TEST METRICS
and
TEST COVERAGE WORKSHOP
Version 1.0

A World of Difference

By **Son Pham**

June, 2015

Duration: 3 hours

www.globalcybersoft.com

CMMI Level 4

TÜV SÜD ISO 9001

Microsoft GOLD CERTIFIED Partner

Contents

- Test metrics
  - What is test metric?
  - Metric Lifecycle
  - How metrics are defined and evaluated?
  - How measurement value can be visualized?
  - Types of test metrics
  - The elements of bad metrics
- Test coverage
  - What is test coverage?
  - How to measure test coverage?
  - Types of test coverage
- Exercises

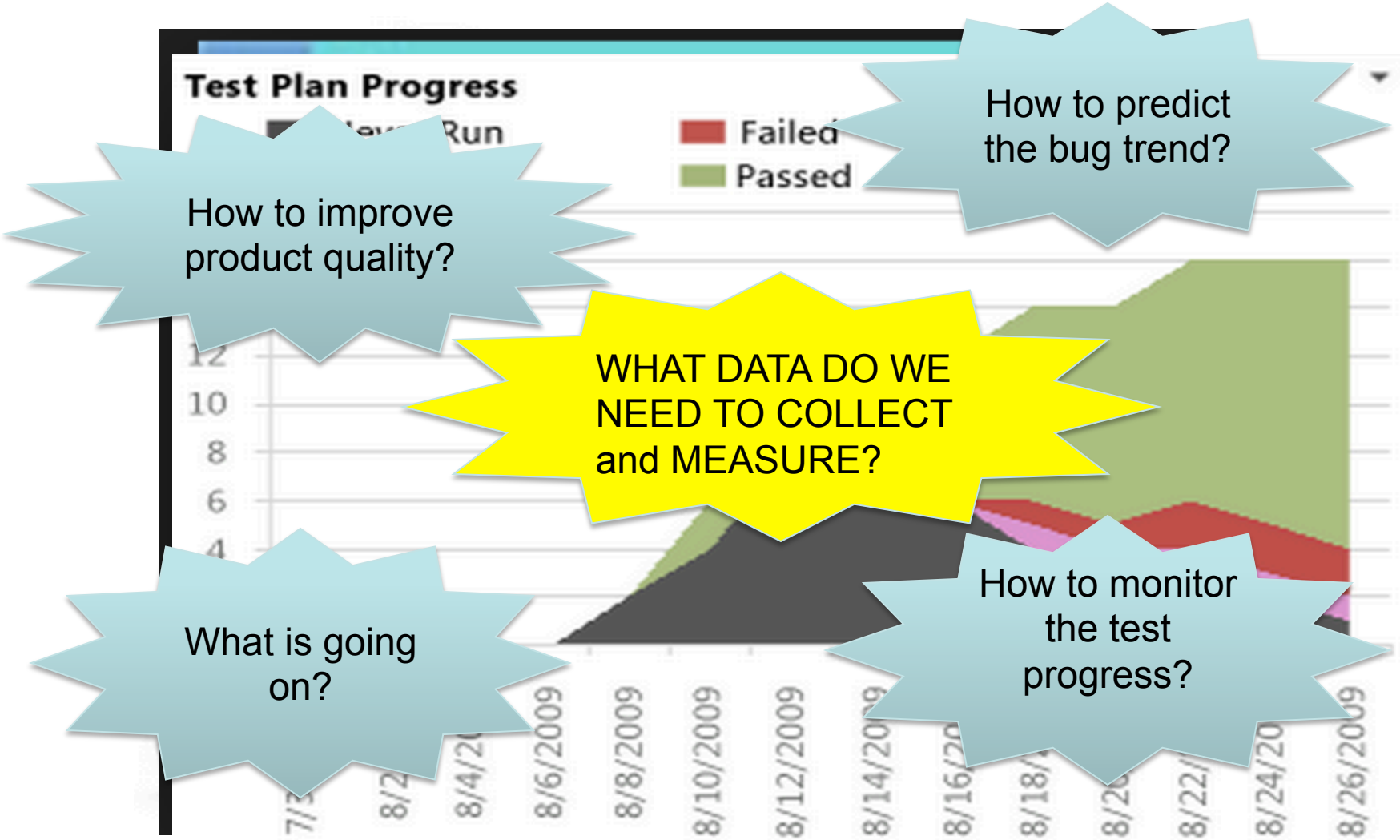
Glossary

- UC: Use Case
- TCs: Test cases
- TSC: Test Scenario
- DDP: Detect Defect Percentage
- DRE: Defect Removal Efficiency
- IDE: Interface Development Environment
- KLOC: Kilo Line of Code

TEST METRICS

- What is test metric?
- Metric Lifecycle
- How metrics are defined and evaluated?
- How measurement value can be visualized?
- Types of test metrics
- The element of bad metrics

Collect Data



What is test metric?

What is test metric?

- Test metrics allow quantitative statements regarding:
  - The product quality
  - Productivity
  - Test process
- Purpose of metrics:
  - Collect data to improve or maintain product quality, process...
  - Are used for comparison or prediction
- Metrics must built on solid measurement theoretic foundations
- Test Metrics form the basis for transparent, reproducible and traceable test process planning and control.
- Metric is used for both *measurement techniques* and *measurement values* used to measure certain features of the *measurement objects*.

Measurement objects

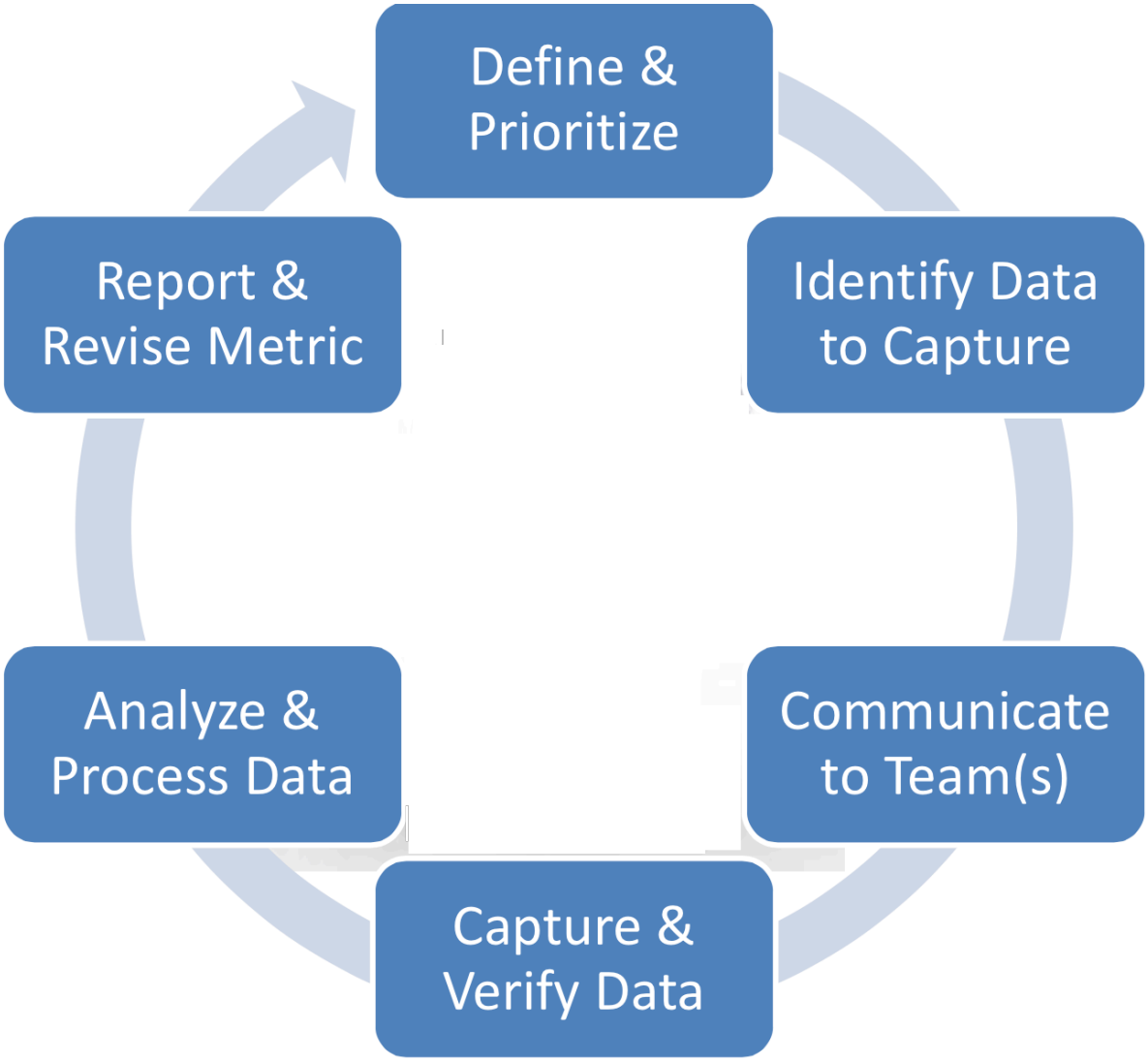
- Measurement objects in test management
  - Test basic document
    - Requirement specifications
    - Design specifications
  - Test Objects (concrete product)
  - Test case specifications
  - Test scripts
  - Defects
  - Test reports
  - Test Progress

