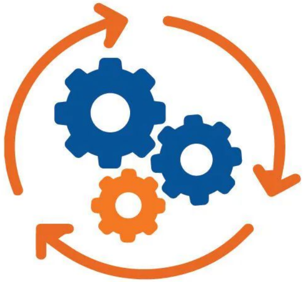
Chapter 9

Test Execution



Lecturer: Nguyen Thanh Quan (MEng)
Email: tg_nguyenthanhquan_cntt@tdtu.edu.vn

Content

1. **Executing and Evaluating Test Phases**

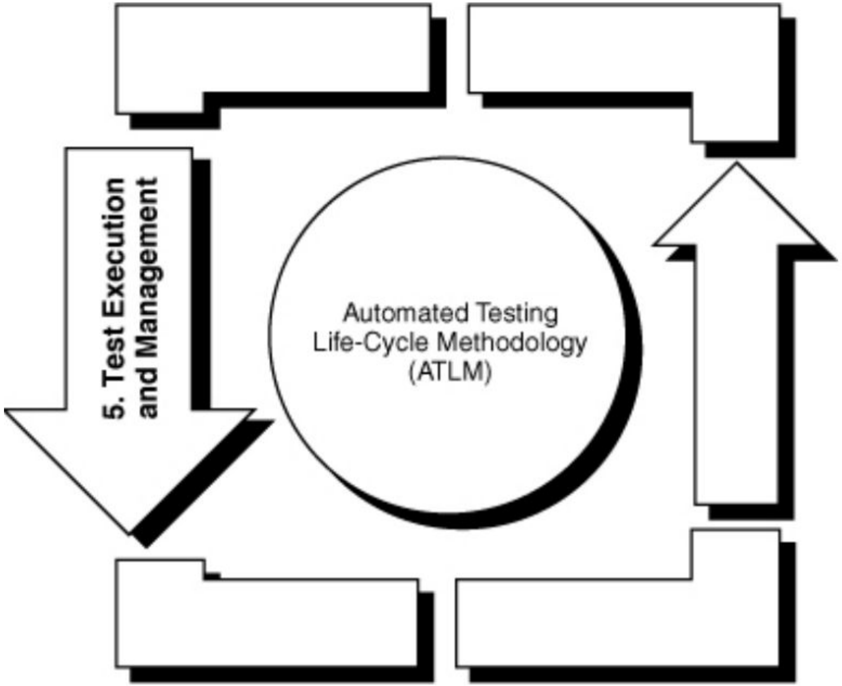
2. **Defect Tracking and New Build Process**

3. **Test Program Status Tracking**

Introduction

- With the test plan in hand and the test environment now operational, it is **time to execute** the tests defined for the test program.
- After test execution ends, the test **results** need to be **evaluated**.
- These procedures describe the steps that should be completed after a test has been executed.
- The test team needs to **collect** and **analyze** various **measurements**.

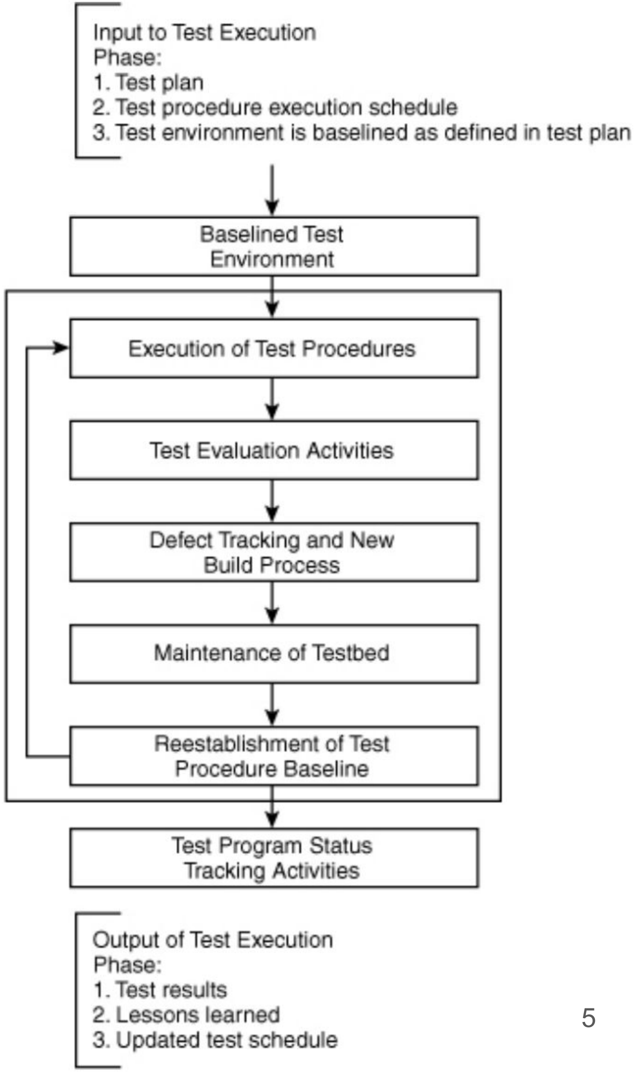
Executing the test plan.


5. Test Execution and Management
Automated Testing Life-Cycle Methodology (ATLM)

Executing and Evaluating Test Phases

Executing and Evaluating Test Phases

- The **test team is ready** to execute and evaluate the test procedures, as defined in the test plan for each of the various test phases.
- **The primary input** for each test phase is the associated suite of test procedures.
- The output of each test phase consists of the achieved or modified **acceptance criteria**, as defined in the test plan.
- The test team needs to **clearly identify** all required test program **documentation**.
- Test program documentation may need to comply with specific industry or regulatory standards.



Input to Test Execution Phase:
1. Test plan
2. Test procedure execution schedule
3. Test environment is baselined as defined in test plan

Baselined Test Environment

Execution of Test Procedures

Test Evaluation Activities

Defect Tracking and New Build Process

Maintenance of Testbed

Reestablishment of Test Procedure Baseline

Test Program Status Tracking Activities

Output of Test Execution Phase:
1. Test results
2. Lessons learned
3. Updated test schedule

Executing and Evaluating Test Phases

Unit Test Execution and Evaluation

- Unit tests should be **performed in accordance with the test plan.**
- Unit tests should **remain consistent** with the detailed development schedule.
- Test procedures should **consist of input and expected results** to facilitate an automated results checkout process.
- At the **white-box** testing level, test procedures **focus on the smallest collection of code** that can be usefully tested => unit testing requires a detailed understanding of the code.

Executing and Evaluating Test Phases

Unit Test Execution and Evaluation

- An individual other than the **developers** should execute tests on the particular unit of code because developers are often **blind to their mistakes.**
- During unit testing, **static analysis** can be performed.
- During the unit testing phase, **code profiling can be performed.**
- Most code development test tools come with their own **debuggers**.
- When unit testing uncovers problems, such **defects** need to be **documented** and **tracked**.

