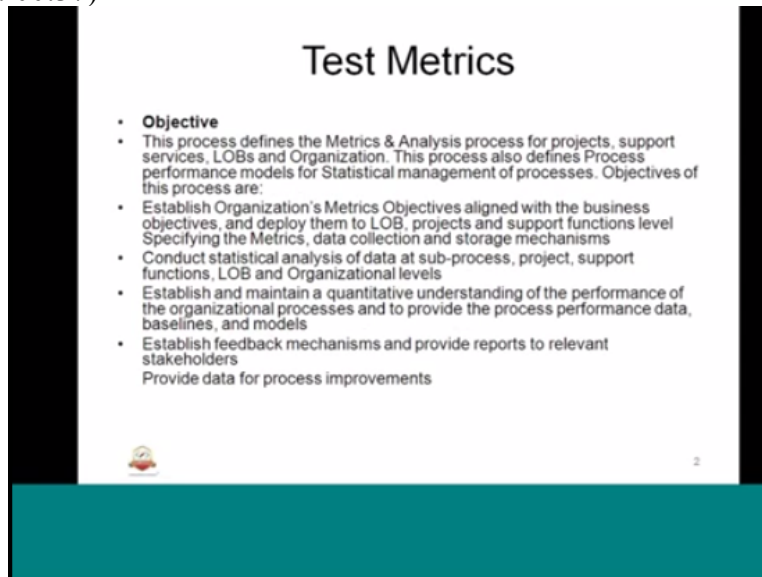


Embedded Software Testing Unit 3: Static analysis and core reviews & Metrics

Lecture 6

Seer Akademi – NPTEL MOU

Yeah Hello Welcome you to the next session of Embedded Software Testing. This is unit-3 the last session of this unit - lecture 6. So, where we go through some of the software metrics and books the tools that are used it is part of the static testing. And this unit is about static analysis core reviews and metrics and we know that in the last session revealed about
(Refer Slide Time: 00:37)

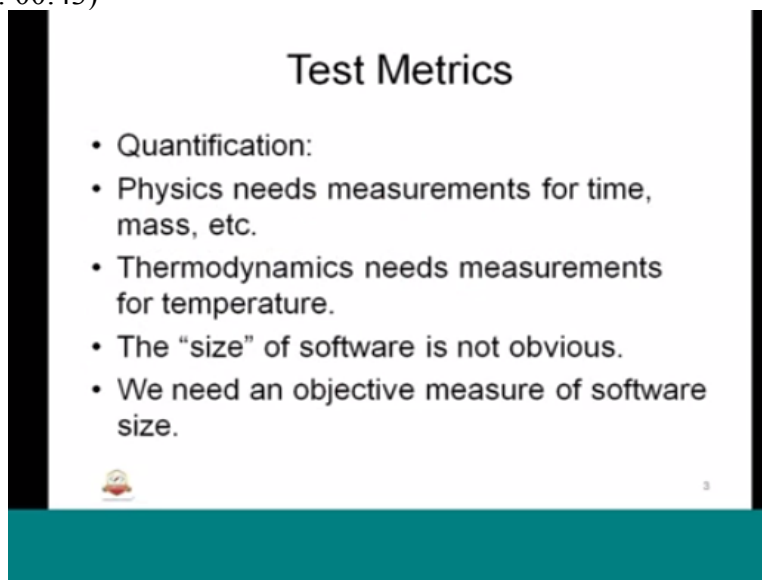


Test Metrics

- **Objective**
- This process defines the Metrics & Analysis process for projects, support services, LOBs and Organization. This process also defines Process performance models for Statistical management of processes. Objectives of this process are:
- Establish Organization's Metrics Objectives aligned with the business objectives, and deploy them to LOB, projects and support functions level Specifying the Metrics, data collection and storage mechanisms
- Conduct statistical analysis of data at sub-process, project, support functions, LOB and Organizational levels
- Establish and maintain a quantitative understanding of the performance of the organizational processes and to provide the process performance data, baselines, and models
- Establish feedback mechanisms and provide reports to relevant stakeholders
Provide data for process improvements

2

The metrics,
(Refer Slide Time: 00:43)



Test Metrics

- **Quantification:**
- Physics needs measurements for time, mass, etc.
- Thermodynamics needs measurements for temperature.
- The "size" of software is not obvious.
- We need an objective measure of software size.

3

(Refer Slide Time: 00:44)

Test Metrics

- Quantification based on Structural metrics:
 - Attention is focused on control-flow and data-flow complexity.
 - Structural metrics are based on the properties of flowgraph models of programs.



3

The importance of a metrics so what is the basis on which we deserve the metrics especially for these software size. So, we know that we are not deciding the size by the code not only the code but the complexity and structure of the code is also important time in the program that is underneath the embedded software. That is how we calculate the metrics.

(Refer Slide Time: 01:10)

Test Metrics Importance

- Metrics is used to improve the quality and productivity of products and services thus achieving
- Customer Satisfaction.
- Easy for management to digest one number and drill down, if required.
- Different Metric(s) trend act as monitor when the process is going out-of-control.
- Metrics provides improvement for current process.



UCSE, ISSN : 0975-3397

5

Basically these metrics are required on a day-to-day basis or week-on-week basis or fortunately basis or based on the customer's inputs is to make sure that how well we have done the program testing or well the other on track. Well the program is progressing.

(Refer Slide Time: 01:31)

Test Metrics Importance

- **Objective**
- This process defines the Metrics & Analysis process for projects, support services, LOBs and Organization. This process also defines Process performance models for Statistical management of processes. Objectives of this process are:
- Establish Organization's Metrics Objectives aligned with the business objectives, and deploy them to LOB, projects and support functions level Specifying the Metrics, data collection and storage mechanisms
- Conduct statistical analysis of data at sub-process, project, support functions, LOB and Organizational levels
- Establish and maintain a quantitative understanding of the performance of the organizational processes and to provide the process performance data, baselines, and models
- Establish feedback mechanisms and provide reports to relevant stakeholders
- Provide data for process improvements



Likewise, so we
(Refer Slide Time: 01:37)

Test Metrics Lifecycle

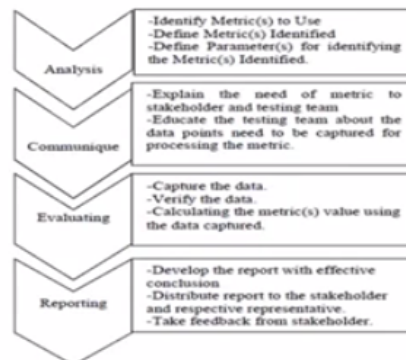


Fig. 2 Software Metrics Lifecycle



Also have test metrics lifecycle analysis, communiqué, evaluating, and the reporting. So, we basically identify define then we will communicate the identified metric how it should be entered how should be used then with the help of that we will evaluate after capturing the data, verifying the data and the metrics calculated using one formula on the capture data. Then we will try to report it effectively with various formats of either using the tool or with the help of any documentation that is useful for the stakeholders. Similarly, we will take the feedback on stakeholders about the complete report of the test metrics.