Measurement objects

- Factors/ attributes of a measurement object can be measured:
  - Quality (ISO 9126)
  - Time
  - Resources/Effort
  - …

  E.g:
  - Attributes of test process:
    - Planned effort, Actual effort --> Effort Variance
    - Planned No. of Date, Actual No. of Date → Schedule Variance

Metric Lifecycle



How metrics are defined and evaluated?

Define and select suitable metrics

1. Identify fundamental objectives need to achieve and track through measurement
   - Major objectives
     - Quality (Based on ISO 9126)
     - Productivity
     - Test process

2. Identify concrete measurement objects and measurable features
   Ex: Defects, test cases…

3. Select suitable metrics

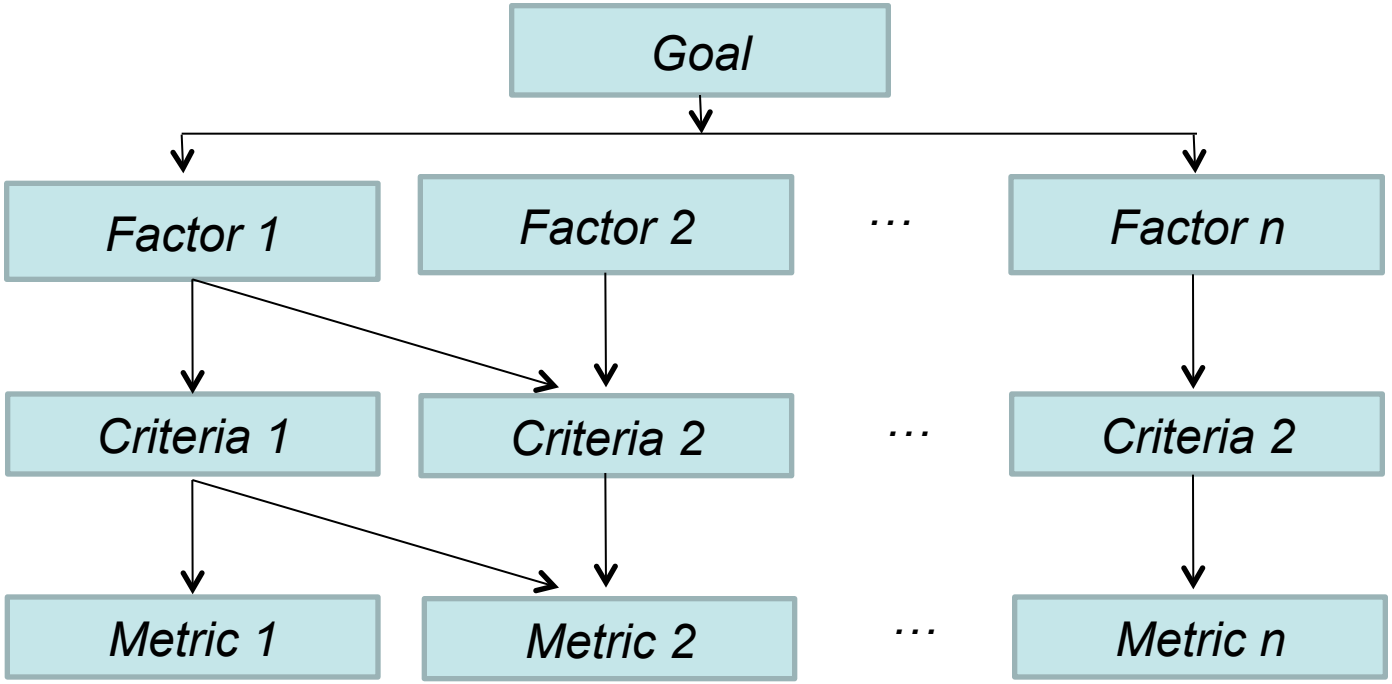
Note:

- For each test level, suitable test metrics need to be defined and collected.

`When a measure becomes a target, it ceases to be a good measure

Define and select suitable metrics – FCM method

- "Factor – Criteria – Metric" method: based on Quality model of ISO 9126

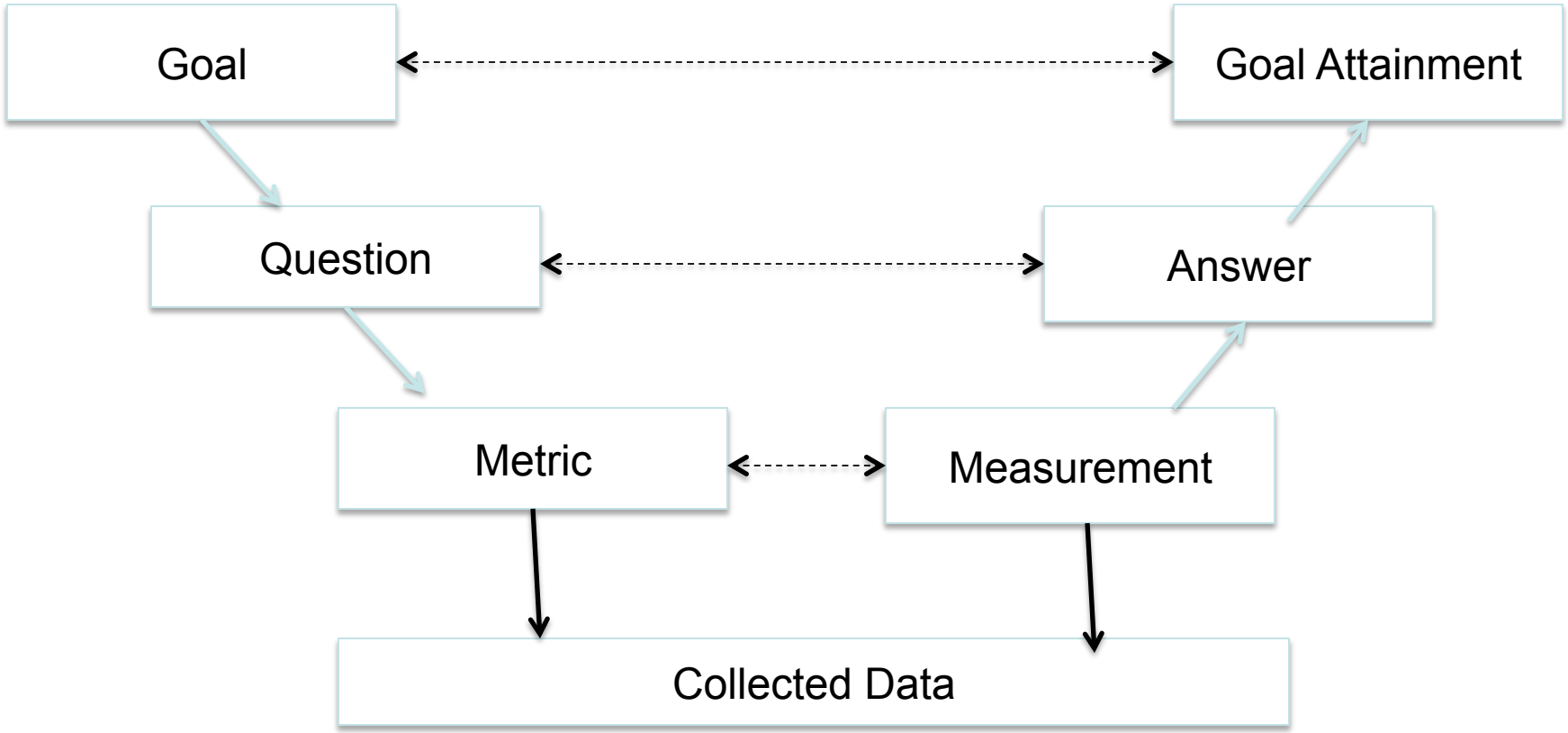


- Each criterion could be associated with metrics in the form of questions allowing subjective "yes" or "no" answers

Exercise

Define and select suitable metrics – GQM method

- "Goal – Question - Metric" method



Exercise

What is a good metric?