Executing and Evaluating Test Phases

Unit Test Execution and Evaluation

Unit Testing Evaluation Criteria List:

- **UT1**: Does the code **meet** the design **specifications**?
- **UT2**: For each conditional **statement**, does the condition execute correctly?
- **UT3**: Have the tests exercised the unit over full range of operational conditions that the unit of software is **expected** to address?
- **UT4**: Do all **exceptions** work **correctly**?
- **UT5**: Are **errors firing correctly**?
- **Code coverage:** Have all **statements** been **covered at least once?**

Executing and Evaluating Test Phases

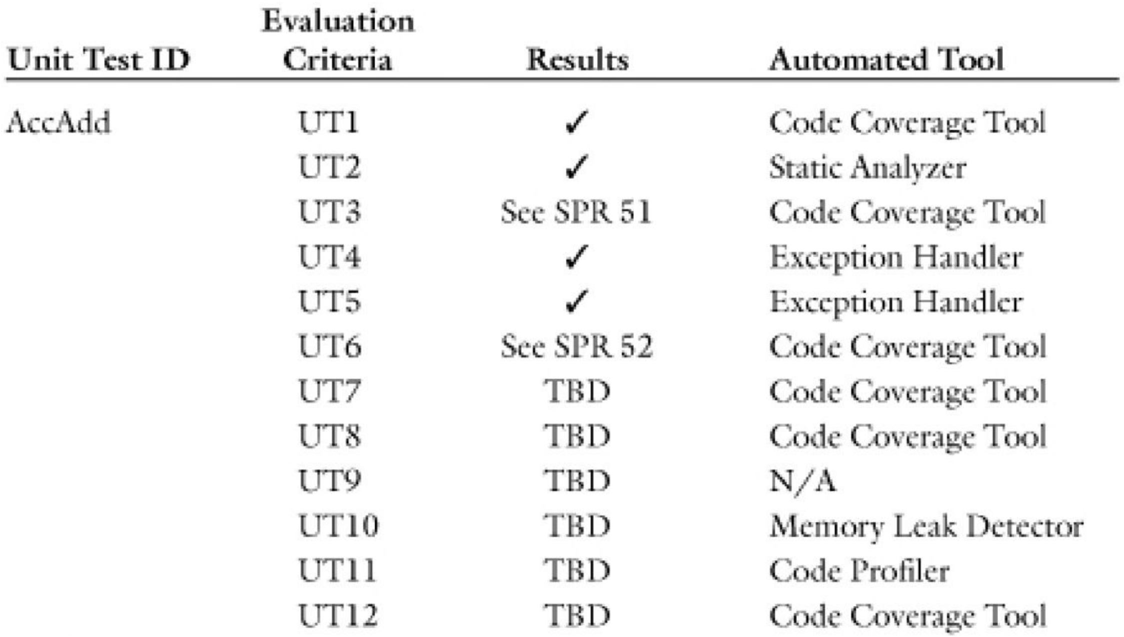
Unit Test Execution and Evaluation

Unit Testing Evaluation Criteria List:

- **Code coverage:** Has each conditional **statement** been exercised **at least once** by the tests?
- **Code coverage:** Have all **boundary** cases been exercised?
- Were any design **assumptions** made about the operation of this unit? Did the tests **demonstrate** the **assumptions**?
- Did the code **pass the memory leakage test?**
- Did it **pass the code profiling test?**
- Have all **control paths of critical units** been **exercised successfully** by the test data?

Executing and Evaluating Test Phases

Unit Test Execution and Evaluation

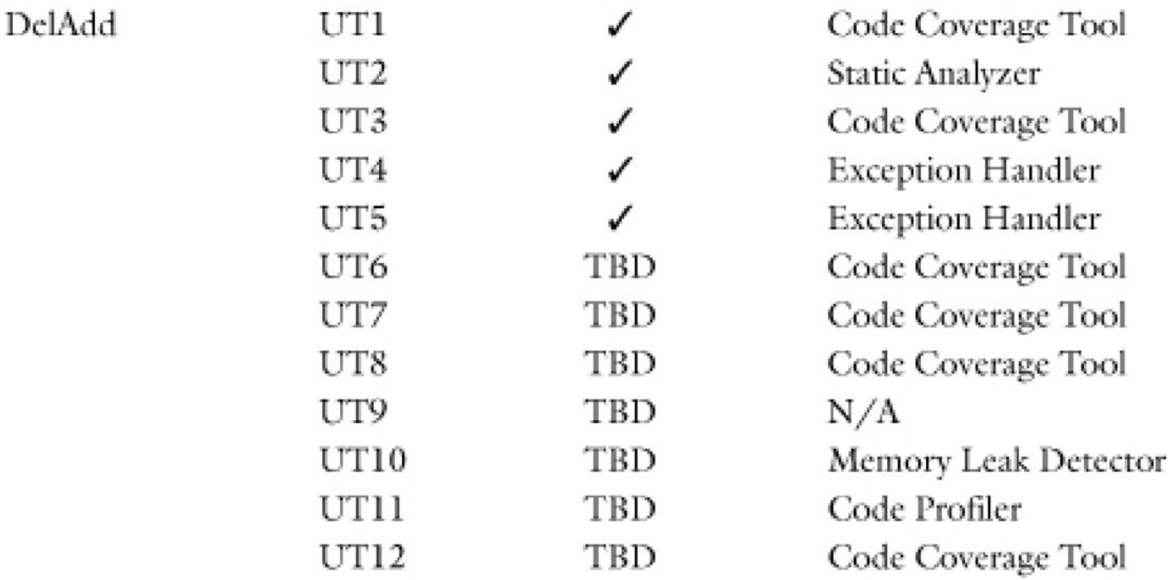


Unit Test ID	Evaluation Criteria	Results	Automated Tool
AccAdd	UT1	✓	Code Coverage Tool
	UT2	✓	Static Analyzer
	UT3	See SPR 51	Code Coverage Tool
	UT4	✓	Exception Handler
	UT5	✓	Exception Handler
	UT6	See SPR 52	Code Coverage Tool
	UT7	TBD	Code Coverage Tool
	UT8	TBD	Code Coverage Tool
	UT9	TBD	N/A
	UT10	TBD	Memory Leak Detector
	UT11	TBD	Code Profiler
	UT12	TBD	Code Coverage Tool

Unit Test Evaluation Report

Executing and Evaluating Test Phases

Unit Test Execution and Evaluation



DelAdd	UT1	✓	Code Coverage Tool
	UT2	✓	Static Analyzer
	UT3	✓	Code Coverage Tool
	UT4	✓	Exception Handler
	UT5	✓	Exception Handler
	UT6	TBD	Code Coverage Tool
	UT7	TBD	Code Coverage Tool
	UT8	TBD	Code Coverage Tool
	UT9	TBD	N/A
	UT10	TBD	Memory Leak Detector
	UT11	TBD	Code Profiler
	UT12	TBD	Code Coverage Tool

Unit Test Evaluation Report

Executing and Evaluating Test Phases

Integration Test Execution and Evaluation

- **Integration** testing can be conducted **either by developers or by the test group**, depending upon the decision made during test planning with regard to the allocation of funding for test activities.
- Integration testing resembles system testing, but **concentrates** on the **application internals** more than system testing does.
- During integration testing, **units** are **incrementally integrated** and **tested together** based upon control flow.

Executing and Evaluating Test Phases

Integration Test Execution and Evaluation

- The development team must **generate software fixes** to resolve problem reports, and integration test procedures subsequently need to be refined.
- When the test team takes responsibility for executing integration tests, the test engineers can **enhance developer’s understanding** of system and software problems and help replicate a problem when necessary.
- Test engineers may participate in engineering **review** boards, as applicable, to review and discuss outstanding defect reports => **mitigate** defect reports, verify closure of the problems with regression tests, reused with baseline.

Executing and Evaluating Test Phases

Integration Test Execution and Evaluation

- **After integration testing ends**, the test team **prepares** a **report** that summarizes test activities and evaluates test results.
- **End-user approval** of the test report constitutes the conclusion of unit and integration testing.

Executing and Evaluating Test Phases

System Test Execution and Evaluation

- System testing is another form of integration testing, albeit one **conducted at a higher level.**
- The test engineer examines the integration of parts, which make up the entire system.
- **Performed by a separate test team** that implements the test procedure execution schedule and the system test plan.
- **May a large number of individual test procedures** to verify all necessary combinations of input, process rules, and output associated with a program function.

