SWE 4603

Software Testing and Quality Assurance

Lecture 11

Prepared By Maliha Noushin Raida, Lecturer, CSE
Islamic University of Technology

Lesson Outcome

- Testing Metrics
- Progress metrics
- Quality metrics
- Cost metrics
- Size metrics
- Testing effort estimation model

Week on:

- Chapter 11: Testing Metrics for Monitoring and Controlling the Testing Process

Testing Metrics

Testing Metrics

- Software testing metrics are a way to measure and monitor your test activities.
- More importantly, they give insights into your team's test progress, productivity, and the quality of the system under test.
- A metric usually conveys a result or a prediction based off the combination of data.

Result Metrics: metrics that are mostly an absolute measure of an activity/process completed.
Example: Time taken to run a set of test cases in a suite

Predictive Metrics: metrics that are derivatives and act as early warning signs of an unfavorable result.
Example: Defects created vs. Resolved chart shows the rate of defect fixing. This grabs the team's attention if this rate is slower than the rate desired.

Why Testing Metrics

- The aim of collecting test metrics is to use the data for improving the test process.
- This includes finding tangible answers to the following questions:
  - How long will it take to test?
  - How bad are the bugs?
  - How many bugs found were fixed? reopened? closed? deferred?
  - How many bugs did the test team did not find?
  - How much of the software was tested?
  - How good were the tests? Are we using low-value test cases?
  - What is the cost of testing?
  - Was the test effort adequate? Could we have fit more testing in this release?

Proper **Measurement** answers all these questions.

Testing Metrics

An organization needs to have a reference set of measurable attributes and corresponding metrics that can be applied at various stages during the course of execution of an organization-wide testing strategy.

Measurable attributes of software testing are:

- Progress
- Cost
- Quality
- Size
- Effort

Progress Metrics

Tracking test progress: The progress of testing should also be measured to keep in line with the schedule, budget, and resources.

For measuring the test progress, well-planned test plans are prepared wherein testing milestones are planned. For example: executing all planned unit tests is one example of a testing milestone.

$$\text{Passed Test Cases Percentage} = \left(\frac{\text{Number of Passed Tests}}{\text{Total number of tests executed}}\right) \times 100$$

$$\text{Failed Test Cases Percentage} = \left(\frac{\text{Number of Failed Tests}}{\text{Total number of tests executed}}\right) \times 100$$

$$\text{Blocked Test Cases Percentage} = \left(\frac{\text{Number of Blocked Tests}}{\text{Total number of tests executed}}\right) \times 100$$

$$\text{Average time for development team to repair defects} = \left(\frac{\text{Total time taken for bug fixes}}{\text{Number of bugs}}\right)$$

Progress Metrics

Tracking test progress

$$\text{Accepted Defects Percentage} = \left(\frac{\text{Defects Accepted as Valid by Dev Team}}{\text{Total defects reported}}\right) \times 100$$

$$\text{Fixed Defects Percentage} = \left(\frac{\text{Defects Fixed}}{\text{Defects Reported}}\right) \times 100$$

$$\text{Defects Rejected Percentage} = \left(\frac{\text{Defects Rejected as invalid by Dev Team}}{\text{Total defects reported}}\right) \times 100$$

$$\text{Defects Deferred Percentage} = \left(\frac{\text{Defects deferred for future releases}}{\text{Total defects reported}}\right) \times 100$$

$$\text{Critical Defects Percentage} = \left(\frac{\text{Critical Defects}}{\text{Total defects reported}}\right) \times 100$$

Progress Metrics

S-Curve in Representation:

What is an S-Curve?

- An S-Curve is a graphical representation of cumulative work effort.
- S-Curves can be used to describe projects as a whole, development efforts, test efforts, as well as defect discovery rates.