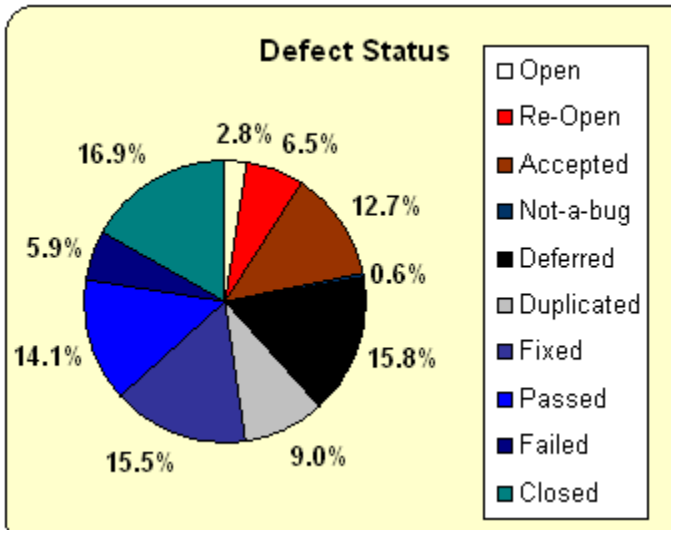
- Valid and easily collected
- Objective
- Simple and intuitive
- Robust
- Timeliness

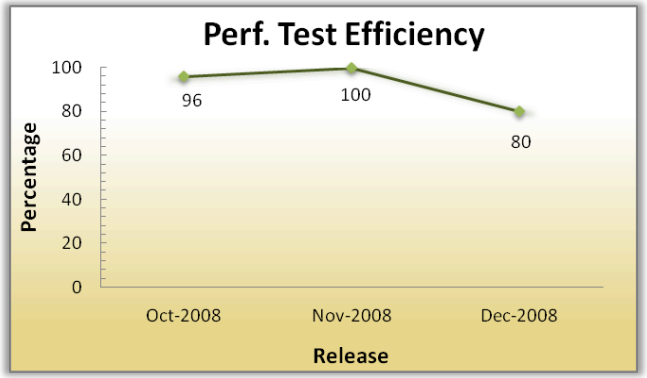
How measurement value can be visualized?

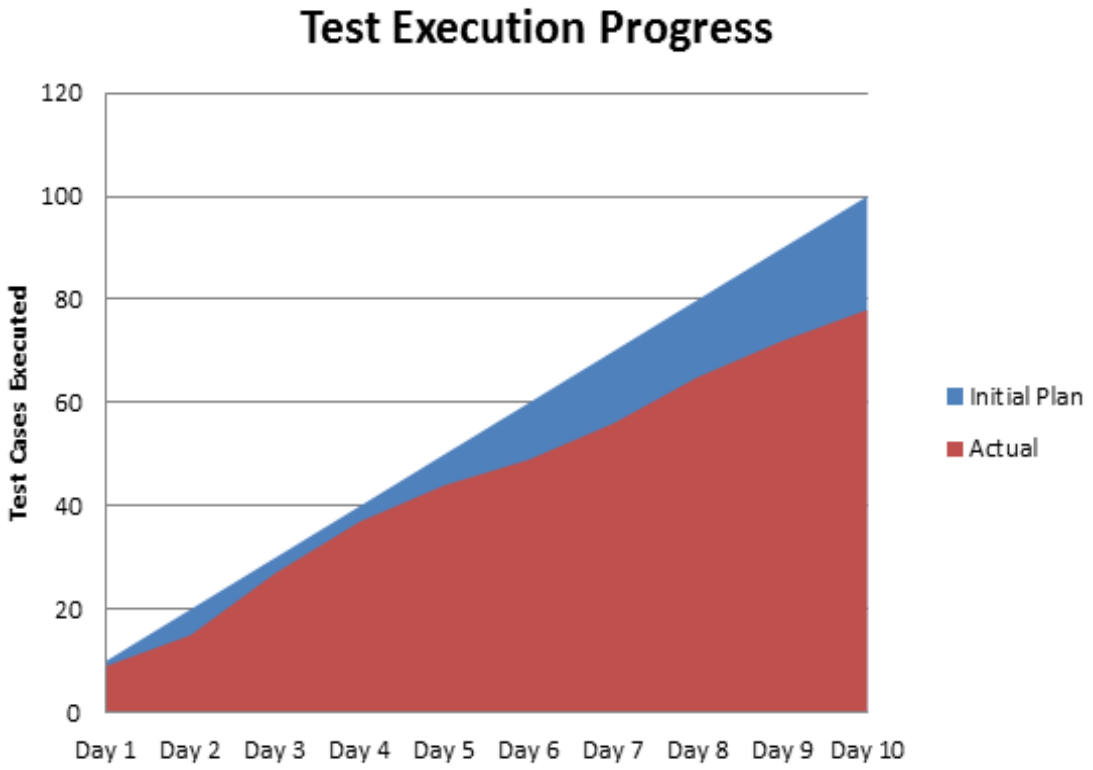
How measurement value can be visualized?

- Using chart or diagram to display measured values and their progression:
  - Bar or column chart: show metric values for several measurement objects
  - Line chart: show the chronological progression of a measurement object or of one or several metrics.
  - Circle or pie chart: present portion of several measurement values or measurement values or measurement objects from a certain total size onward.
  - Cumulative chart: show the sum of several measurement values in their chronological progression in the form of stacked line charts.

How measurement value can be visualized? - Example







Why the gap continues to increase over time?

Measurement Guidelines

- Establish your current baseline
- Focus on fundamentals (e.g. size, effort, time quality, etc.)
- Implement metrics as a service
- Pilot
- Take small steps
- Ask the engineers what metrics they need…
- Tailor metrics for your project
- Validate each metric with at least one other metric
- Remember that all metrics are not forever….
- KISS

Types of test metrics

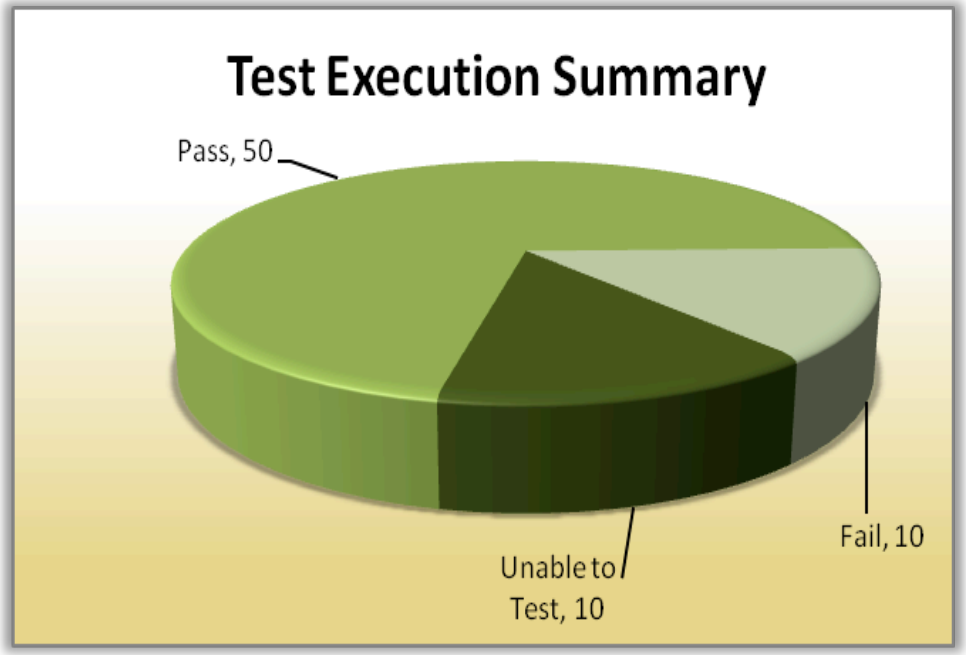
- Test-case based metrics
- Test-basic and test-object-based metrics
- Defect-based metrics
- Cost or effort-based metrics