(Refer Slide Time: 02:33)

Software Testing Metrics types

- Manual Testing metrics
- Performance Testing metrics
- Automation Testing metrics



Then we have studied about the software testing metrics types.
(Refer Slide Time: 02:40)

Software Testing Metrics

- **Manual**
 - Test Case Productivity
 - Test Execution Summary
 - Defect Acceptance
 - Defect Rejection
 - Bad Fix Defect
 - Test Execution Productivity
 - Test Efficiency
 - Defect Severity Index
- **Performance**
 - Performance Scripting Productivity
 - Performance Execution Summary
 - Performance Execution Data - Client Side
 - Performance Execution Data - Server Side
 - Performance Test Efficiency
 - Performance Severity Index



Manual, Performance and Automation testing metrics so, manual what we do is test case productivity, execution productivity, defect and execution efficiency, severity about this manual type of metrics. We collect in performance in terms of our execution some of scripting and that especially the performance of the whole environment and technicality of the tested program will be.

(Refer Slide Time: 03:08)

Software Testing Metrics

- **Important metrics details: TCP**

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

Example

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.

$$\text{TCP} = 183/8 = 22.8$$



Reported and some of the important metrics also we have studied like test case productivity. So, Total number of raw steps divided by number of hours that took to develop that will give the steps per hour is what they discussed about the defects. This many steps can be developed for our work so based on this it will be used for
(Refer Slide Time: 03:35)

Software Testing Metrics

- **Important metrics details: DA (Defect Acceptance)**
- This metric determine the number of valid defects that testing team has identified during execution

$$\text{Defect Acceptance} = \left[\frac{\text{Number of valid Defects}}{\text{Total Number of Defects}} * 100 \right] \%$$



Estimation and the improvement over the period similarly, we have defect acceptance in terms of how many defects are valid and what is the total number of defects that gives the percentage of defect acceptance.
(Refer Slide Time: 03:52)

Software Testing Metrics

- **Important metrics details: DR (Defect Rejection)**
- This metric determine the number of defects rejected during execution.

$$\text{Defect Rejection} = \left[\frac{\text{Number of Defects Rejected}}{\text{Total Number of Defects}} * 100 \right] \%$$



Similarly, we have defect rejection total number of defects that are rejected divided by total number of defects give a percentage of the rejection ratio.
(Refer Slide Time: 04:05)

Software Testing Metrics

- **Important metrics details: TEP (Test Execution Productivity)**
- This metric gives the test cases execution productivity which on further analysis can give conclusive result

$$\text{Test Execution Productivity} = \left[\frac{\text{Number of TCexecuted (Te)}}{\text{Execution Efforts (hours)}} * 8 \right] \text{Execution(s)/Day}$$

Where Te is calculated as,

Where,

Base Test Case = No. of TC executed at least once.

T (1) = No. of TC Retested with 71% to 100% of Total TC steps

T (0.66) = No. of TC Retested with 41% to 70% of Total TC steps

T (0.33) = No. of TC Retested with 1% to 40% of Total TC steps



Similarly, test case execution productivity number of test cases executed / the execution efforts in terms of hours will give the productivity of the execution.
(Refer Slide Time: 04:16)

Software Testing Metrics

- **Important metrics details: TEP (Test Execution Productivity)**
- This metric gives the test cases execution productivity which on further analysis can give conclusive result

$$\text{Test Execution Productivity} = \left[\frac{\text{Number of TCexecuted (Te)}}{\text{Execution Efforts (hours)}} * 8 \right] \text{Execution(s)/Day}$$

Where Te is calculated as,
Where,

Base Test Case = No. of TC executed at least once.

T (1) = No. of TC Retested with 71% to 100% of Total TC steps

T (0.66) = No. of TC Retested with 41% to 70% of Total TC steps

T (0.33) = No. of TC Retested with 1% to 40% of Total TC steps



So, it is more productive if it is 100%.

(Refer Slide Time: 04:27)

Software Testing Metrics

- **Important metrics details: TE (Test Efficiency)**
- This metric determine the efficiency of the testing team in identifying the defects. It also indicated the defects missed out during testing phase which migrated to the next phase

$$\text{Test Efficiency} = \left[\frac{DT}{DT + DU} * 100 \right] \%$$

Where,

DT = Number of valid defects identified during testing.

DU = Number of valid defects identified by user after release of application. In other words, post-testing defect



It is based on hours next one is the efficiency. So, efficiency is basically an important metric to determine the efficiency of the testing team in identifying the defects and execution of the software testing.

(Refer Slide Time: 04:51)

Software Testing Metrics

- **Important metrics details: Defect Severity Index (DSI)**
- This metric determine the quality of the product under test and at the time of release, based on which one can take decision for releasing of the product i.e. it indicates the product quality.

$$\text{Defect Severity Index} = \left[\frac{\sum (\text{Severity Index} * \text{No. of Valid Defect(s) for this severity})}{\text{Total Number of Valid Defects}} \right]$$

One can divide the Defect Severity Index in two parts:

a) *DSI for All Status defect(s):*

This value gives the product quality under test.

b) *DSI for Open Status defect(s):*

This value gives the product quality at the time of release. For calculation of DSI for this, only open status defect(s) must be considered.

$$\text{DSI (Open)} = \left[\frac{\sum (\text{Severity Index} * \text{No. of Open Valid Defect(s) for this severity})}{\text{Total Number of Valid Defects}} \right]$$



Then we came to defect severity index in terms of how severe are the defects are. So, this is calculated based on formula which is cot index from the number of defects or that severity we have a total number of well defects that will give the severity index and that can be divided into two parts, severity index for all valid defects, and Severity index for open statistics. (Refer Slide Time: 05:26)

Software Testing Metrics

- **Important metrics details: Automation coverage**
- This metric gives the percentage of manual test cases automated

$$\text{Automation Coverage} = \left[\frac{\text{Total No. of TC Automated}}{\text{Total No. of manual TC}} * 100 \right] \%$$

Example

If there are 100 Manual test cases and one has automated 60 test cases then Automation Coverage = 60%

☒




Similarly, the automation coverage in terms of total number of our coils percentage will report. The example there are 10 test cases that are automated out of 100 tests then the coverage is 60% sorry it's 60 test cases are automated in automation coverage, the total number of percentage is (Refer Slide Time: 05:52)

Software Testing Metrics

- Important metrics details: *Automation coverage*
- This metric gives the percentage of manual test cases automated

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

☒




UCSE, ISSN : 0975-3397 16

60%. Then there are two important metrics in terms of progress on the embedded software testing or optical viewfinder. So, actual effort- estimate effort / estimated efforts into 100 is what effort variance. So, this should not be more than 5%. Usually the embedded industry variance should not be more than + or – 5%.

