment +**
- **Testware) / 21**

Test Point Analysis

Table 11.10 Test tool ratings

1	Highly automated test tools are used.
2	Normal automated test tools are used.
4	No test tools are used.

Table 11.11 Development testing ratings

2	Development test plan is available and test team is aware about the test cases and their results.
4	Development test plan is available.
8	No development test plan is available.

Table 11.12 Test basis rating

3	Verification as well as validation documentation are available.
6	Validation documentation is available.
12	Documentation is not developed according to standards.

Table 11.13 Development environment rating

2	Development using recent platform.
4	Development using recent and old platform.
8	Development using old platform.

Table 11.14 Test environment rating

1	Test platform has been used many times.
2	Test platform is new but similar to others already in use.
4	Test platform is new.

Table 11.15 Testware rating

1	Testware is available along with detailed test cases.
2	Testware is available without test cases.
4	No testware is available.

Test Point Analysis

Calculating Total Test Hours

Total Test Hour = PTH + Planning and control allowance

- **Planning & Control Allowance (%)= weights of (Team size + Planning and Control Tools)**
- **Planning & Control Allowance (hours)= Planning & Control Allowance (%) x PTH**

Testing Progress Metrics

- Test Procedure Execution Status:

Test proc Exec. Status=Number of executed test cases/Total number of test cases

- Defect Aging :Turnaround time for a bug to be corrected.

Defect aging = closing date of bug - start date when bug was opened

- Defect Fix Time to Retest:

Defect Fix Time to Retest = Closing date of bug and releasing in new build – Date of retesting the bug

- Defect Trend Analysis: defined as the trend in the number of defects found as testing progresses.
  - Number of defects of each type detected in unit test per hour
  - Number of defects of each type detected in integration test per hour
  - Severity level for all defects

Testing Progress Metrics

- <u>Recurrence Ratio</u> : Indicates the quality of bug –fixes.

Number of bugs remaining per fix.

- <u>Defect Density</u>:

1. Defect Density = Total number of defects found for a requirement /Total number of test cases executed for that requirement
2. Pre- ship defect density/Post-ship defect density

- <u>Coverage Measures</u>: Helps in identifying the work to be done.

<u>White-box testing</u>

Degree of statement , branch , data flow and basis path coverage

Actual degree of coverage/Planned degree of coverage

<u>Black-box Testing</u>

Number of features or Ecs actually covered/ Total number of features or Ecs.

Testing Progress Metrics

Tester Productivity:

1. Time spent in test planning
2. Time spent in test case design
3. Time spent in test execution
4. Time spent in test reporting
5. Number of test cases developed
6. Number of test cases executed

Testing Progress Metrics

- <u>Budget and Resource Monitoring Measures:</u>

Earned value tracking

For the planned earned values, we need the following measurement data :

1. Total estimated time or cost for overall testing effort
2. Estimated time or cost for each testing activity
3. Actual time or cost for each testing activity

Estimated time or cost for testing activity / Actual time or cost of testing activity

- <u>Test case effectiveness Metric</u>:

TCE= 100*(Number of defects found by the test cases/ total number of defects)