Executing and Evaluating Test Phases

System Test Execution and Evaluation

- **False Negatives**
  - **Meaning that the test failed even though no problem exists with the AUT.**
  - If the actual **result differs from the expected result,** the delta (that is, the discrepancy) must be further diagnosed.
  - A failed test procedure does not necessarily indicate a problem with the AUT.

Executing and Evaluating Test Phases

System Test Execution and Evaluation

- **False Negatives can be caused by**
  - A necessary change in the application.
  - Test setup errors.
  - Test procedure errors.
  - User errors.
  - Automated test script logic errors.
  - Test environment setup errors (e.g wrong versions).

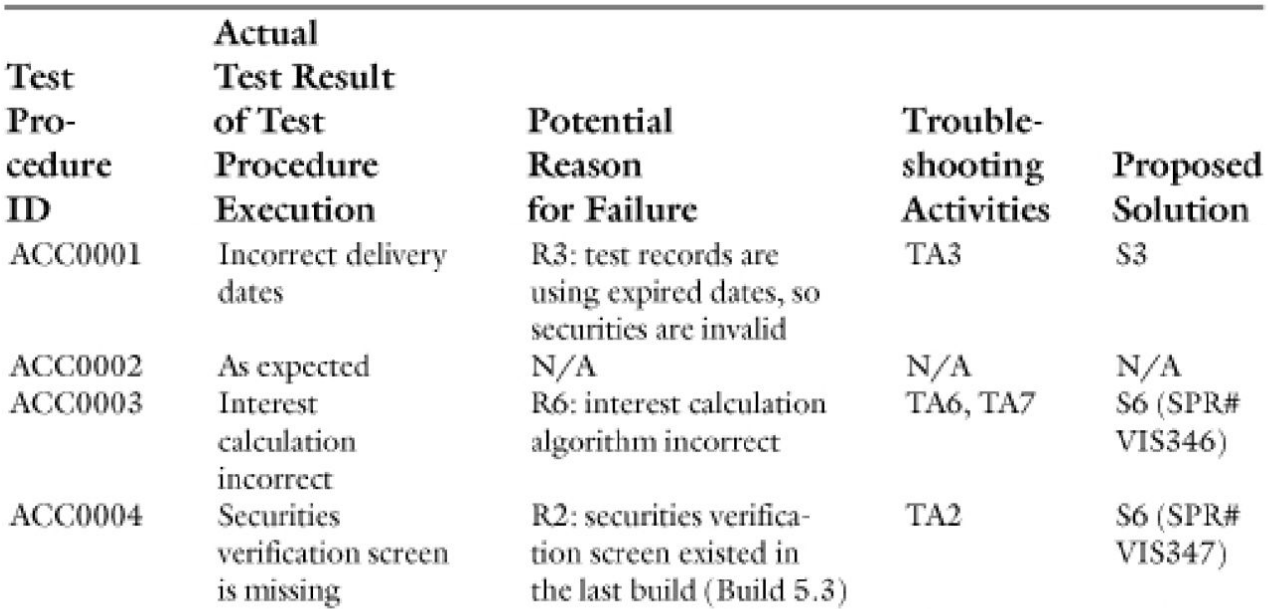
Executing and Evaluating Test Phases

System Test Execution and Evaluation

- **False Negatives**
  - The test team **needs** to be able to **replicate** the **problem** and must ensure that the **problem** did **not result from a user error.**
  - Test engineers must ensure that the **problem does not result from a procedural error.**
  - E.g a screen menu item may have been changed based upon user input, but the automated script, which was created during a previous software version => no problem with AUT.

Executing and Evaluating Test Phases

System Test Execution and Evaluation

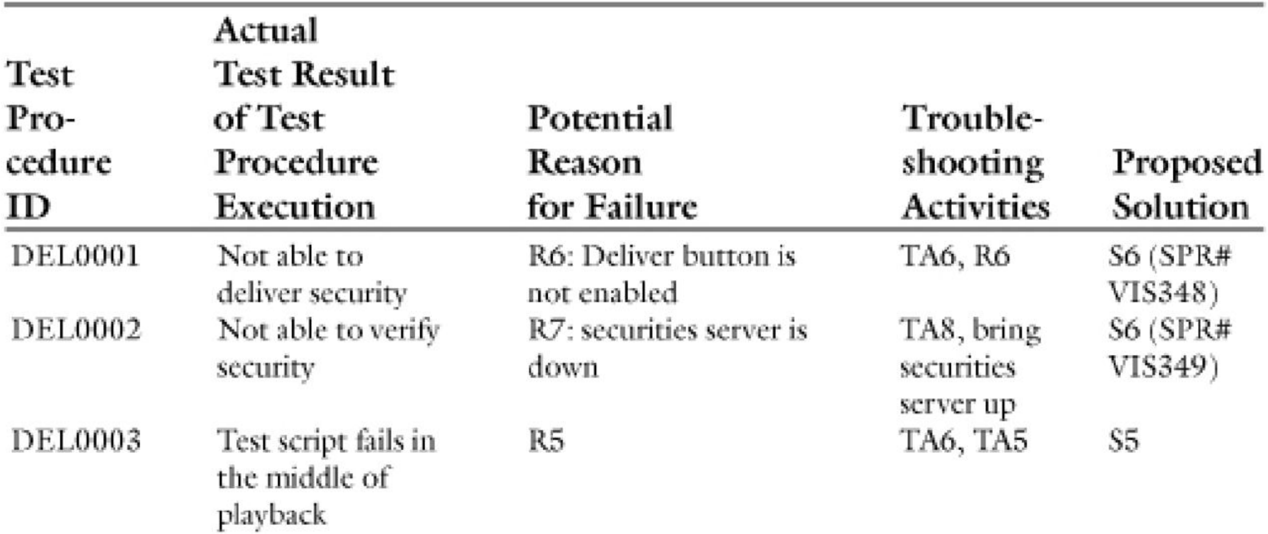


Test Procedure ID	Actual Test Result of Test Procedure Execution	Potential Reason for Failure	Troubleshooting Activities	Proposed Solution
ACC0001	Incorrect delivery dates	R3: test records are using expired dates, so securities are invalid	TA3	S3
ACC0002	As expected	N/A	N/A	N/A
ACC0003	Interest calculation incorrect	R6: interest calculation algorithm incorrect	TA6, TA7	S6 (SPR# VIS346)
ACC0004	Securities verification screen is missing	R2: securities verification screen existed in the last build (Build 5.3)	TA2	S6 (SPR# VIS347)

Test Outcome Evaluation Activities

Executing and Evaluating Test Phases

System Test Execution and Evaluation



Test Procedure ID	Actual Test Result of Test Procedure Execution	Potential Reason for Failure	Troubleshooting Activities	Proposed Solution
DEL0001	Not able to deliver security	R6: Deliver button is not enabled	TA6, R6	S6 (SPR# VIS348)
DEL0002	Not able to verify security	R7: securities server is down	TA8, bring securities server up	S6 (SPR# VIS349)
DEL0003	Test script fails in the middle of playback	R5	TA6, TA5	S5

Test Outcome Evaluation Activities

Executing and Evaluating Test Phases

System Test Execution and Evaluation

- **False Positives**
  - A test procedure appears to have **executed successfully** but a **problem actually exists** with the AUT => **needs to stay alert.**
  - Even when test execution results **match the expected results**, the test team must ensure that **the results are not based upon false positives.**
  - If the expected test result **does not match the actual test result**, because of a problem in the AUT rather than because of a false positive or a false negative, the test team needs to create a software problem report to document the defect.

Executing and Evaluating Test Phases

Test Results Analysis of Regression Tests

- When the test group receives a new application baseline, **build release notes should accompany the new build.**
- The release notes should **address all new functionality** additions and defects that were fixed.
- A **smoke test** should be executed to **verify that the major functionality.**
- If **discrepancies** are found, then the new build should **not be accepted.**
- When a smoke test **passes**, the new build is **accepted** for system testing and incremental regression testing is performed.