(Refer Slide Time: 06:24)

Software Testing Metrics

- Important metrics details: *Schedule Variance*
- This metric gives the variance in the estimated schedule i.e. number of days

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


UCSE, ISSN : 0975-3397 18


If it is minus also it is not good it is plus also it is not good.

(Refer Slide Time: 06:29)

Software Testing Metrics

- Important metrics details: **Effort Variance**
- This metric gives the variance in the estimated effort

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


 UCSE, ISSN : 0975-3397 17

But this we have underestimated or we would have works with that shows the index basically on how the estimation effort works and how well the program are coped up with those systems. (Refer Slide Time: 06:43)

Software Testing Metrics

- Important metrics details: **Schedule Variance**
- This metric gives the variance in the estimated schedule i.e. number of days

$$\text{Schedule Variance} = \left[\frac{\text{ActualNo. of Days} - \text{EstimatedNo. of Days}}{\text{EstimatedNo. of Days}} * 100 \right] \%$$

 UCSE, ISSN : 0975-3397 18

schedule overrun = >.5%
 schedule underrun = <.5%

The next one is being the schedule variance. Here iteration for the duration actual number of days - estimated number of days/ estimated number of days bounded into the percentage. So, this also cannot be more than + or - 5% of variance because we cannot afford too much of an overrun. Two important terms schedule overrun that means if it is -> -5%. Schedule under run if it <5%.

This schedule is so fast that it is well ahead of what has been estimated. That is what it gives an indication. So, that is what is about schedule variance.

(Refer Slide Time: 07:57)

Software Testing Metrics

- **Important metrics details: Scope change**
- This metric indicates how stable the scope of testing is.

$$\text{Scope Change} = \left[\frac{\text{Total Scope} - \text{Previous Scope}}{\text{Previous Scope}} * 100 \right] \%$$

Where,

Total Scope = Previous Scope + New Scope, if Scope increases

Total Scope = Previous Scope - New Scope, if Scope decreases



We have the scope change so scope change indicates how stable the system is and how much of scope change from the original scope. Scope can be increased as well as decreased depending on the complexity of the project and overall the program went ahead in terms of development testing. So, that is what the way we have studied in the lecture 5. Today we will continue on the six which details about the metrics for automation. Automation scripting productivity, automation test execution productivity, automation coverage costs compression all this will be automated.

So, that is the sort of a thing we generate is typically based on some tools like mixing sea power. So, we know that common metrics are used effort variance, schedule variance, scope changes. (Refer Slide Time: 09:02)

Metrics for Software Testing: Managing with Facts

- For RBT :
 - How many requirements are completely tested without any failures?
 - How many requirements have failures?
 - How many requirements are untested?



So basically for testing we need to deal with the facts in terms of managing what are we facts that we have considered for generating be test metrics, test metrics basically. So, for RBT, RBT is nothing but Requirement Based Testing. So, we need to have a metrics ridiculously done in terms of reporting for example how many requirements are completely tested without any

failures? How many requirements have failures? How many requirements are untested? So, these are very important metrics.

It is not just enough to report a pass/fail count. Failure count, Invalid count, Not applicable count. There are different sorts of counts which are a result of either re-execution or six in the software core or pics in the requirements. It is also important to highlight how many requirements in terms of what are the sorts of requirements of functionality? There are executed without any failures. That is what this indicates and similarly how many requirements really have failures? That is what the functionality that is failing is?

That is what should be indicated using the metrics. And the other one is the other requirements we move uncovered or untested is also another important element of the software metrics which needs to be factored. Basically, this will be a building work for the components in terms of oh well how solid is a context testing and strategy is and how well the product is. Basically, the both are complemented each other and as a whole it should be very well supported with this sort of the metrics.

So, how do we achieve with the comprehensive actually? Before releases a degree software of our customers we will build all this metrics in terms of reporting. So, what are the fixes we are fixed? What are the issues we had? And how many programs? Basically he will ask correctly how you tested. What is the strategy? What is the report that you have?

All this will be reported. And coverage is also one of the matrices so coverage will copy requirements as you said in terms of the failures non-failures and the coverage etc, so basically more thoroughly tested products will have more confident outputs in terms of metrics so there is no surface at large.

So, definitely coverage is a strict concept because it has multiple dimensions including code a very detailed combination for all these tests design techniques coverage also requirements covering more likewise we have various aspect factor for all the reporting. So basically at the end up testing we will be reporting bumper percentage the goal is to achieve 100% testing requirements or 100% requirement and at the end whatever the regression you do this should not be any failures.

So always the percentage should be decreasing that is what the aim of the metrics report well we are reporting a band of error's end of our error's inverse of our testing

(Refer Slide time: 13:58)

Metrics for Software Testing: Managing with Facts

Requirements Area	Currently Untested	Tested and Failed	Tested and Passed
Functionality	7%	3%	90%
Usability	25%	17%	58%
Reliability	0%	17%	83%
Performance	5%	10%	85%
Installability	7%	13%	80%

Table 1: Requirements Coverage by Area



Provided by Rex Black Consulting Services (www.rbc-us.com)

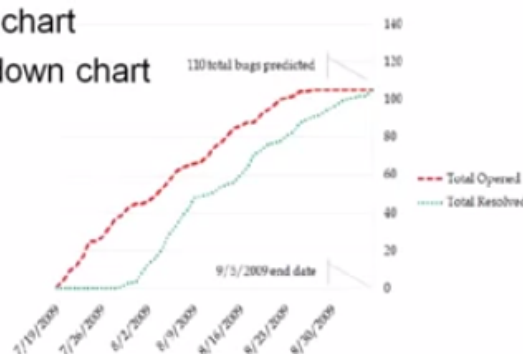
4

So let's say look at the some of the example for the various charts that organization uses in terms of reporting the facts and the software metrics, you can see an example is basically being provided by mix black account setting services an example again, so nicely it is presented in terms of requirements are in one column and the current progress is showing in terms of coverage it is this table talks about requirements coverage by area currently untested tested and failed tested and passed.

So altogether the leftmost column should be one hundred percent that what it aims for right. So what are the types of requirements or categorization of the requirements but it looks forward. (Refer Slide time: 14:55)

Metrics for Software Testing: Managing with Facts

- Trend chart
- Burn down chart



Provided by Rex Black Consulting Services (www.rbc-us.com)

4

When is our functionality that means requirements in terms of functionality how are they tested? What are the values for other puzzles? The next aspects are something like usability how useful the product is comes on usability aspects then reliability requirements then we have performance requirements, then we have instability requirements so all this categorization of the requirements you cannot anymore like performance, performance article, performance again you can sub

categorized timing, speed memory likewise communication also one of the requirement functionality or requirements category that you can have for all these here to report on a total saying that how many are currently being untested? How many are failed? And how many passed?

Here in the first one you see function there are a seven percent of the total number of a test us not being tested, there is ninety-three percent that have been tested in that three percent are failed ninety percent is passed, similarly next one twenty-five percent untested seventy percent failed fifty eight percent passed.