Types of Metrics

- Based on the measurement objects, we get 4 types of metric as below:
  - **Test-case based metrics**
  - Test-basic and test-object-based metrics
  - Defect-based metrics
  - Cost or effort-based metrics

Test-case-based metrics

- *Purpose*: focus on a large number of test cases and their respective state to **track test progress**.
- *Can apply at all test levels*
- *Some metrics:*
  - Metrics concern in test execution:
    - % of executed TCs
    - % of passed TCs
    - % of failed TCs
    - % of non-executed TCs



Test-case-based metrics

- *Some metrics (cont)*
  - Metrics concern in test development
    - Test case density (TCs/KLOC)
  - Metrics concern Requirement changes:
    - Number of unplanned new TCs
    - Number of changed TCs
    - Number of deleted TCs

Types of Metrics

- Based on the measurement objects, we get 4 types of metric as below:
  - Test-case based metrics
  - **Test-basic and test-object-based metrics**
  - Defect-based metrics
  - Cost or effort-based metrics

Test-basic and test-object-based metrics

- *Purpose*: aim at the features and coverage of the test basis and test object to **track test progress**.
- *Can apply at all test levels*

Some metrics:

***1.Unit Test*:**
refer to Code Coverage - *"How many code is covered by test?"*

- Structure-oriented **dynamic metrics**: required *instrumentation* of the program
  - % of statements that have been covered
  - % of branches in the control flow that were covered
  - % of condition in the control flow that were covered
  - % of path in the control flow that were covered
  - % of procedure/function calls that have been executed

Test-basic and test-object-based metrics (cont)

2. Integration Test: focus on interface metrics

"How many interfaces in the system are covered by test?"

- % of interfaces tested
  - # of tested interface/ Total # of interface in Basic Design
- % of interface usage tested
  - Matrix with 2 dimensions (Usage, Interface)
- % of interface parameters tested with test technique XYZ
  - Matrix with 3 dimensions (Usage, Interface, Data Input)

Test-basic and test-object-based metrics (cont)

3. System and Acceptance Test: focus on requirement-based metrics:

- Functional or requirement-based metrics:
  - % of all requirements covered by TCs
  - % of all use case scenarios (Basic and alternative flows) that have been executed by a test suite
  - Number of tested function/ total number of function

- Nonfunctional-requirement-based metrics:
  - Number of platforms covered by test
  - Number of performance requirement per platform covered by test

Test-basic and test-object-based metrics (cont)

3. System and Acceptance Test: focus on requirement-based metrics:

- Functional or requirement-based metrics:

Ex: Requirement Traceability Matrix

	Test Cases														
Objectives	**TC #1**	**TC #2**	**TC #3**	**TC #4**	**TC #5**	**TC #6**	**TC #7**	**TC #8**	**TC #9**	**TC #10**	**TC #11**	**TC #12**	**TC #13**	**TC #14**	**TC #15**
Req. 1	X														
Req. 2						X									
Req. 3		X		X											
Req. 4		X	X												
Feature 1							X								
Feature 2	X				X										
Feature 3					X										
Feature 4		X				X									

Global CyberSoft

Defect-Based Metrics

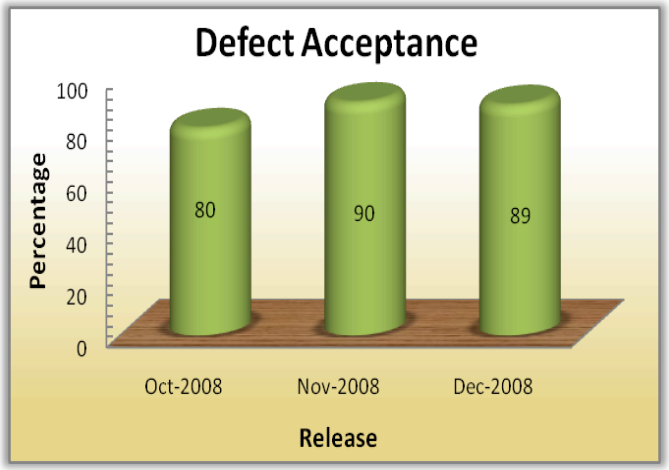
- *Purpose*: To evaluate test results and control test process and product quality.
- *Can apply at all test levels*

Some metrics:

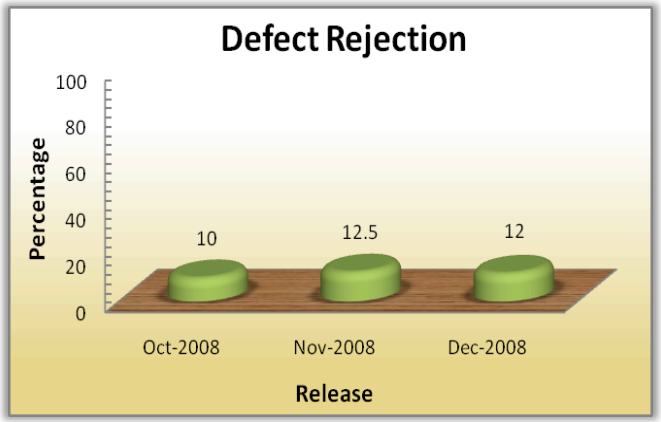
- Defect Density: # defects/ KLOC or # defects/pages
- → Defect distribution
- → Failure rate: # expected failures to be detected during execution time of a test or program
- → Fault days # days from defect injection to result failure

Defect-Based Metrics (cont)

- % defect found
  - Defect Acceptance (DA)
    - DA = (# of valid defects/Total reported defects) *100%



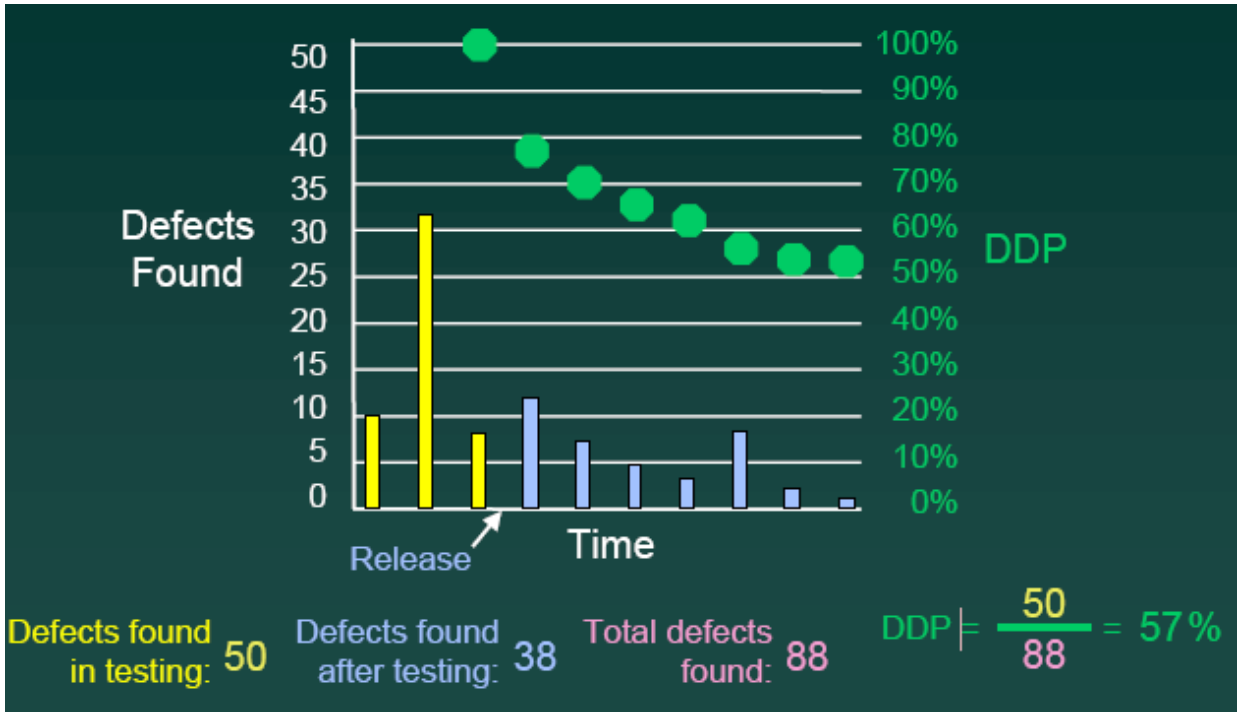
  - Defect Rejection (DR)
    - DR = (# of invalid defects/Total reported defects) * 100%