Executing and Evaluating Test Phases

Test Results Analysis of Regression Tests

- **Regression** tests should be performed against **both modified code** as well as code which **was not changed**, but potentially could have been affected by the change.
- If a large number of **errors associated with functionality that previously worked** => **developers** may have been **careless**.
- The test team needs to identify other functional areas related to the errors.
- If developers indicate that a particular functional area **is now fixed**, but **regression** testing **uncovers problems** for the particular software => **check** the **environment** or **poor implementation**.
- Need to consider **reallocation** of test engineer effort and reassessment of application risk allocation.

Executing and Evaluating Test Phases

User Acceptance Test Execution and Evaluation

- The test team may need to perform **a user acceptance test (UAT)** that **involves end-user participation.**
- The UAT commonly consists of a subset of the suite of tests performed at the system test level => defined and communicated to the **customer or end user for approval => in a defined test environment.**

Defect Tracking and New Build Process

Defect Tracking and New Build Process

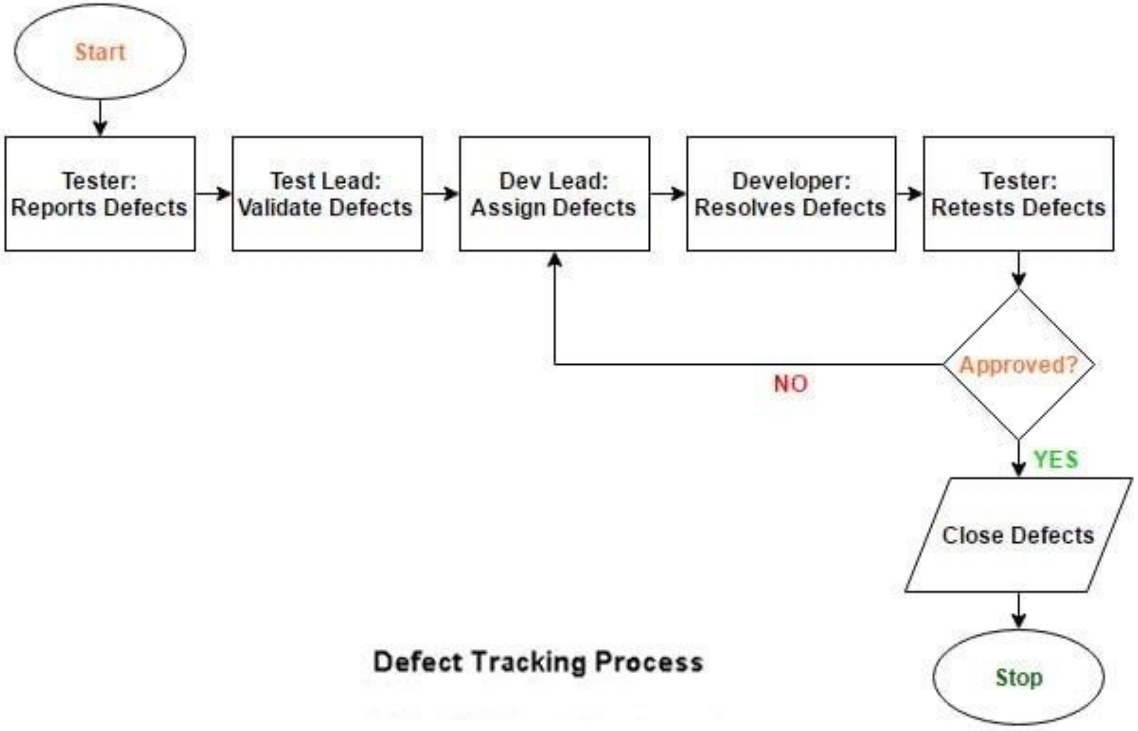
- **Test engineers** will need to **help developers understand and replicate system and software problems**, when necessary.
- Each defect is commonly **classified in a range** of 1 to 4 based upon degree of priority.
- Test engineers need to **participate in discussions on defect reports.**
- The test engineer must **assess the importance** of the solution to the successful operation of the system.
- **High-priority defects need to be fixed soon and vice versa.**

Defect Tracking and New Build Process

A common classification of defect priority levels follows:

- **Fatal**: Operation of the application is interrupted, and testing cannot be continued.
- **High priority**: A significant problem, but the application is still operational.
- **Medium priority**: The problem has little impact on operation of the application.
- **Low priority**: The problem has no impact on the operation of the application.

Defect Tracking and New Build Process



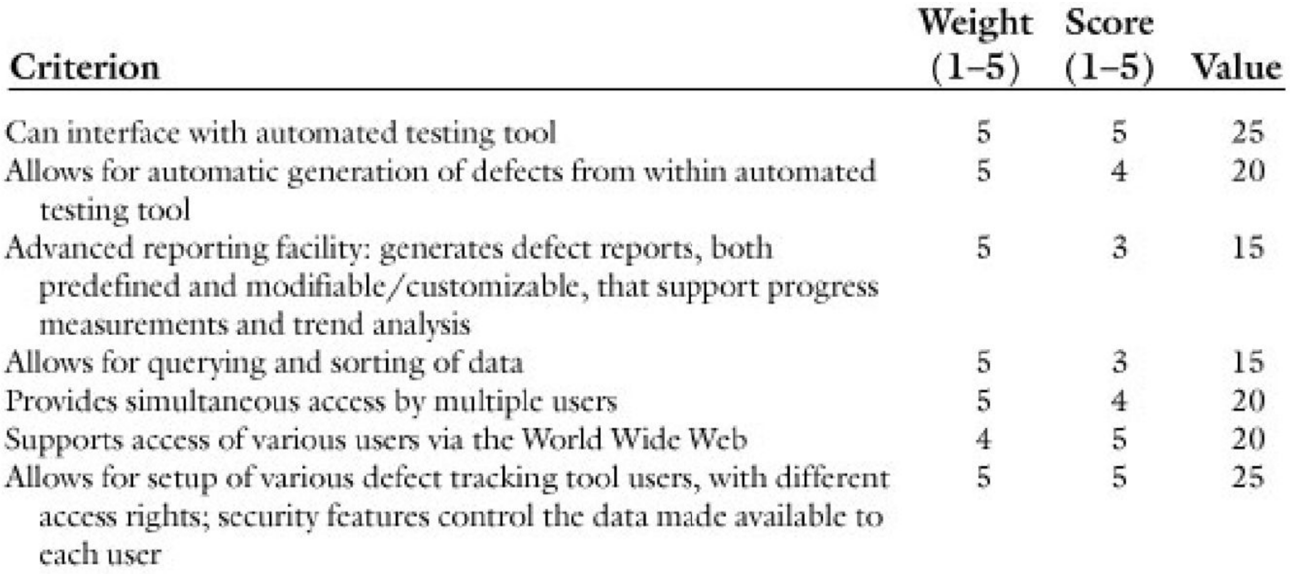
Defect Tracking Procedure

Defect Tracking and New Build Process

An automated defect tracking tool helps to ensure that reported defects receive the proper attention. The tool should be able to perform the following tasks:

- **Identify** the priority of a defect.
- **Assign** a unique identifier to each defect.
- **Link** each defect to the applicable test procedure as well as to a particular application build.
- **Log** the date on which the defect was reported.
- **Log** the date on which the defect was assigned to an application developer.
- **Log** the date on which the defect was updated.
- **Identify** the developer assigned to the defect.
- **Identify** the test engineer who reported the defect.
- **Log** and track the status of the defect, including values such as new, open, assigned, fixed, retest, and closed.