So basically all the facts in terms of requirements it will bring out, so very nice way of providing the report. We say the metrics can be factored and presented so that the customer and the higher program management will be aware of what is going on in the software testing project it basically use a trend.

Next one is again from the same have said very interesting way of for presenting V the representation in terms of all it's called as a trend chart also called as a burn down chart that means how well the program is progressing? And where it is going to end? So what is the trend that is what is happening in terms of various metrics that are going to be that are being captured during the test execution?

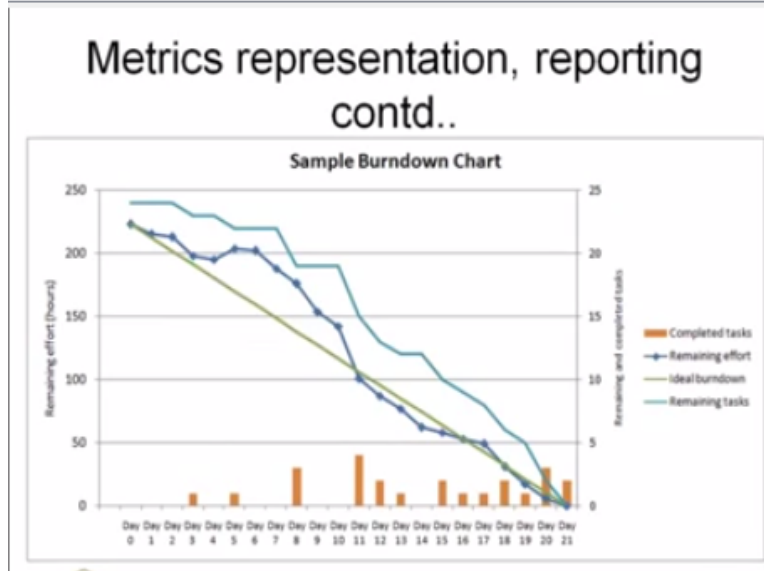
This is reported using an excel sheet or there are several tools / be used for reporting the metrics you can see so it can be represented by anyway but here our example, this figure talks about the bugs open and relations how it is being rendered for the particular duration, so we know the teeth at the end it is here over we want to reach it by the current trend, so 9\5\2009 was the last date and the program is being reported every week starting from 719 the program the test program was started on cyanide to and the till 8:30 august 30.

This was reported and it was ending on 9\5, so you can see there are two colored graph or the dotted lines the red one being total opened bug open and the green being total resort that means how many bugs on the particular day are open and how many bugs have been closed see finally, we should aim for the complete bugs closure that means the green should meet the red so that is what the trend should end with testing or MD program.

So here there are total number of normal and then 110 bucks of total bucks are projected and start of the program so there is a zero bug and the next week so we have about you have to put a cursor here and move it and said about 30 bugs are opened out of which still it is zero no resolution or bugs are still open and during the second week we have about 50 bugs that are open and you at the same time there are about 10 or 15 bugs have been reserved.

Similarly we have one two three fourth week about 70 bucks have been reported and about 60 to 70, 50 to 60 bugs is odd, and in the fifth week we see about the 92 bugs have been opened and about 80 bugs having 70 to 95 having closed our resolve those bugs are resort, and in sixth week we have about 110 bugs and bugs are already open that means testing is done but it's not fixed, because not all the bugs have been resolved sorry about 90 bugs are they still and in the end when we reach the milestone or in the last date we have 110 bugs there are being resolved this is all the trendsetting trend set or trend chart will be drawn, you can clean these details using an

excel sheet, excel sheet we need total number of open bugs total number of evolve and we need a date and the upper limit of for the right hand side in terms of goggle references. So with the help of that excel sheet there are formulas we can apply and draw the chart, of course there are different types of charts it is up to the user or the test manager or the test lead how we can present it so that it is convincible to the customer and the higher program management. (Refer Slide time: 22:13)



So this is a trend chart at the first one the next one being the burn down chart that means this basically talks about the burn down of the program how I am going to meet on a given deadline, you can see an example here of take tasks as test, so there are hours like 250 hours remaining and 25 tasks are to be executed, how I am going to reach it? so from the I will start from the left hand side with their total number of allocated hours.

And as I progress my hours are going to decrease accordingly the tasks also expected to be reduced so that many tasks we have reached and when we are left with they are zero our task also should be reducing on the downside that's what this burn down chart will be prepared. This is also being developed using an excel sheet here you can see on the horizontal axis from a day zero to 21, how the burn down chart is being projected are in terms of day.

And on the column side we have two columns from remaining a hog and on the right-hand side the number of tasks remaining and then completed task, so you can add as many as a metrics into this for about 3 metrics you can see one being the green one which is nothing but the ideal burner. That means from the beginning how are we going to burn down towards the end that means how are we want to complete the program on 21st day what is the ideal burn down chart should look like and this one the arrangers sort of a column is the completed tasks.

So on each given one day or maybe alternate days or a week there are number of tasks that are going to be completed and that will be highlighted in this color so all together those tasks should accumulate as 25 and the blue ones are remaining reports at least we have a total of like a cocoon the hours and you can see the variation finally we are left with almost zero it where should be done well before the day 21 and light blue and is the remaining tasks so from the day one on there is zero today we have all the tasks remaining.

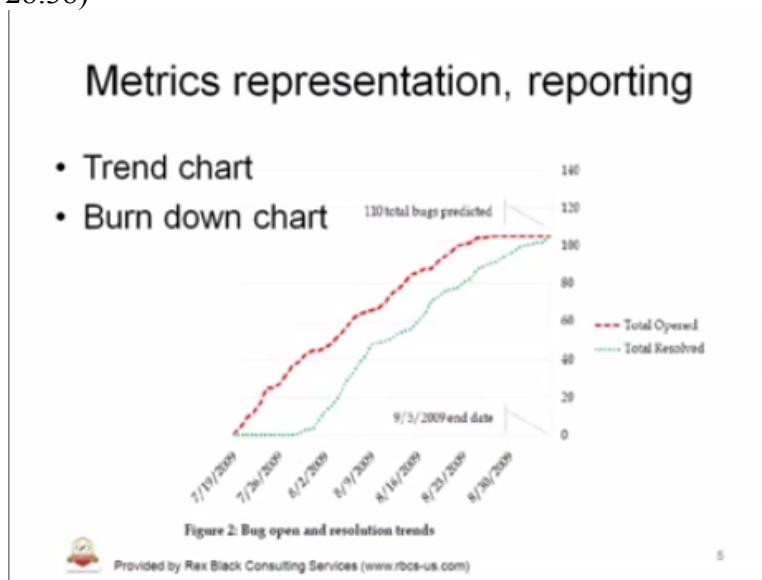
And we have the full time available as we progress the task should reduce as well as the day of the remaining time we will also getting reduced that's how the ideal burn down you can see, for example you take on day eight so on day 0, 250 hours are left and task remaining are above 25, so this is how we are going to develop the burn down chart and you see an example day 8 we have complicated about how many if I draw in the right hand side, one two three so about three tasks plus we already computed on day five one task and day 3-1 task so three plus four plus, so that means about five task have been completed.