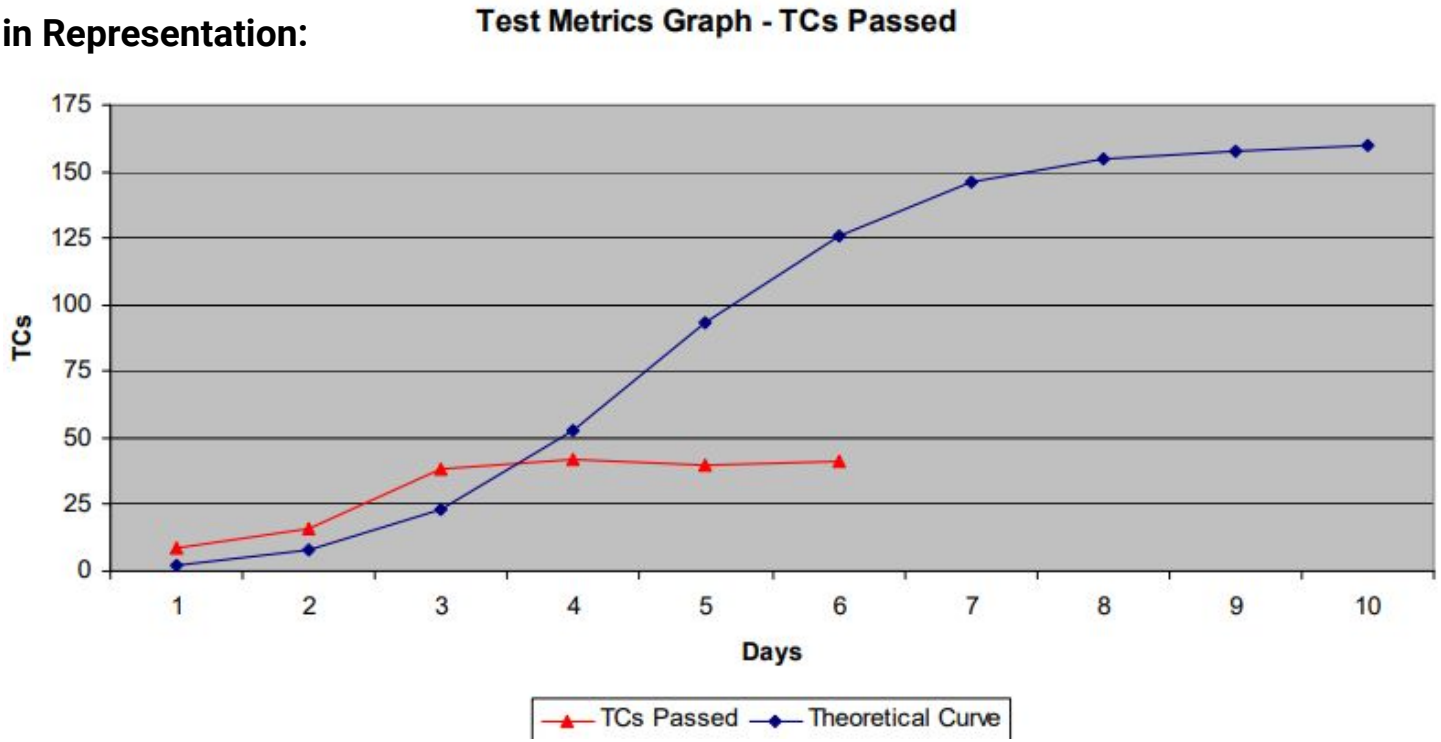
What makes it an "S" shape?

- Test efforts typically start out slowly as testers run into a few major defects
- As the initial issues are resolved, the testers are able to execute more tests covering more Functionality
- As the test effort nears its end, there are typically a few leftover issues that must be resolved thus slowing the process down again

How is it used?

- Measure test progress by comparing the actual test curve to a theoretical S-Curve
- Use the curve to determine if the application is stable enough to be released

S-Curve in Representation:



What are some potential causes associated with this S-Curve? How might you correct these issues?

Progress Metrics

Defect Backlog:

It is the number of defects that are outstanding and unresolved at one point of time with respect to the number of defects occurring. The defect backlog metric is to be measured for each release of a software product for release-to-release comparisons.

Staff productivity

The basic and useful measures of a tester's productivity are:

- Time spent in test planning.
- Time spent in test case design.
- Number of test cases developed.
- Number of test cases developed per unit time.

Cost Metrics

Infrastructure and tools contribute to the expense of testing. Testing systems do not have fathomless financial resources to spend. Thus, estimating how much you can spend and how much you indeed wrap up spending is eventful.

