Chapter 11
Testing Metrics for Monitoring and Controlling the Testing Process

OXFORD
HIGHER EDUCATION

Objectives

- Testing metrics are important to monitor and control all testing activities.
- Attributes for software testing on the basis of which testing metrics are designed:
- Progress metrics
- Quality metrics
- Cost metrics
- Size metrics
- Testing Effort estimation models

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

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 removal efficiency
- 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

Estimation models for estimating testing efforts

OXFORD
HIGHER EDUCATION

Halstead metrics for estimating testing effort

- **PL = 1/ [(n1 / 2) * (N2 / n2)]**
- **e = V/PL**
- **Percentage of testing effort (k) = e(k) / ∑e(i)**

V is program volume
PL iis Program level
e(k)=effort for module k
∑e(i)=effort across all module of the system

- **Project staff ratio method:**
- Calculating the percentage of testing personnel from the overall allocate resources planned for the project

Estimation models for estimating testing efforts

OXFORD
HIGHER EDUCATION

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

Test procedure method

The no. of test procedures decides the number of tester required and testing time.

	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

OXFORD
HIGHER EDUCATION

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

Calculating number of person hours

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

Task planning method

	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

Team size using task planning method

Architectural Design Metric used for Testing

Structural Complexity

S(m) = f2out(m)
where S is the structural complexity of a module m
and fout(m) is the fan-out of module m.

Data Complexity
D(m) = v(m) / [fout(m) + 1]
where v(m) is the number of input and output variables that are passed to and from module m.

System Complexity
SC(m) = S(m) + D(m)

Information Flow Metrics used for Testing

OXFORD
HIGHER EDUCATION

- Local Direct Flow:
- (i) module invokes a second module and passes information to it
- (ii) the invoked module returns a result to the caller

- Local InDirect Flow: exists if the invoked module returns information that is subsequently passed to a second invoked module.

- Global Flow :flow from one module to another module via a global data structure.

- **Fan-in** of a module :

No. of local flows that terminates at m,

Plus the no. of data structures from which information is retreived by m

- **Fan-out** of a module :

No. of local flows that emantate from m,plus no. of data structures that are updated by m

Henry & Kafura Design Metric

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

OXFORD
HIGHER EDUCATION

- 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

Function Point Metrics for Testing

- 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 = (Function Points)$^{1.2}$
- Number of Test Cases = (Function Points) x 1.2

Test Point Analysis

- Dyanamic and static points calculated
- Dyanamic function point

a. FP

b. FDC

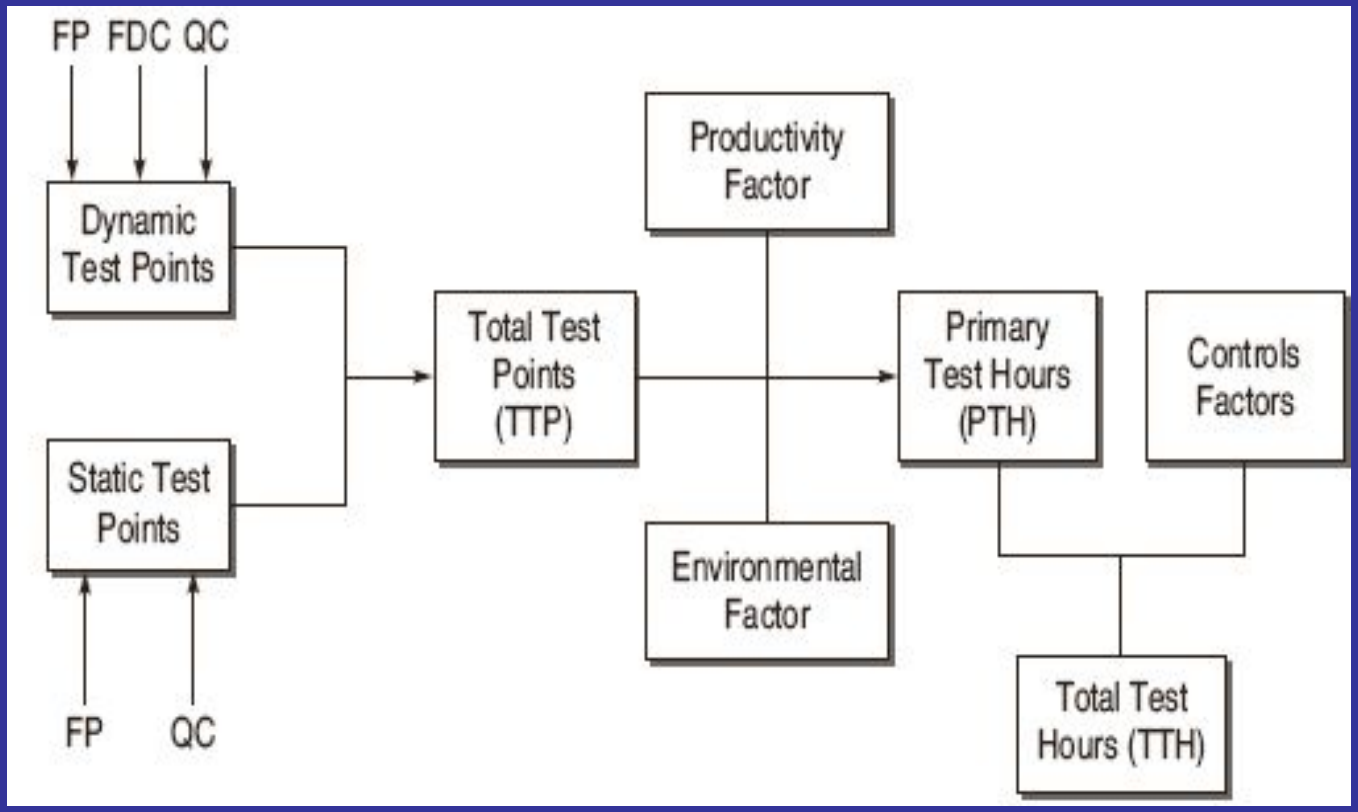
c. QC

- Static test point

a. FP

b. QC

Test Point Analysis



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

- QCdw=Quality Characterstics
- Suitability
- Security
- Usability
- effieciency

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.

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

Test Point Analysis

OXFORD
HIGHER EDUCATION

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

OXFORD
HIGHER EDUCATION

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**

Some Testing Metrics

Test Procedure Execution Status
= *Number of Executed test cases / Total number of test cases*

Defect Aging = Closing date of bug – Start date when bug was opened

Defect Fix Time to Retest = Date of fixing the bug and releasing in new build – Date of Retesting the bug

Defect Density = Total number of defects found for a requirement / Number of test cases executed for that requirement

Some Testing Metrics

OXFORD
HIGHER EDUCATION

Tester Productivity measures

- Time spent in test planning
- Time spent in test case design
- Time spent in test execution
- Time spent in test reporting
- Number of test cases developed
- Number of test cases executed.

Test Case Effectiveness metric

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