On this data day 8 so you can see the total number of tasks from 25 to 20 it has combined, similarly on day 15 suppose oh so sorry okay on day 11, 5 plus another 4,9 so total number of tasks should become about nine this is what you can see 9-25 about a 15 tasks are remaining, so been while for executing or doing a task we have also consumed some effort alright so the effort also accordingly has come down.

So remaining effort on each day and they each reporting day you can see it is getting reduced it taken depends on the resources how much effort they have put whether resources are on vacation and overall program you can see they up and down are common but at the end of the program. We should make sure that we are going to reach the tasks completed as well as the hours well within that the effort hours whatever we are going to going to consume should be well within the range here in this example.

We should not consume more than 250 hours and we should make sure that when we reach 21 days we are done with good defect asks, so three parameters are important the duration the effort and the number of tasks. With these three parameters we are going to report the burn down chart, so this is basically being done with the help of several tools or a MTP or Microsoft Project man or excessive so developing that and going to be embedded software testing but typically these parameters are used for reporting the metrics of sample burn down chart.

(Refer Slide time: 28:38)



All you have to remember is two things we will report it for embedded softer testing then chart and burn down chart these are important metrics this will give a clear picture of where are we heading.

(Refer Slide time: 28:54)



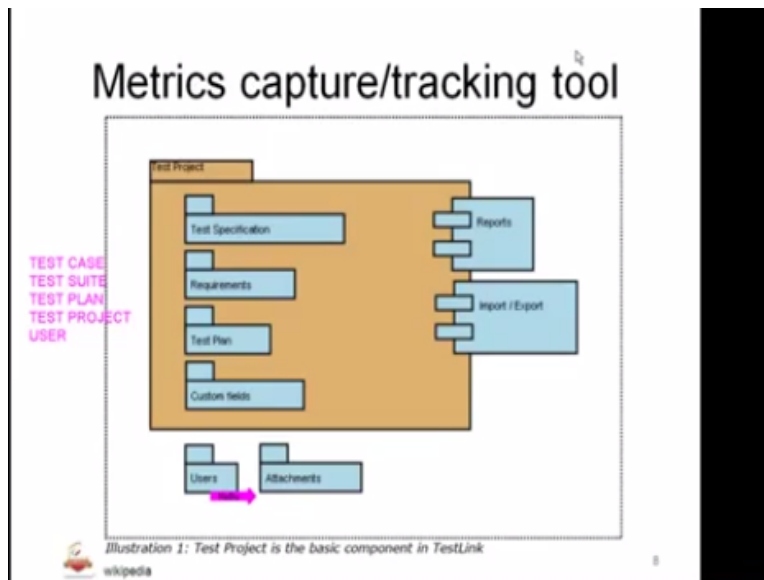
The next one being the metrics capture and tracking tool, what are the tools in general reviews? as I said there are several tools exist in the embed software industry or that could be excel sheet MPP or any other capturing and tracking tool like we have a RT, RT which gives a percentage of how much is the text have been gone past and failed we have LDR ray and we have vector cast likewise there are main tools cantata like this.

Let us look into some of the common web base tool easy to use and maintain in general they follow for the applications close to our test link and bugzilla, test link is for test the defect sorry test management in terms of articulating the text metrics and bugzilla is used for defect tracking and defect management purposes these two are these two tools are common tools in email applications and any applications you take.

But these tools are not for testing or testing secretion, but they use for capturing and reporting the metrics also for tracking, so test clips from a sourceforge.net you can go to back website or find out or you can download can use is an open source as long as you are not using for commercial purpose you are free to use that one and download and use it tall test cases test strategy and execution given all that you can put it.

Of course it has a Configuring internal configuration management and you can use it for plugging into the actual configuration also, and you can fine tune the tool as for move that is also one of the good features of that. Maybe in the future session you can see this, I will try to go through the sample test link and AD bug tracking tool bugzilla, so you basically it's the web based test management tool, you need a server and hardware to install and use that term he knows. We will try to see a the basics of

(Refer Slide time: 31:40)



Test link how it will capture and the track? so it's an illustration of the test link, so for example project test project, so this is the basic component in the test link, so test project will have this many parts, now before that the definition of a test link or what it says about different things that we need to use for the tool test case we need, so basically test case describe certain thing task by our steps that means steps can be actions scenario and expected results will be part of that.

So test cases are the fundamental piece of the test link test suit, so we need to have a Test case we need to have a test suit, I will put next one to this each of this probably definition of the test link tool, there's test plan, test project and user, so these are the basic definite of a test link tool and so in that framework we are going to have it, so this framework will are going to have it, so test case the describes the test case through steps.

Steps are action type scenario that suit so bridge gate test suit organize test cases two units will organize multiple test cases and let it suit its structured certain specification into logical parts, the next one being the test plan. So test plan is created when you would like to execute test cases, so test plans can be made up of the test cases from crash project test plan includes builds false forms and any of the binary is equitable environmental contrary the items all this can be probably the part of the test plan,

These are assignments, test results, all the part of the test plan, next one being test project, test project is something that like this till the end in the test link, so till that project is closed the test project will be created in test link, test project I will undergo many different options throughout the test link like them, because we are going to configure and reconfigure it again.

As I said it has an inbuilt term ocean mechanism so we can use that, this project includes the test specification with the test cases requirements and all of the keywords, users within the project have defined also, this framework will be assigned to different users, so users will have their own roles and as per the role, they will keen use update in all that activities they do in the excel, so that the metrics is collaborated all together.

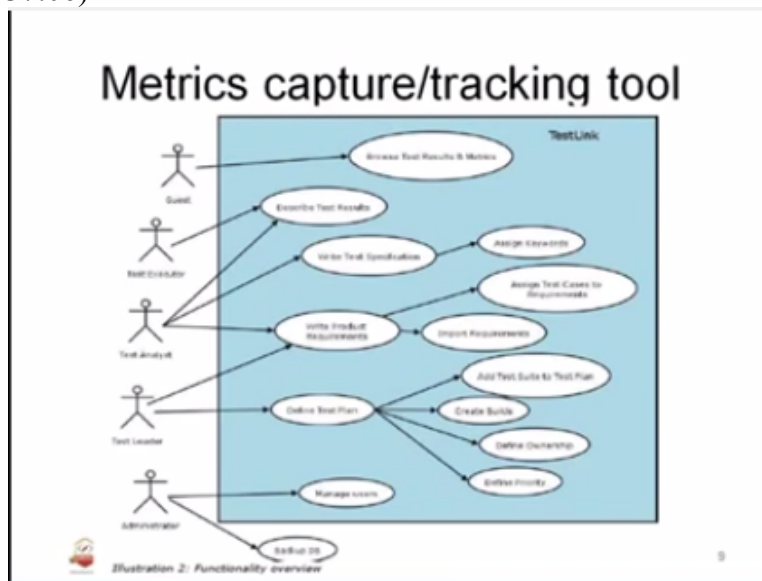
So, next one is being the user, what is the user? so each testing user has a role that defines available test link feature, the user can be administration or other user likewise depending on the role we will have a project team using the test link, so this is at test link 1.6 example what is

being told here, so this from the source forge, see test project has at a test specification requirements test plan and any other customized fields.