Budget variance:

The variance between the actual and planned costs is referred to as budget variance.

Schedule variance:

It is the difference between the actual time taken to complete tests and the planned time.

Cost per bug fix:

It refers to the amount of effort spent on a defect per developer.

However, the cost of a bug fix is 10 * $60 = $600; if a developer spends 10 hours fixing a defect, their hourly rate is $60.

Quality Metrics

Effectiveness of test cases

$$\text{Number of bugs per test} = \frac{\text{Total number of defects}}{\text{Total number of tests}}$$

$$\text{Bug find rate or Number of defects per test hour} = \frac{\text{Total number of defects}}{\text{Total number of test hours}}$$

$$\text{Defect-removal efficiency} = \left(\frac{\text{Bugs found in Test}}{\text{Total bugs found(bugs found in test + bugs found after shipping}}\right) \times 100$$

Context Based: Test Effectiveness Using Team Assessment

- You can ask your team to rate the test set for how good it is.
- Your team could also use a subjective scaling method. Out of a 100% rating (1 to 10 scale), ask your team to give a score to the test set as to how complete, up to date, and effective the test set stands today.
- Get an average on the score to get the team's perceived average test effectiveness.
- It is important to tell your team to be unbiased and to define what a good test set means.

Quality Metrics

Effectiveness of test cases

Defect spoilage depends on the defect age or the phase in which it is discovered.

$$\text{Spoilage} = \frac{\text{Sum of (Number of defects} \times \text{Defect age)}}{\text{Total number of defects}}$$

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

Generally, low values of defect spoilage mean more effective defect discovery processes.

SDLC phase	No. of defects	Defect age
Requirement Specs.	34	2
HLD	25	4
LLD	17	5
Coding	10	6

Solution

$$\text{Spoilage} = (34 \times 2 + 25 \times 4 + 17 \times 5 + 10 \times 6)/86 = 3.64$$

Effort Metrics

Halstead Metrics

Halstead developed expressions for program volume V and program level PL which can be used to estimate testing efforts.

$$PL = \frac{1}{(n1/2) * (N2/n2)}$$

$$e = \frac{V}{PL}$$

where , V= Halstead Volume = $Nlog_2n$

See Example 11.2 page 328

Effort Metrics

Function points

- The function point (FP) metric is used effectively for measuring the size of a software system.
- Function-based metrics can be used as a predictor for the overall testing effort.

Some derived metrics from FP

Defect Density = Number of defects (by phase or in total) / Total number of FPs

Test Coverage = Number of test cases / Total number of FPs

Number of total test cases = (function points)$^{1.2}$

Number of acceptance test cases = (function points) × 1.2

Effort Metrics

Example from 11.3 Page 334

In a project, the estimated function points are 760. Calculate the total number of test cases in the system and the number of test cases in acceptance testing. Also, calculate the defect density (number of total defects is 456) and test case coverage.

Solution

Total number of test cases = $(760)^{1.2}$ = 2864
Number of test cases for acceptance testing = (760) × 1.2 = 912
Defect density = 456/760 = 0.6
Test case coverage = 2864/760 = 3.768

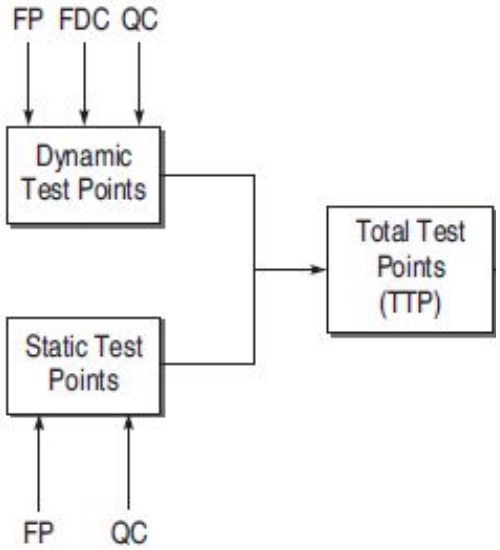
Effort Metrics (Test Point Analysis)

- Test point analysis is a technique to measure the black-box test effort estimation.
- The estimation is for system and acceptance testing.
- The technique uses function point analysis in the estimation.
- TPA calculates the **test effort** estimation in test points for highly important functions, according to the user and also for the whole system.