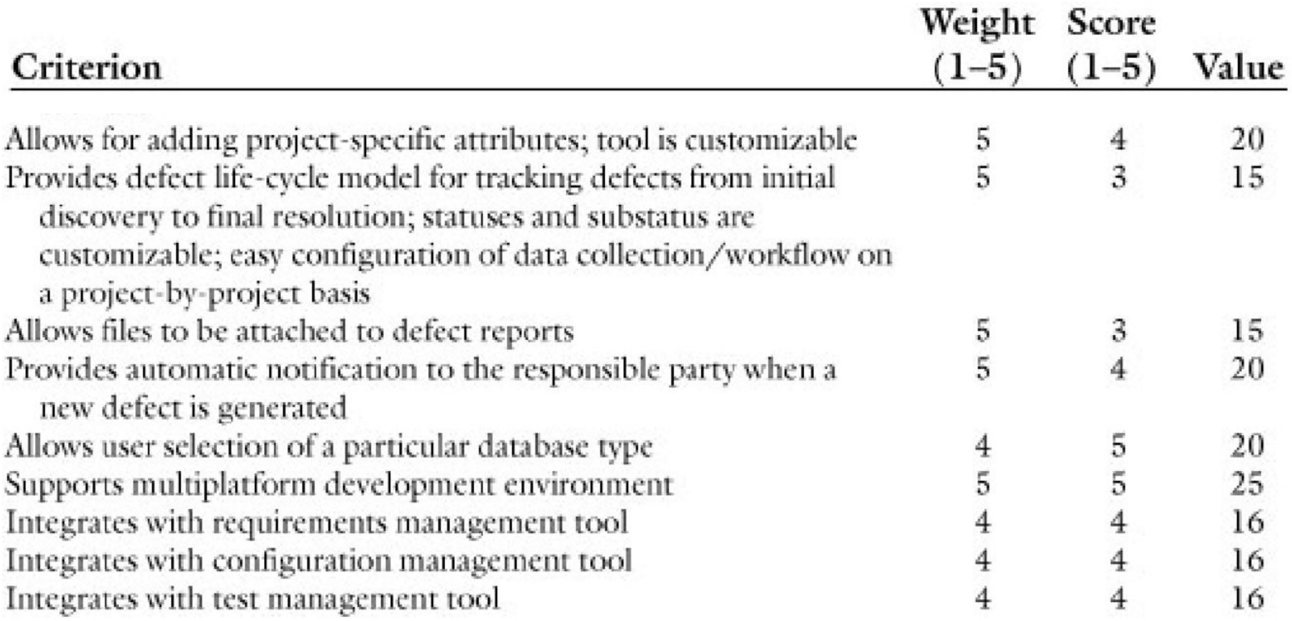
Defect Tracking and New Build Process



Criterion	Weight (1–5)	Score (1–5)	Value
Can interface with automated testing tool	5	5	25
Allows for automatic generation of defects from within automated testing tool	5	4	20
Advanced reporting facility: generates defect reports, both predefined and modifiable/customizable, that support progress measurements and trend analysis	5	3	15
Allows for querying and sorting of data	5	3	15
Provides simultaneous access by multiple users	5	4	20
Supports access of various users via the World Wide Web	4	5	20
Allows for setup of various defect tracking tool users, with different access rights; security features control the data made available to each user	5	5	25

Defect Tracking Tool Evaluation Criteria

Defect Tracking and New Build Process



Criterion	Weight (1–5)	Score (1–5)	Value
Allows for adding project-specific attributes; tool is customizable	5	4	20
Provides defect life-cycle model for tracking defects from initial discovery to final resolution; statuses and substatus are customizable; easy configuration of data collection/workflow on a project-by-project basis	5	3	15
Allows files to be attached to defect reports	5	3	15
Provides automatic notification to the responsible party when a new defect is generated	5	4	20
Allows user selection of a particular database type	4	5	20
Supports multiplatform development environment	5	5	25
Integrates with requirements management tool	4	4	16
Integrates with configuration management tool	4	4	16
Integrates with test management tool	4	4	16

Defect Tracking Tool Evaluation Criteria

Defect Tracking and New Build Process

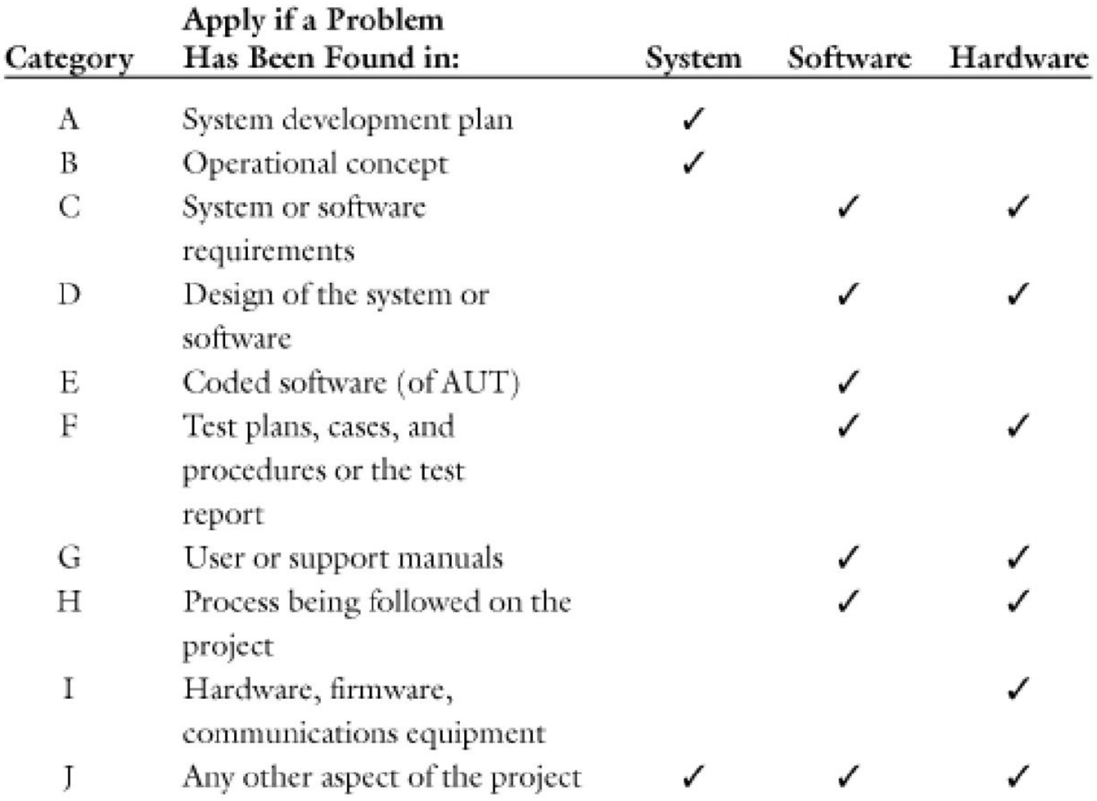
- The test management tool should **permit the automatic validation** of as many of the test results as possible.
- The **sequential order of the transactions may vary,** and the transactions occurring prior to the test may affect the results.

=> The test results could be evaluated by querying the database directly using SQL statements and then comparing the query results and the application generated results to the expected results.