And reports on the other hand will be used and we can import and export the compatible elements into these framework depending on the type of import item or we can we form to export the existing one we can do it then the users of this frame work will be using the test link tool with this framework and the he will play around all the specification, requirements, test plan, custom fields and reports.

And he can also attach digital items like in his scripts part of the test cases he can attach within that. So all this will be part of the framework that will be used in test link. So that is the important of our metrics capturing tool and the metrics tracking tourism is basically used for test metrics test cases and test plan and reporting.

(Refer Slide time: 37:06)



So the next one being snapshot of what are the functionality that it has this illustration of test link example, so what are the different roles and functions that the people will use a test link as a whole team, so the users can be a guest, user can be a test executor, user can be analyst, user can be a test lead or a administrator, so in the end of this complete test before work that means basically backs up the complete database of the test link.

Also is responsible for creating home or adding the rights to the users, modifying the contents and managing the users. All this will be responsibility of the administrator, and test leader will define the test plan and he will add test suits to the test plan and he can create builds, define the ownership, define the priority all this will be part of the test leader, and also he can write the requirements or import requirements from the external entity.

And in parallel to test leader there is a test analyst, who can also write and modify the requirements or import the requirements, further he can assign test case of the requirements because test analyst and just leads are very important around people. Who can decide what test cases need to be used for what sort of a requirements? And test analyst also will help in terms of describing the test results and reporting it is also responsible for test specification and assigning the keywords.

Test analyst is also can be called as test case writer or test case developer, first in here, it is up to you how we want to define in your project. Test executor is an independent person according to this you can have a same tester executing the a test ,so basically you will execute and describe the test result, And the guests will have some sort of the wide axis you can browse the test results and metrics .

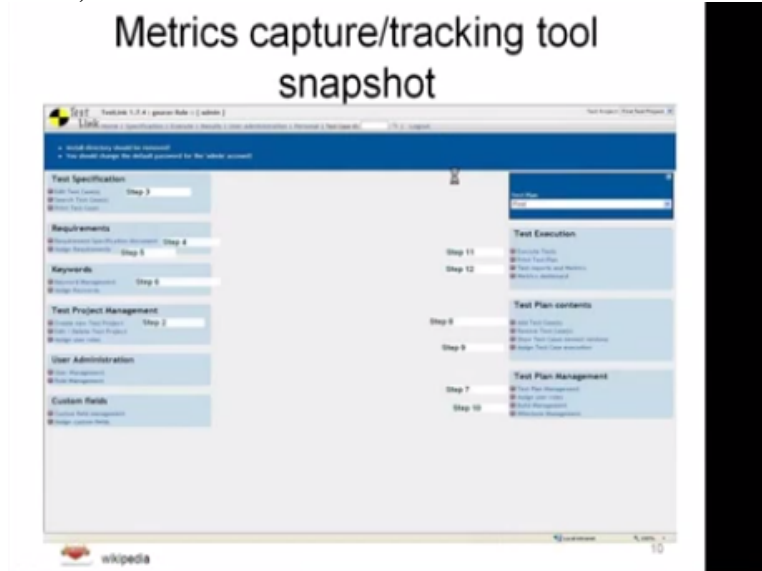
You can basically analyze identify the complete project details basically this can be customers or PM right manager or any other interfaces people, such as QA quality assurance called deletion, we also can be part of the test link depending on the project nature it can be done, so administrator creates the test project test lead will import the requirements from mine or that the member any tool produced one could such as doors from IBM or rectify.

And the tester describes the test scenario and context of the test cases and the alpha test specification and he can also the test link also can create a regression testing aspects and tester can be key in all the test cases and test leader will create this plan and you can add build any script execution mechanisms working so few weeks ago.

Now the builders have developed the build the test link will add those builder and manager can have a guest role as well to see the results of test cases mocking also, the another important aspect of this thing is that traceability in dismissing here it's a very ,very, very important term that has been used in embedded software industry disability form design requirements is a must to be reported when we are done with the testing.

So there are several steps that are evolved for test link usage once all these test links are kidding the execution is carried out and results are logged in and the project is being reported that is how this tool test link is used, the next one being bugzilla.

(Refer Slide time: 42:23)



You can see an example sorry a test link only the tool is snapshot how it looks, you can see whatever we have seen the roles in the previous use case diagram are the functional value here you can see different elements that are displayed here on the left hand side you can see test specification requirements, keywords, test project management, user administration, custom fields etc.


On the right hand side you can see test execution, test plan contains test plan management, each one has its own folders that folders can be customizable and these are the basic framework elements form of part of the test link tone, so upon login these elements can be inspected modified or reported are used so here you can see test link 1.7. 4 on had been has logged in you can view this we can see this and add or delete whatever it is because here's the administrative activities.

So the first step is already is logged in or clear to be project, then the second step you can see the steps also being specified here as an example, a step 2 will be test project management in terms of creating test project editing reediting test project as any of the user roles, step three is the specification all the test cases will be kidding and in addition printing all that will be done, step 4 will have a requirements specification document assigning requirements to be test cases as step five.

Step six will be key words management in terms of crisis ability, step 7 test plan management, so test project management is different that test plan management test plan be should be having the user roles and the environment and all that how builds are managed all this will be part of this, step eight is test plan contents will be kidding, step nine is a test cases exhibition.

Step 10 is milestone management, weight management or release management, 11 into 12 are executing and reporting the tests. So this is simple snapshot of a sample program using test link, like this there are several tools and those tools can be used depending on the project and its complexity that is being applied, so accordingly the reports can be done.

(Refer Slide time: 45:25)



Metrics capture/tracking tool

- **Defect Management (defect Tracker)**
 - The test manager keeps track of the following metrics:
 - . number of open defects per severity category at a times;
 - . number of solved defects in a period per severity category;
 - . total number of raised defects;
 - . number of retests per defect;
 - . total number of retests.