Test Point Analysis

PROCEDURE FOR CALCULATING TPA:

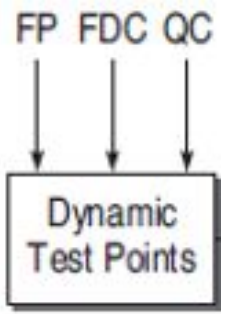
- First the dynamic test points
- Then static test points are calculated
- dynamic test point is added to the static test point to get the **total test points (TTP)** for the system.



Test Point Analysis

Dynamic Test Point

- Dynamic test points are the number of test points which are based on dynamic measurable quality characteristics of functions in the system.
- To calculate a dynamic function point, we need the following:
  - **Function points** (FPs) assigned to the function.
  - **Function dependent factors** (FDC), such as complexity, interfacing, function-importance, etc.
  - **Quality characteristics** (QC).
- The dynamic test points for individual functions are added and the resulting number is the dynamic test point for the system.



Test Point Analysis

Dynamic Test Point

The number of dynamic test points for each function in the system is calculated as

$$DTP = FP \times FDC_w \times QC_{dw}$$

where,

DTP = number of dynamic test points
FP = **function point** assigned to function.
FDC_w = weight-assigned **function-dependent factors**
QC_{dw} = **quality characteristic** factor, wherein weights are assigned to dynamic quality characteristics

Test Point Analysis

Dynamic Test Point

FDC_w is calculated as

$$FDC_w = ((FI_w + UIN_w + I + C) \div 20) \times U$$

where

FI_w = function importance rated by users

UIN_w = weights given to usage intensity of the function, i.e. how frequently the function is being used

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

C = weights given to the complexity of function, i.e. how many conditions are in the algorithm of function

U = uniformity factor

Test Point Analysis

Dynamic Test Point

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	

Test Point Analysis

Dynamic Test Point

- QC_{dw} is calculated based on four dynamic quality characteristics, namely **functionality, security, usability, and efficiency.**
- First, the rating to every quality characteristic is given and then, a weight to every QC is provided.
- Based on the rating and weights, QC_{dw} is calculated.

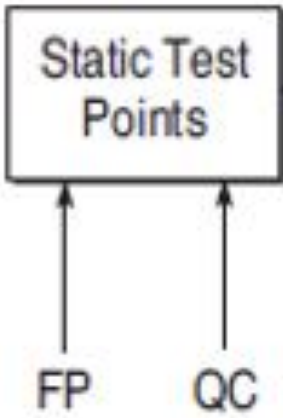
$$QC_{dw} = (QC_f*0.75+QC_S*0.05+QC_u*0.1+QC_e*0.1)/4$$

QC_f= Functionality Factor
QC_s= Security Factor
QC_u= Usability Factor
QC_e= Efficiency Factor

	Not Important	Relatively Unimportant	Medium Importance	Very Important	Extremely Important
Rating	0	3	4	5	6

Test Point Analysis

- **static test points** are the number of test points which are based on static quality characteristics of the system.
- It is calculated based on the following:
  - Function points (FPs) assigned to the system.
  - Quality requirements or test strategy for static quality characteristics (QC).



Test Point Analysis

The number of static test points for the system is calculated as

$$STP = FP \times (\Sigma QC_{sw} / 500)$$

where

STP = static test point
FP = total function point assigned to the system
QC_{sw} = quality characteristic factor, wherein weights are assigned to static quality characteristics

- Static quality characteristic refers to what can be tested with a checklist.
- QC_{sw} is assigned the value **16** for each of 6 ***ISO 9126*** Quality characteristics which can be tested statically using the checklist.

$$\text{Total test points (TTP)} = DTP + STP$$

Test Point Analysis

Example

Calculate the total test points for a module whose specifications are: **functions points** = 414, ratings for all **FDC_w** factors are normal, **uniformity factor** =1, rating for all **QC_{dw}** are ‘very important’ and for **QC_{sw}**, three static qualities are considered.

Solution See in page 341 Example 11.4