Defect-Based Metrics (cont)

- % defect found (Cont)
  - DDP (Detect Defect Percentage): → Test Efficiency

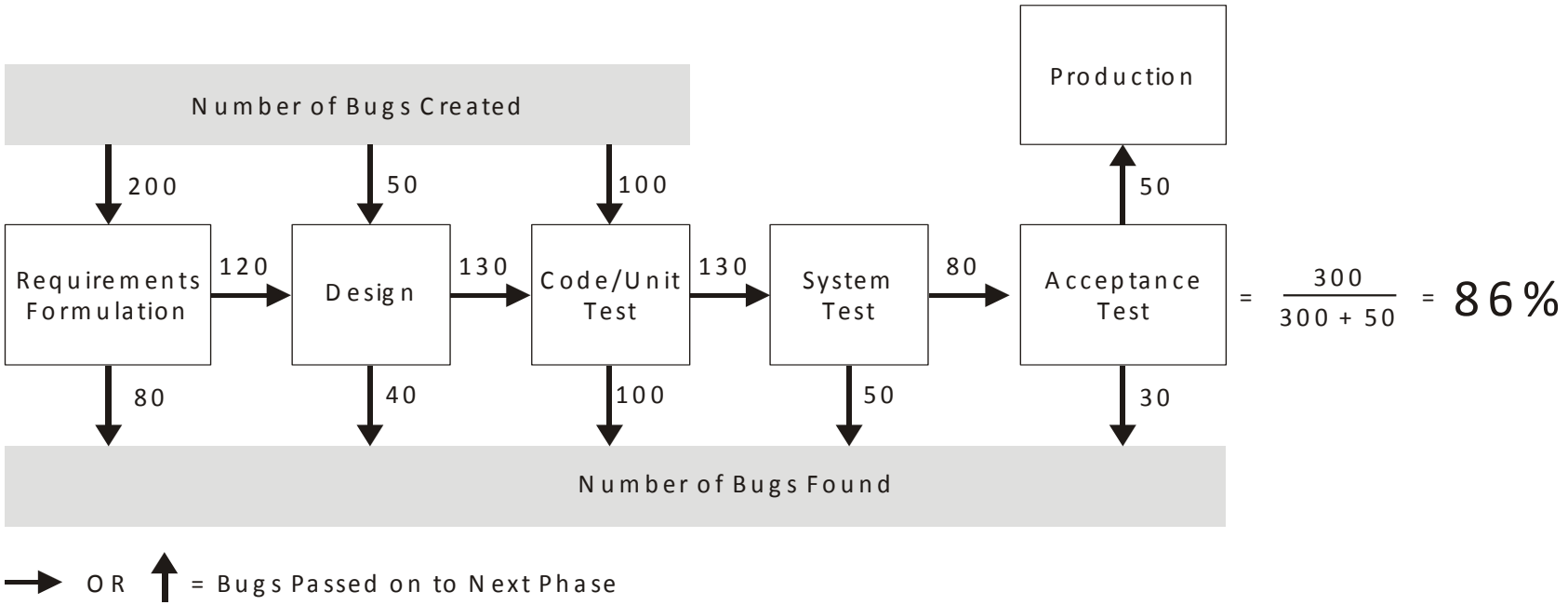
$$DDP = \frac{\text{\# of Valid defects found by this testing}}{\text{Total defects including those found afterwards}} \times 100\%$$



Defect-Based Metrics (cont)

- % of bug fixed
  - DRE (Defect Removal Efficiency) is the percentage of bugs eliminated by reviews, inspections, tests

$$DRE = \frac{\text{Defects removed during a development phase}}{\text{Defects latent in the product at that phase}} \times 100\%$$

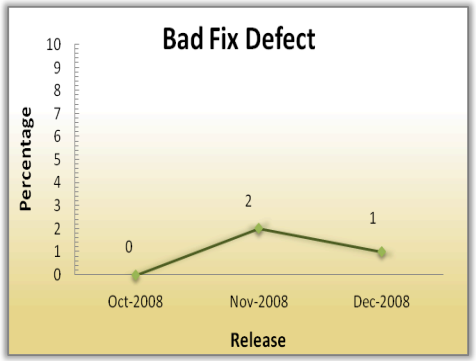


$$= \frac{300}{300 + 50} = 86\%$$

Defect-Based Metrics (cont)

- % of bug fixed (cont)
  - % of defect corrected = # Defect verified Passed/ # defect fixed
  - % of defect fixed = # Defect verified fixed/ # defect found
  - Bad fix defect whose resolution give rise to new defect(s)

$$Bad\ Fix\ Defect = \left[\frac{Number\ of\ of\ Bad\ Fix\ Defect(s)}{Total\ Number\ of\ Valid\ Defects} * 100 \right] \%$$



- Defect Priority Distribution
  - # Priority Defect reported against the test cycle

Cost or effort-based metrics

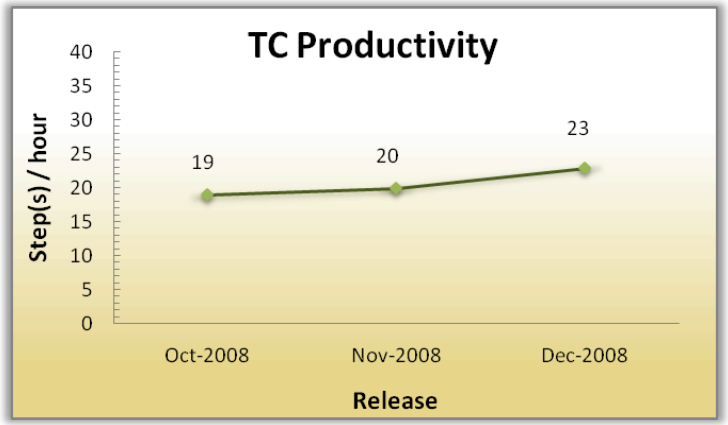
- *Purpose*: to evaluate test effort, provide the context for financial and time-related measurements, can be used in the planning of future test project.
- *Can apply at all test levels*

Some metrics:

- Test case productivity (# of steps/hour or # of TCs/hour)

$$Test\ Case\ Productivity = \left[\frac{Total\ Raw\ Test\ Steps}{Efforts\ (hours)}\right] Step(s)\ /\ hour$$

Test Case Name	Raw Steps
XYZ_1	30
XYZ_2	32
XYZ_3	40
XYZ_4	36
XYZ_5	45
Total Raw Steps	**183**



Efforts took for writing 183 steps is 8 hours.
TCP=183/8=22.8
Test case productivity = 23 steps/hour