Defect Tracking and New Build Process

- Most automated test tools **maintain** test **results** and **generate reports**:
  - pass or fail status.
  - the name of each test.
  - the start and end times for each test execution.
- The **more test result attributes** that can be **documented** by a test tool, the **more information** that the test engineer can **use to analyze results.**

Defect Tracking and New Build Process



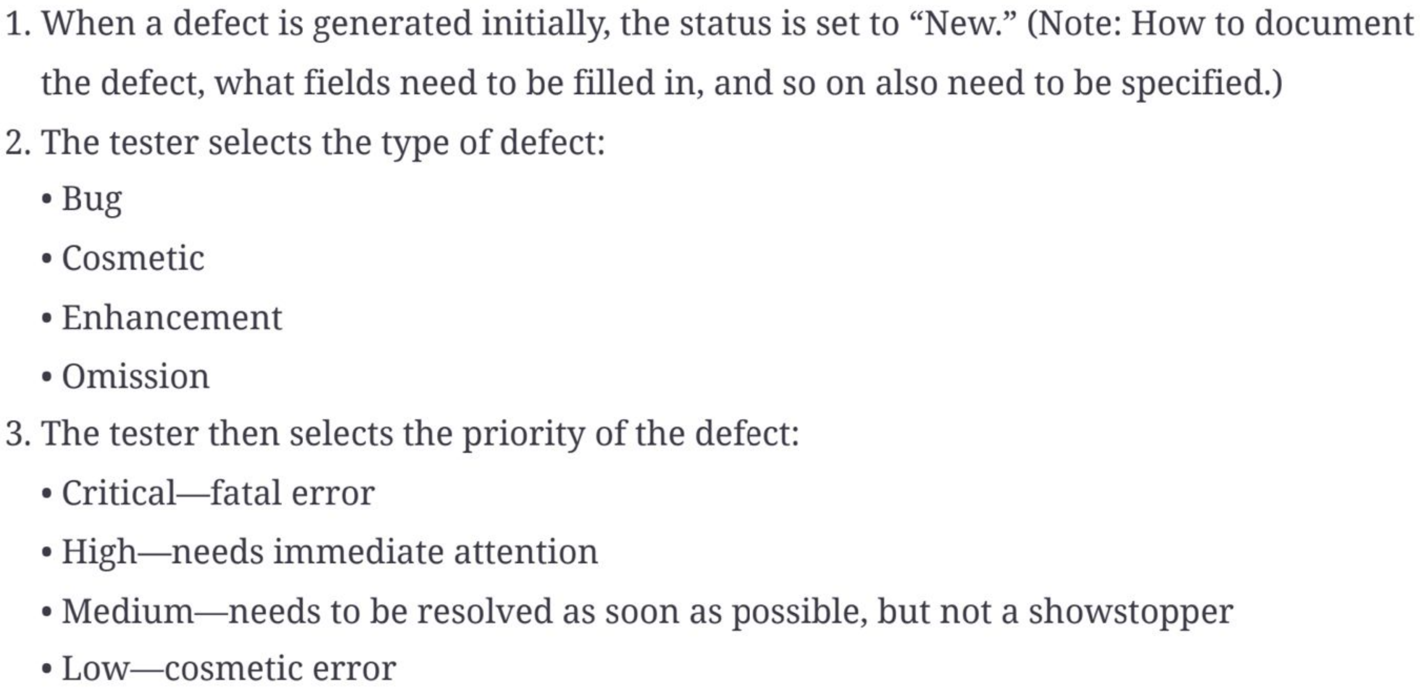
Category	Apply if a Problem Has Been Found in:	System	Software	Hardware
A	System development plan	✓		
B	Operational concept	✓		
C	System or software requirements		✓	✓
D	Design of the system or software		✓	✓
E	Coded software (of AUT)		✓	
F	Test plans, cases, and procedures or the test report		✓	✓
G	User or support manuals		✓	✓
H	Process being followed on the project		✓	✓
I	Hardware, firmware, communications equipment			✓
J	Any other aspect of the project	✓	✓	✓

Defect Tracking and New Build Process

Defect Life-Cycle Model

- When using a defect tracking tool, the test team will need to define and **document** the **defect life-cycle model**, also called the **defect workflow**.
- In some organizations, **the configuration management group** takes responsibility for the defect life cycle.
- In other organizations, it is a **test team responsibility.**

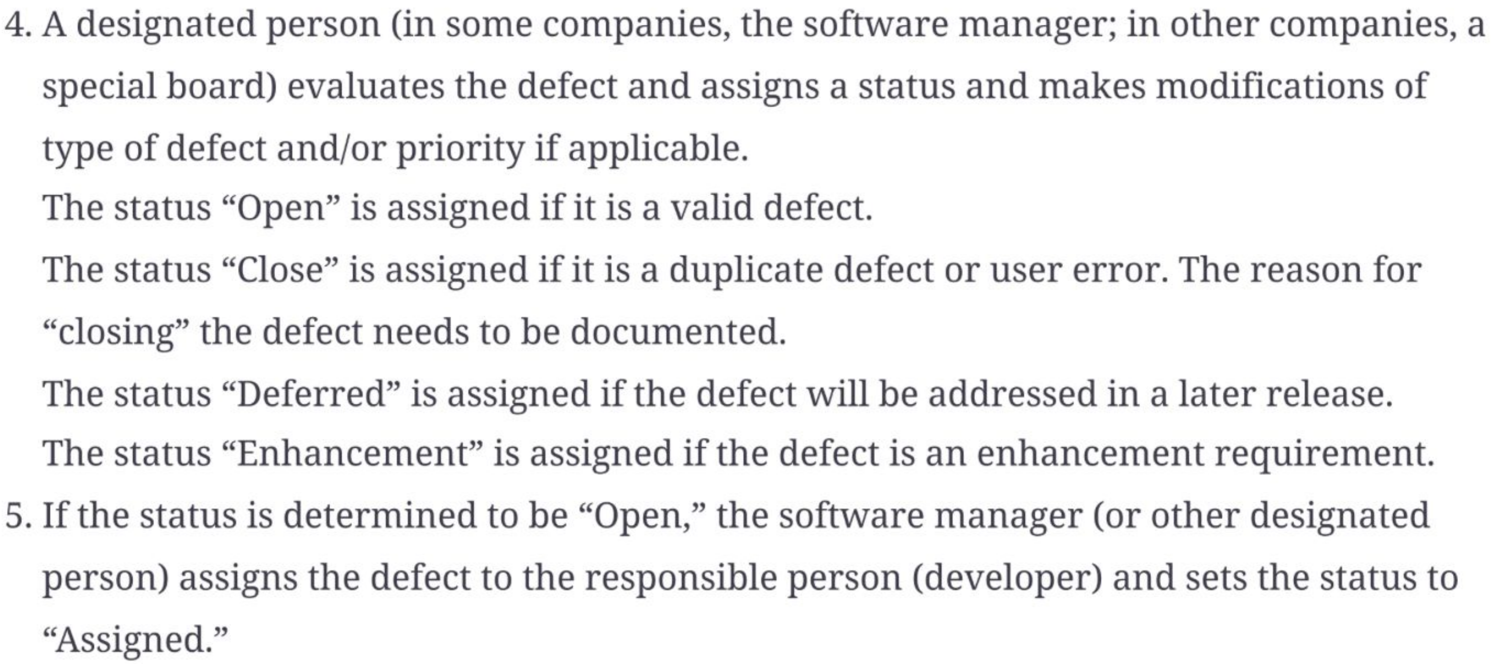
Defect Tracking and New Build Process



1. When a defect is generated initially, the status is set to "New." (Note: How to document the defect, what fields need to be filled in, and so on also need to be specified.)
2. The tester selects the type of defect:
   - Bug
   - Cosmetic
   - Enhancement
   - Omission
3. The tester then selects the priority of the defect:
   - Critical—fatal error
   - High—needs immediate attention
   - Medium—needs to be resolved as soon as possible, but not a showstopper
   - Low—cosmetic error

Defect Life-Cycle Model

Defect Tracking and New Build Process



4. A designated person (in some companies, the software manager; in other companies, a special board) evaluates the defect and assigns a status and makes modifications of type of defect and/or priority if applicable.

   The status “Open” is assigned if it is a valid defect.

   The status “Close” is assigned if it is a duplicate defect or user error. The reason for “closing” the defect needs to be documented.