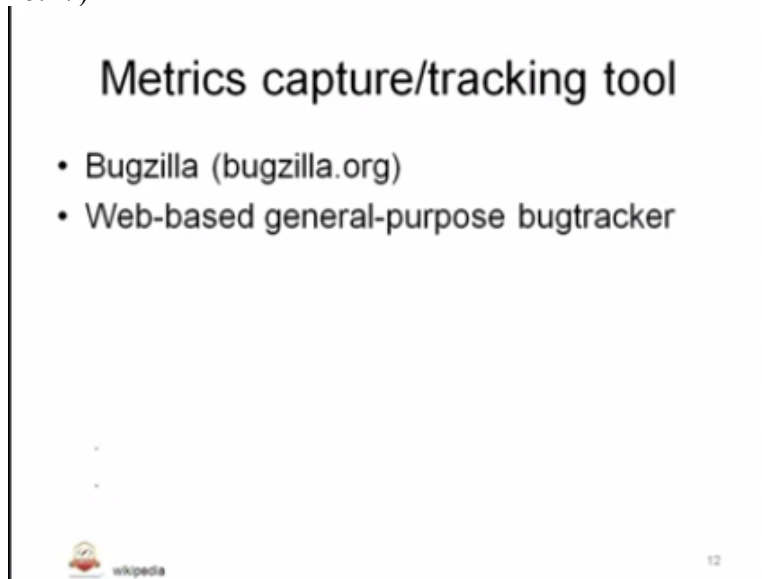
wikipedia 11

So the next one being so it's not a defect it is the test cases and test management and the next one being the defector tracker or the defect management it is also important along with the test group of management, to identify the defects track the defects of closure report it affects to the appropriate stakeholders, so that is why we use the metrics for capturing the defects and tracking the defects, so it is called defect management.

So the test manager keeps track of following metrics of probably pick tracker number of or opened effects per severity category at all times number of salt defects in a period or severity

category. you know the severity of the defects depending on its priority and you will report it, and he will assign for the next fixes, total number of raised effects number of retests or defects of immigration or retest the inscription are one, for defect to close it or report it total number of retests, so all this will be part of the defect rack so test manager basically keep track of this defector tracking.

(Refer Slide time: 46:47)

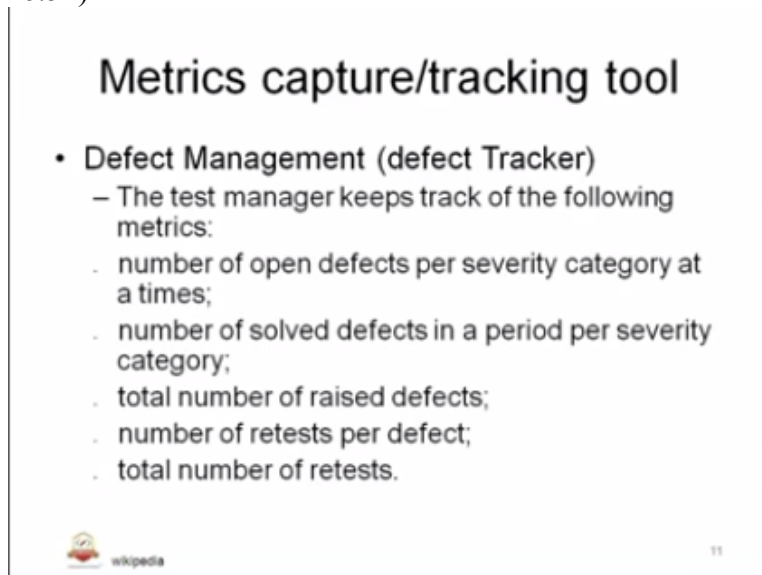


Metrics capture/tracking tool

- Bugzilla (bugzilla.org)
- Web-based general-purpose bugtracker

wikipedia 12

So defect tracking his done,
(Refer Slide time: 46:51)



Metrics capture/tracking tool

- Defect Management (defect Tracker)
 - The test manager keeps track of the following metrics:
 - number of open defects per severity category at a times;
 - number of solved defects in a period per severity category;
 - total number of raised defects;
 - number of retests per defect;
 - total number of retests.

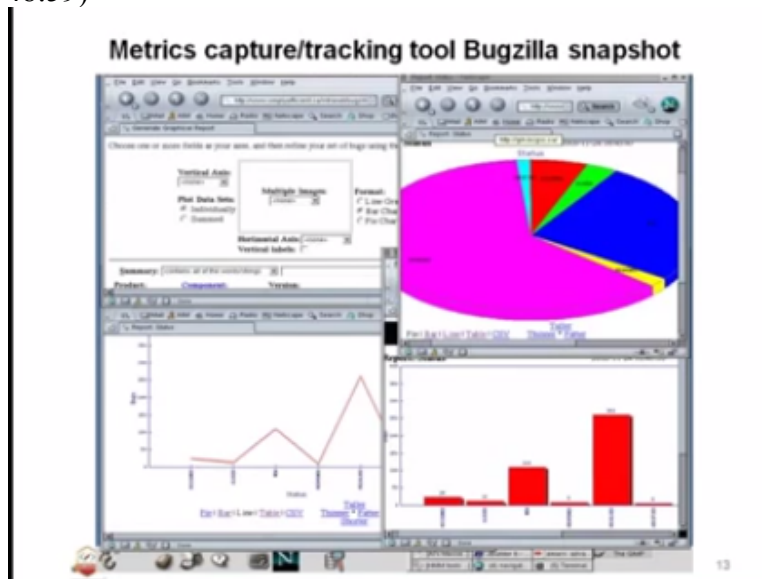
wikipedia 11

With the help of several tools, those tools are basically tied up with the test case tools, because primarily because it can be imported, exported with the help of power tools such as a test link and bugs are also can be reported accordingly. So one good tool this is also a open source tool it is called the bugzilla, this can be referred and used from a particular, this also basically web based on paper tracker.

Where you use an HTML or basic problem to report update and generate the bugs and this has a server client link and all that with the help of web based application this report will be generated to capture the metrics. so that is how bugzilla being used you see very similar to the test link so both can be used in conjunction to generate and reporting testing metrics ,you can see an example of how typically bugzilla is a likely reporting tool, so this is the way that it is being used for vouchering the metrics, and it is also used for tracking on a day-to-day basis or depending on the type of tracking and reporting mechanism.

This also a web based general purpose bug tracker, so based on the HTML and apache server or what all the servers we have been using, and these are open source tool as long as the commercially is not being good it can be used as they open source.

(Refer Slide time: 48:59)



And you can see the next page a snapshot of bugzilla tool there are four windows that is been shown here on my website that I have referred here, and you can choose the way how you want to report a type of image? How you want to report? Windows are there, horizontal axis what you want to report? Vertical axis what you want to plot of the floor? A line chart being used this day to day progress we used for reporting with a number of bugs.

Which are in numbers on the electron more vertical axis the same report is being displayed reported the help of a bar chart you can see on each day how many bugs. so there is the uptrend so on this day is 110 bugs have in report and in 6 on the other day it is so 261 likewise, similarly the bugs can be reported using the pie chart, where you can see a pie symbol having is a slides of resolved in pink color and new bugs in blue color.