Cost or effort-based metrics

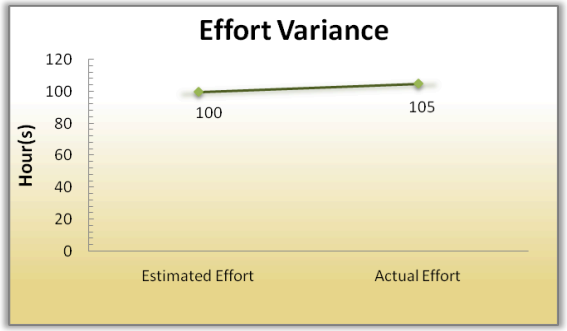
Some metrics (con't)

- Test execution productivity:
  (# of executed TCs/hour or # of executed TCs/day)
- Time for detecting a defect (man-hours)
- Time for removing a defect (man- hours)
- Time for or specification or test planning (man-days, man-hours)
- Review Density (Mins/KLOC or Mins/Page)
- Review Time Deviation

Cost or effort-based metrics

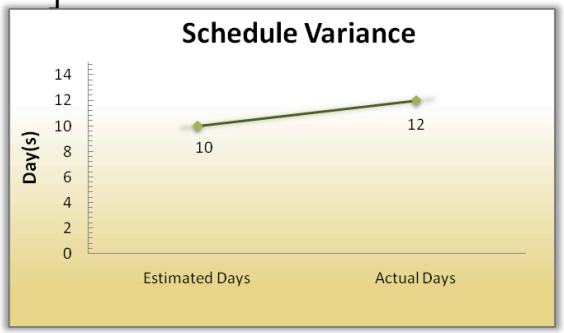
- Some metrics (con't)
  - Effort Variance (EV)

$$Effort\ Variance = \left[\frac{Actual\ Effort - Estimated\ Effort}{Estimated\ Effort} * 100 \right] \%$$



  - Schedule Variance (SV)

$$Schedule\ Variance = \left[\frac{Actual\ No.\ of\ Days - Estimated\ No.\ of\ Days}{Estimated\ No.\ of\ Days} * 100 \right] \%$$



Elements of Bad metrics

Two Sides of Measurement:

…the information will help me understand what is going on and do a better job

…the information will be used against me….

- Measure and/or compare elements that are inconsistent in size or composition
- Create competition between individuals and/or teams
- Easy to “game” or circumvent the desired intention
- Contain misleading information or gives a false sense of completeness

Exercises

- List some metrics we should have to:
  - control test progress
  - monitor the test execution
  - measure how good our test cases are at finding defects
  - analyze and predict bug trends
  - monitor testing productivity
  - monitor product quality

TEST COVERAGE?

- What is test coverage?
- How to measure test coverage?
- Types of test coverage

What is Test Coverage?

- One of the most difficult aspects of testing is answering the question, ***"How good are the tests?"***

➔ We almost always need a quantitative answer.

- Analyzing test coverage is to eliminate gaps in a test suite. It helps most in the absence of a detailed, up-to-date requirements specification.

What is Test Coverage?

- **Test Coverage**: The degree, expressed as a percentage, to which a specified Coverage Item has been exercised by a test suite.

- Code coverage: “Coverage Item" can be
  - Statement
  - Decision/branch
  - Condition
  - Path: control flow sub-paths
  - Data flow: data flow sub-paths

  tracking from unit-level through application-level tests

- Test coverage: “Coverage Item" can be
  - Equivalence Partition Class
  - Boundary of Equivalence Partition Class
  - Sequence of transitions between states
  - Rule (Cause-Effect)
  - Use Case
  - Requirement

Test coverage is applied at all levels

How to measure test coverage?

1. Select suitable **Metrics** we'd like to measure
2. Set goal/ expected ranges/ current baseline for all Metrics

Example:

- Setting an intermediate goal of 100% coverage (of any type) can impede testing productivity.
- Before releasing, strive for 80%-90% or more coverage of statements, branches, or conditions

3. Select method to measure all Metrics

   Example:

   - Manually insert measurement code into tested program
   - Using coverage tools

4. Implement, Execute and Report

Types of test coverage

- Code Coverage
- Interface Coverage
- Requirement Coverage
- Test case Coverage

Code Coverage - Types

- Code coverage analysis is a white-box testing technique. It refers to *"How many source code is covered by the test".* It includes:
  - Function Coverage
  - Statement coverage
  - Branch coverage/ Decision coverage
  - Condition coverage
  - Condition combination coverage (multiple condition coverage)
  - Condition determination coverage
  - Path coverage
  - Data flow coverage
- Can measure
  - Executable file
  - .dll file
  - .lib file

For more detail, refer to white-box testing techniques

Code Coverage – Metrics

Structure-oriented dynamic metrics:

- Function Coverage =
  (number of executed functions / total numbers of functions) * 100%
- Statement coverage =
  (number of executed statements / total number of statement) * 100%
- Branch coverage =
  (number of executed branches / total numbers of branches) * 100%
- Condition coverage = % of all single condition outcomes in complete condition that have been executed by a test suite
- Condition combination coverage = % of combinations of all single condition outcomes in complete condition that have been executed by a test suite
- Condition determination coverage = % of all single condition outcome that independently affect a decision outcome that have been executed by a test suite
- Path coverage =
  (number of executed paths / total number of paths) * 100%

Note:

- Condition outcome is True or False

For more detail, refer to white-box testing techniques

Code Coverage – Summary

- Code-based test coverage is calculated by the following equation:

  Test Coverage = I^e / Tlic * 100%

  where:

  - **I^e** is the number of items executed expressed as code statements, code branches, code paths, data state decision points, or data element names.
  - **Tlic** is the total number of items in the code.

Code Coverage – How to measure?

Structure-oriented dynamic metrics will be measured at run-time

There are two ways:

1. Using Instrumentation:

- Instrumentation is the insertion of additional code into the program in order to collect information about program behavior such as code coverage during execution
- Instrumentation often works this way:
  - Determination of the executed program parts
  - Manually insert counters in the program and initializes them with zero.
  - During program execution, the counters are incremented when they passed. At the end of the test execution, the counters are contain the number of passes through the according program fragments.
  - If counter = 0: the program fragment has not been executed
  - If counter > 0: …
- Limitation:
  - Manual instrumentation is error-prone
  - Take resources

Code Coverage – How to measure? (cont)

2. Using Tools:

- Purpose: Measure code coverage to increase the productivity and indirectly improve the quality of the test object
- Strong points:
  - Can get over the limitation of instrumentation.
  - A lot of tools perform this tasks
- Some tools:
  - BullseyeCoverage:
    - BullseyeCoverage is a code coverage analyzer for C++ and C that tells you how much of your source code was tested
    - Integrated with Microsoft Visual Studio
    - Include or exclude any portion of your project code
    - Run-time source code included, for custom environments
    - Merge results from distributed testing
    - Refer to: http://www.bullseye.com/help/ for help
  - Android SDK emma library
  - …

Code Coverage – Example

Example with BullseyeCoverage tool for C++ application:

Step to do:

1. Select **metrics**:
   - Function coverage
   - Condition/ Decision coverage

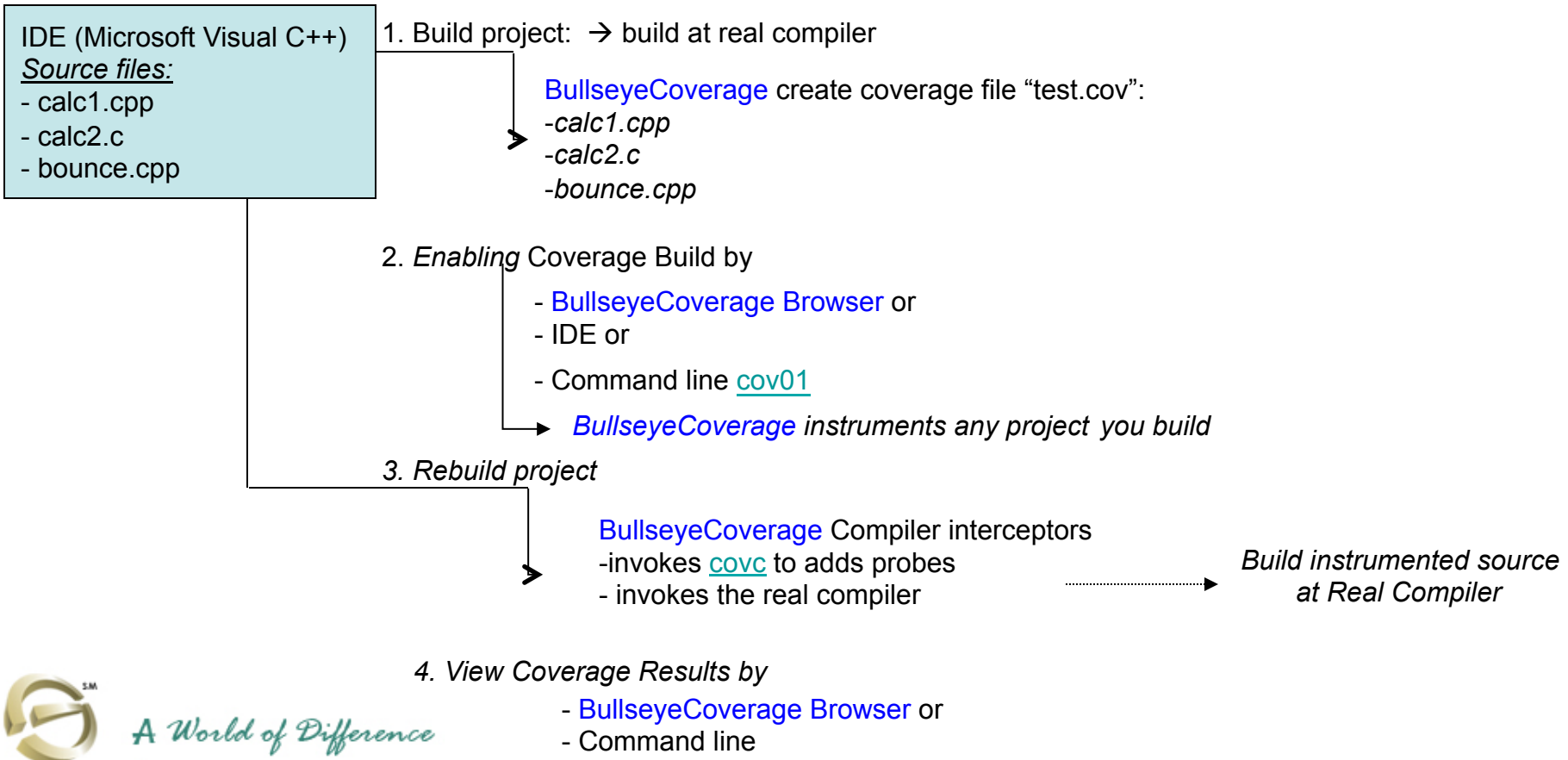
2. Set goal/ expected ranges/ current baseline for all Metrics
   - Function coverage = 80%
   - Condition/ Decision coverage = 80%

3. Using *BullseyeCoverage tool to measure coverage*

4. Implement, Execute and Report

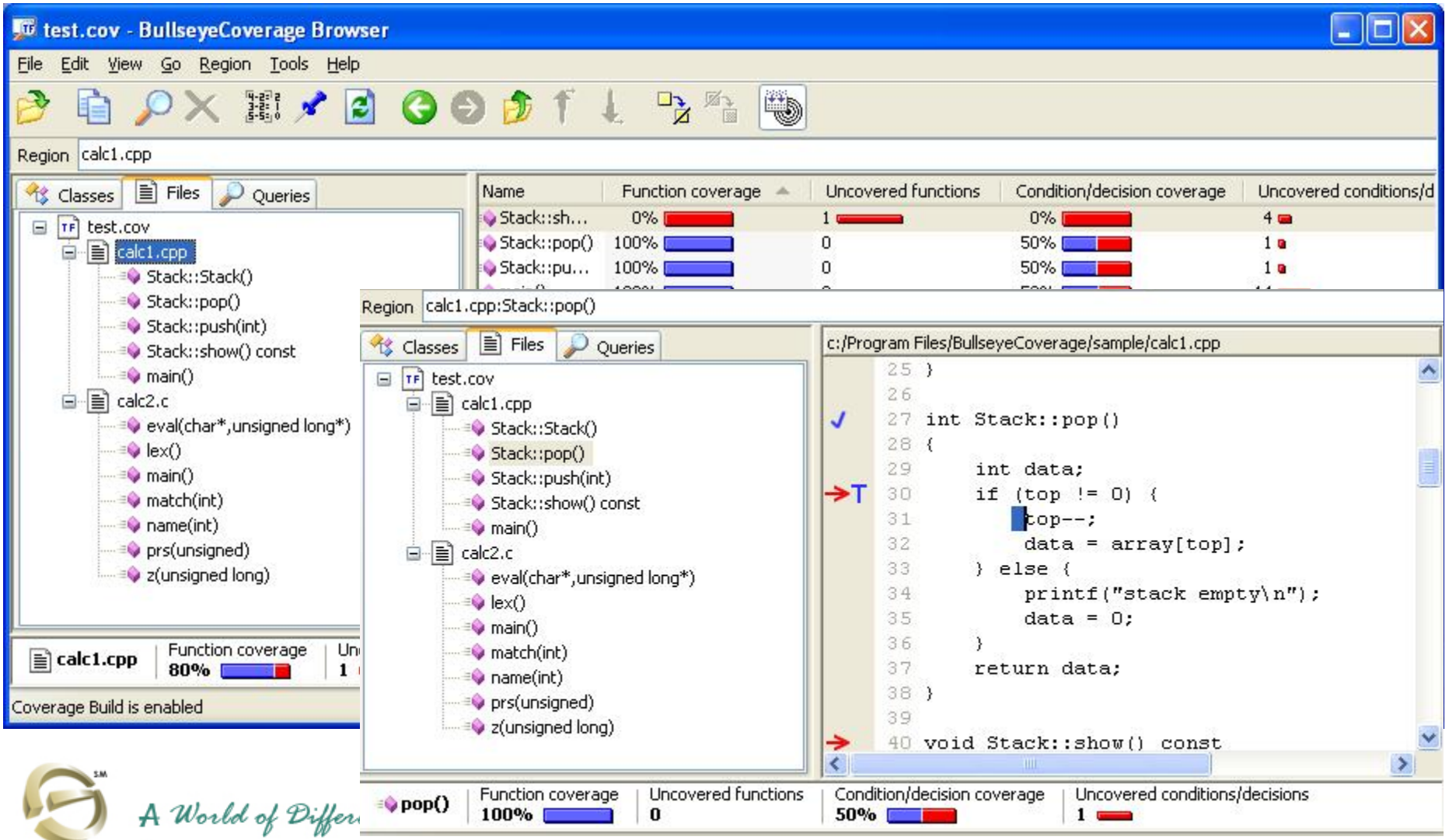
Code Coverage – Example (cont)

- *Pre-steps*:
  - Select source files will be measured
  - Set environment variable (COVFILE) for coverage file, if not default coverage name is “test.cov”