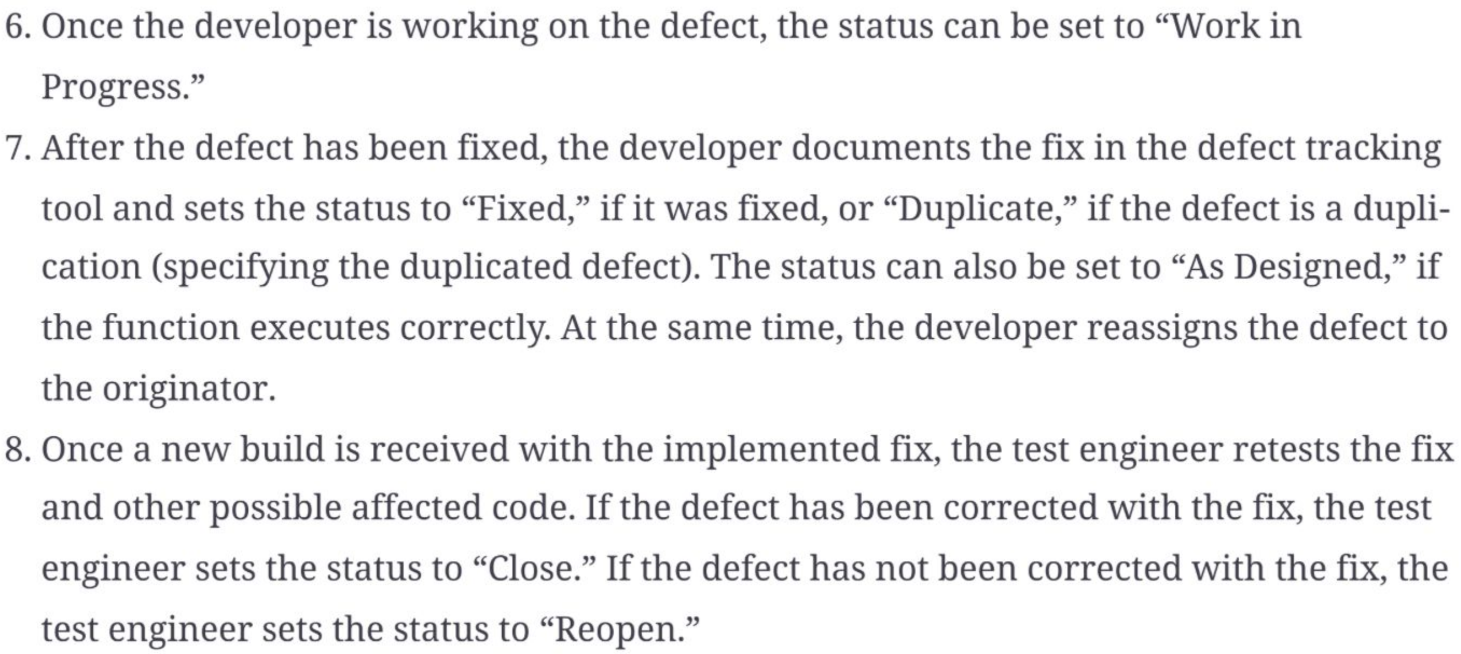
   The status “Deferred” is assigned if the defect will be addressed in a later release.

   The status “Enhancement” is assigned if the defect is an enhancement requirement.

5. If the status is determined to be “Open,” the software manager (or other designated person) assigns the defect to the responsible person (developer) and sets the status to “Assigned.”

Defect Life-Cycle Model

Defect Tracking and New Build Process

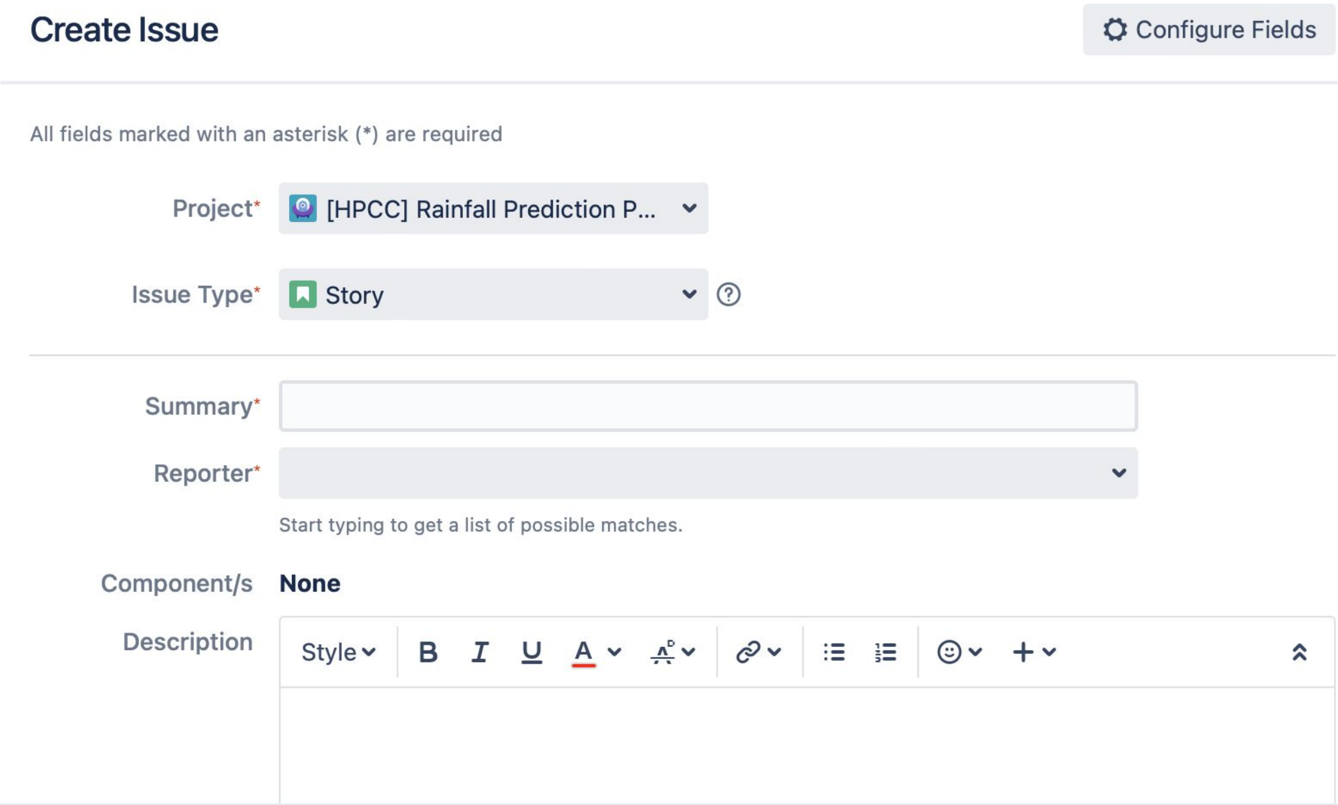


6. Once the developer is working on the defect, the status can be set to “Work in Progress.”
7. After the defect has been fixed, the developer documents the fix in the defect tracking tool and sets the status to “Fixed,” if it was fixed, or “Duplicate,” if the defect is a duplication (specifying the duplicated defect). The status can also be set to “As Designed,” if the function executes correctly. At the same time, the developer reassigns the defect to the originator.
8. Once a new build is received with the implemented fix, the test engineer retests the fix and other possible affected code. If the defect has been corrected with the fix, the test engineer sets the status to “Close.” If the defect has not been corrected with the fix, the test engineer sets the status to “Reopen.”

Defect Life-Cycle Model

Defect Tracking and New Build Process

Jira system



Create Issue

All fields marked with an asterisk (*) are required

Project* [HPCC] Rainfall Prediction P...