Reopened the bogs in yellow closed in green and assigned, verified all this can be reported with alpha bugzilla tracking tone. So this can be effectively used on a date by pinching in terms of reporting, so users re-test lead or who are the people who are still report will collect the data from individuals or individuals also keen as a user in the bugzilla tool.

So that there is no manual intervention much required just they are to open the tool and the key in the data and the report will be generated automatically with couple of clicks, so that is what about the bugzilla.

(Refer Slide time: 51: 13)

ES/T glossary	
<i>Test set</i>	A collection of test cases.
<i>Test design technique</i>	A standardized method of deriving test cases from a test basis.
<i>Test strategy</i>	A description of the relative importance of the system parts and quality attributes leading to decisions about desired coverage, techniques to be applied, and resources to be allocated.
<i>Test team</i>	A group of people responsible for executing all the activities described in a test plan.
<i>Test technique</i>	A standard description of how to execute a certain test activity.
<i>Test tool</i>	An automated aid that supports one or more test activities, such as planning and control, specification, building initial files and data, test execution, and test analysis.
<i>Test type</i>	A group of test activities aimed at checking a system on a number of related quality characteristics.
<i>Test unit</i>	A set of processes, transactions and/or functions which are tested collectively.
<i>Testability review</i>	The detailed evaluation of the testability of a test basis.

So with that we come to the end of a test metrics capturing, now we will go through some of the glossary but we were there using so far or we are going to use it this again from the software test book embed software test testing boot just, set a collection of test cases test design technique a standardized method of deriving this case inform open spaces, test strategy we know that a description of the related importance of the system parts.

And quality attributes leading to this change about desired coverage, techniques to be applied and resources are to be located. Test team the group of people responsible for we could evolve work with this the square will be described in the test plan, testing technique a standard description of how to coupon certain tests activity, test tool and automated the aid that supports one or more perceptivities.

Such as a planning and control and specification building shell pines and data test execution and test analysis, test type a group of a test activities in by checking the system on number of a people quality characteristics test unit a set of processors transactions and functions will focus to collectively this ability review the detailed evaluation of the testability of test basis,

(Refer Slide time: 53:05)

ES/T glossary

<i>Test type</i>	A group of test activities aimed at checking a system on a number of related quality characteristics.
<i>Test unit</i>	A set of processes, transactions and/or functions which are tested collectively.
<i>Testability review</i>	The detailed evaluation of the testability of a test basis.
<i>Testing</i>	The process of planning, preparation, execution and analysis aimed at establishing the quality level of a system.
<i>Testware</i>	All products produced as a result of a test project.
<i>Unit test</i>	The testing of individual software components.
<i>White-box testing</i>	Test design techniques that derive test cases from the internal properties of an object, using knowledge of the internal structure of the object.

Test type, test unit, testability we have seen, we can see testing a process of planning preparation execution analysis aimed at establishing the quality level of a system test where all products produced as a result of a positive square. Unit testing is testing of individual software components white box testing the test isn't techniques that derive our test cases from internal properties of an object using knowledge of the internal structure of the object. So with that we have completed the unit three test metrics, static analysis and code reviews, we will before we end, we will just a recap of unit three.

(Refer Slide time: 53:46)

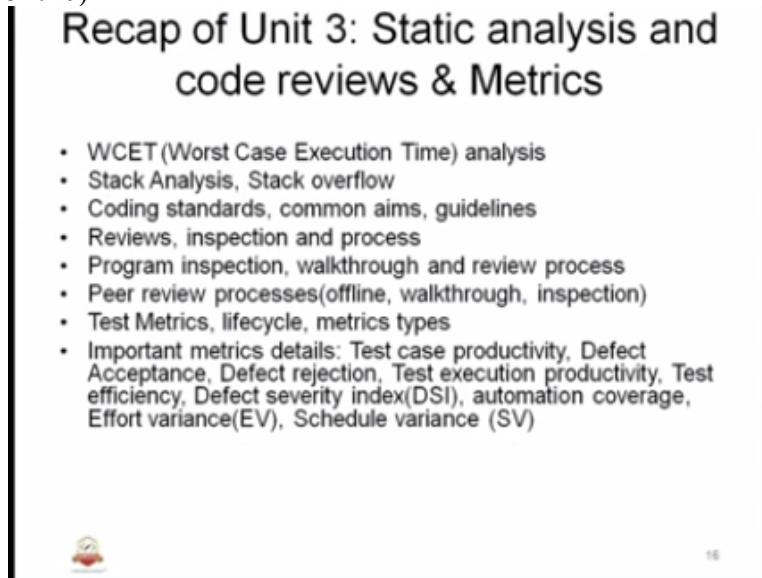
Recap of Unit 3: Static analysis and code reviews & Metrics

- Static testing – definition, static vs dynamic testing
- Static analysis, reviews, inspection and the test processes
- Control coupling/flow analysis and data coupling
- SW complexity, McCabe SW complexity
- Static analysis tools : Understand for C/C++, MISRA rule checker..

So what are the unit 3 items that we have covered so far, so we have covered static testing this is definition static versus dynamic testing static analysis reviews inspection and the test processes control coupling and flow data coupling control coupling is also nothing but and will control flow and flow analysis, then we had gone through software complexity the formula are using in MAC away complexity make a base after complexity.

So we took a few examples and with the help of a static analysis tool understand because to hitting space or miss for its return etc we generated a softer complexity, and static analysis reporting parts then,

(Refer Slide time: 54:40)



The slide is titled "Recap of Unit 3: Static analysis and code reviews & Metrics". It contains a bulleted list of topics covered in the unit. At the bottom left, there is a small logo of a person with arms raised, and at the bottom right, the number "16" is visible.

Recap of Unit 3: Static analysis and code reviews & Metrics

- WCET (Worst Case Execution Time) analysis
- Stack Analysis, Stack overflow
- Coding standards, common aims, guidelines
- Reviews, inspection and process
- Program inspection, walkthrough and review process
- Peer review processes (offline, walkthrough, inspection)
- Test Metrics, lifecycle, metrics types
- Important metrics details: Test case productivity, Defect Acceptance, Defect rejection, Test execution productivity, Test efficiency, Defect severity index (DSI), automation coverage, Effort variance (EV), Schedule variance (SV)

We had gone through some of the worst case the execution analysis or aspects and tools that are used in terms in terms of reporting the definition worst-case execution time and we had seen a stack analysis and stack overflow, coding standards common objectives guidelines, reviews inspection process, program inspection, program walkthrough and review process peer review processes, what are the types of review process through offline walkthrough inspection?

And test metrics we saw in the today's initiative class, life cycle metrics types, or depth of different metric types and metrics life cycle, test metrics and reporting it and we also touch based important metrics details, test case productivity, defect acceptance, defect rejection, test execution productivity, test efficiency, defect severity index, automation coverage, effort variance, schedule variances for these are some of the important metrics that we had spoken, so with that we come to the end of unit, so we will start the next unit which is nothing but the software integration in the next session.