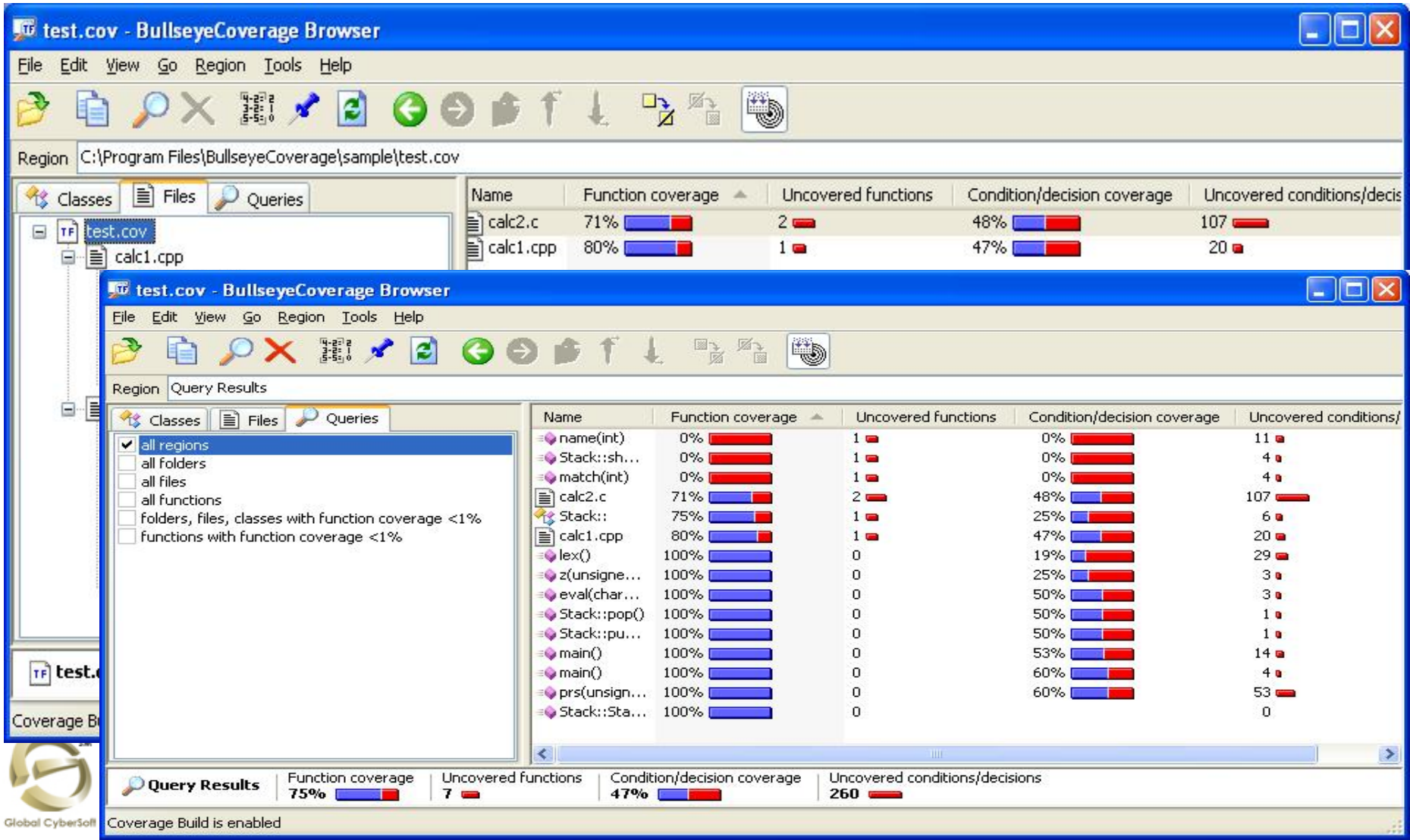
Code Coverage – Example (cont)

- *View results by using BullseyeCoverage browser:*



Code Coverage – Example (cont)

- *View results by using BullseyeCoverage browser:*



Code Coverage – Example

- EMMA code coverage for Java application:

EMMA Coverage Report (generated Sat May 09 08:24:44 ICT 2015)

[all classes][com.nuance.nina.promise]

COVERAGE SUMMARY FOR SOURCE FILE [Deferred.java]

name	class, %	method, %	block, %	line, %
Deferred.java	100% (6/6)	85% (23/27)	79% (321/407)	77% (80.9/105)

COVERAGE BREAKDOWN BY CLASS AND METHOD

name	class, %	method, %	block, %	line, %
class Deferred	100% (1/1)	78% (14/18)	74% (238/321)	74% (69/93)
cancel (): boolean		0% (0/1)	0% (0/27)	0% (0/10)
getCancelException (): Exception		0% (0/1)	0% (0/3)	0% (0/1)
resolve (): void		0% (0/1)	0% (0/4)	0% (0/2)
setCancel (Callabl				
whenRejected (Prom				
whenResolved (Prom				
tee (Deferred): Pr				
<static initialize				
Deferred (): void				
Deferred (long): v				
execute (PromiseRu				
getId (): long				
getState (): Promi				
reject (): void				
reject (Object): v				
resolve (Object):				
update (Object): v				
whenUpdated (Promi				
class Deferred$5				
<static initialize				
class Deferred$1				
Deferred$1 (Deferr				
run (Object): void				
class Deferred$2				
Deferred$2 (Deferr				
run (Object): void				

EMMA Coverage Report (generated Sat May 09 08:24:44 ICT 2015)

[all classes]

OVERALL COVERAGE SUMMARY

name	class, %	method, %	block, %	line, %
all classes	90% (324/361)	79% (1545/1948)	75% (33078/44054)	76% (6935.9/9110)

OVERALL STATS SUMMARY

total packages:	7
total executable files:	109
total classes:	361
total methods:	1948
total executable lines:	9110

COVERAGE BREAKDOWN BY PACKAGE

name	class, %	method, %	block, %	line, %
com.nuance.nina.promise	50% (7/14)	50% (26/52)	47% (364/767)	43% (84.8/197)
com.nuance.nina.agency	100% (4/4)	40% (19/48)	70% (332/477)	61% (85.7/141)
com.nuance.nina.mmf	90% (228/253)	84% (1131/1354)	73% (22081/30430)	74% (4690.8/6324)
com.nuance.nina.grammar	100% (3/3)	70% (21/30)	81% (797/986)	81% (169.5/209)
com.nuance.nina.dialog	95% (36/38)	78% (218/281)	83% (5752/6933)	84% (1152.1/1371)
com.nuance.nina.mmf.listeners	92% (37/40)	63% (59/93)	83% (1427/1710)	87% (260.8/301)
com.nuance.nina.ssml	100% (9/9)	79% (71/90)	85% (2325/2751)	87% (492.2/567)

[all classes]

EMMA 2.0.5312 (C) Vladimir Roubtsov

Interface Coverage

- Interface Coverage is refer to *"How many interfaces in the system are covered by test?"*
  - % of interfaces tested
    - # of tested interface/ Total # of interface in Basic Design
  - % of interface usage tested
    - Matrix with 2 dimensions (Usage, Interface)
  - % of interface parameters tested with test technique XYZ
    - Matrix with 3 dimensions (Usage, Interface, Data Input)

Requirement – Based Coverage

- Requirement coverage is refer to "*How many requirements are covered by Test cases".* It includes:
  - Requirement Coverage
    - (# of currently developed TCs/ # of estimated of TCs)*100%
  - Equivalence Class Coverage (EC-Coverage)
    - (# of executed EC/ # of identified EC)*100%
  - Boundary Value Coverage (BV-Coverage)
    - (# of executed BV/ # of identified BV)*100%
  - Use Case coverage (UC-Coverage)
    - (# of executed UC/ # of identified UC)*100%
  - Use Case Flows coverage (UCF-Coverage)
    - (# of executed possible flows of UC/ # of possible flows of UC) * 100%

Requirement – Based Coverage

- State Transition Coverage (ST-Coverage)
  - State coverage
    - (# of state has been reached at least once / total state) * 100%
  - Valid Transition coverage
    - (# of valid transition has been executed / total transitions) * 100%
  - Invalid Transition coverage

Based on Black-box TC design techniques, we can identify the total of EC, BV… for each requirement

Exercise

Requirement – Based Coverage – How to measure?

- Make requirement traceability by using excel file …

	Test Cases														
Objectives	TC # 1	TC # 2	TC # 3	TC # 4	TC # 5	TC # 6	TC # 7	TC # 8	TC # 9	TC # 10	TC # 11	TC # 12	TC # 13	TC # 14	TC # 15
Req. 1	X														
Req. 2						X									
Req. 3		X		X											
Req. 4		X	X												
Feature 1							X								
Feature 2	X				X										
Feature 3					X										
Feature 4		X				X									

Reference

Appendix: Course detail form

Author	Son Pham	**Duration**	3 hours
Category	Theory	**Type**	Test Analysis and management

Examination	
Intended Audience	
Pre-requisites	Test Lead engineer or above
Completion criteria for the course	
Criteria for granting training waivers	

Thank you

THANK YOU

Inquires regarding the above may be directed to:
Someone, Title, truonghx@gcs-vn.com