Issue Type* Story

Summary*

Reporter*

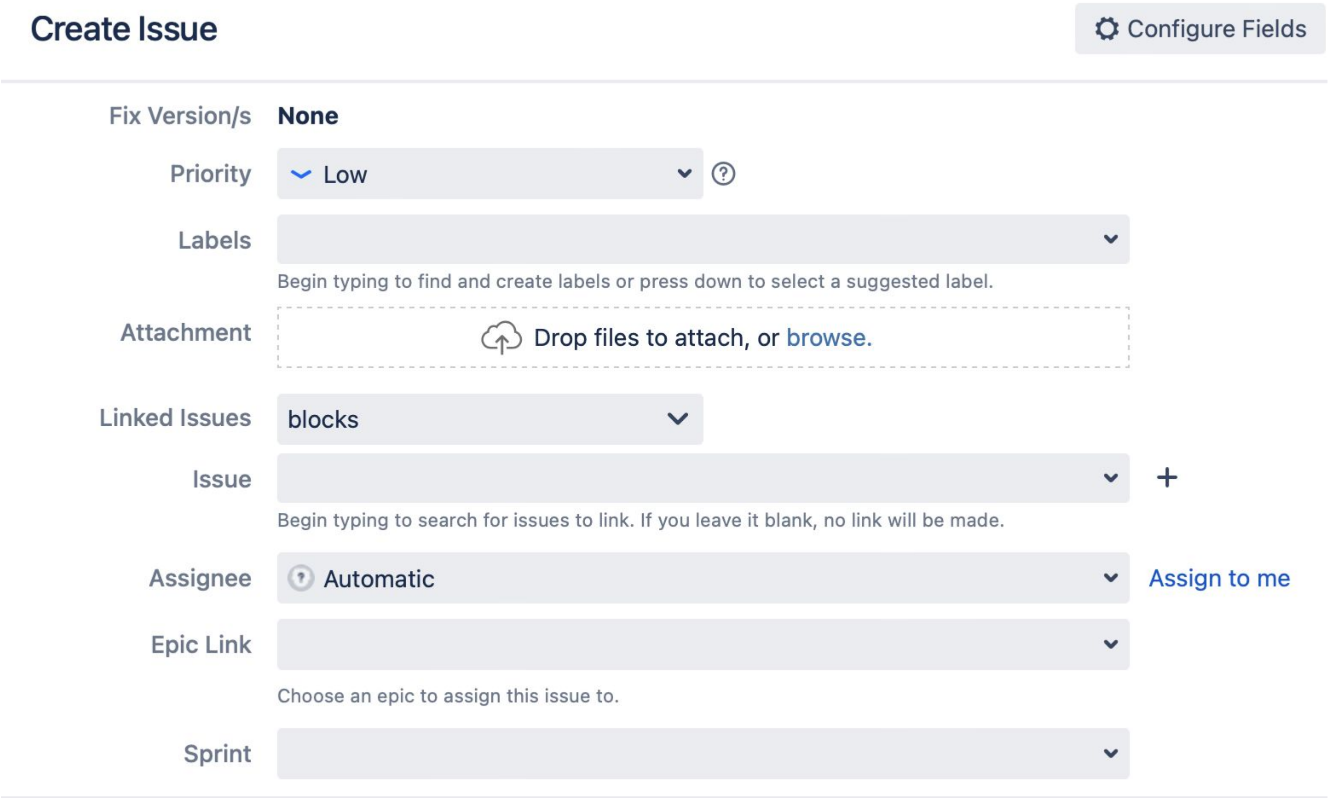
Start typing to get a list of possible matches.

Component/s None

Description

Defect Tracking and New Build Process

Jira system

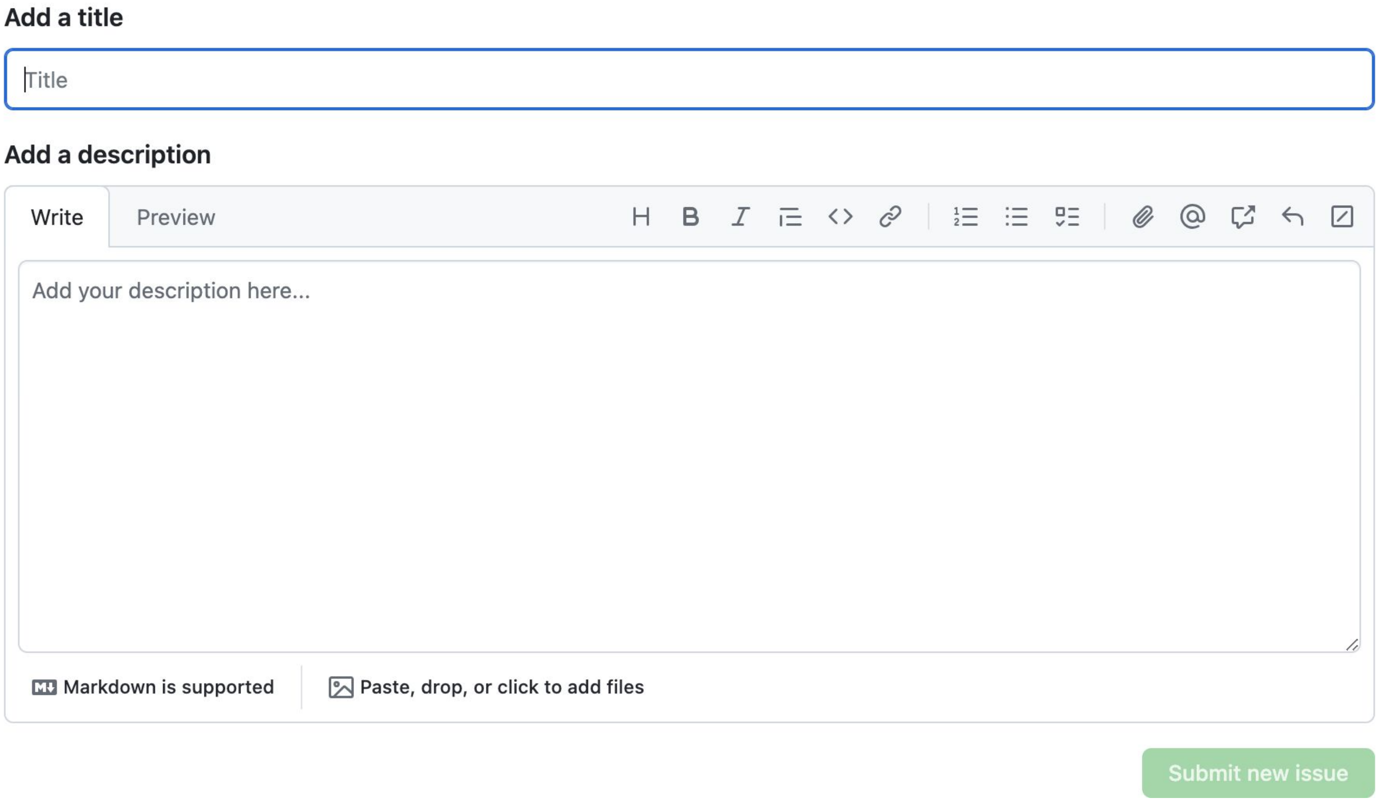


Create Issue

Fix Version/s	**None**
Priority	Low
Labels	Begin typing to find and create labels or press down to select a suggested label.
Attachment	Drop files to attach, or browse.
Linked Issues	blocks
Issue	Begin typing to search for issues to link. If you leave it blank, no link will be made.
Assignee	Automatic
Epic Link	Choose an epic to assign this issue to.
Sprint	

Defect Tracking and New Build Process

Github system



Add a title

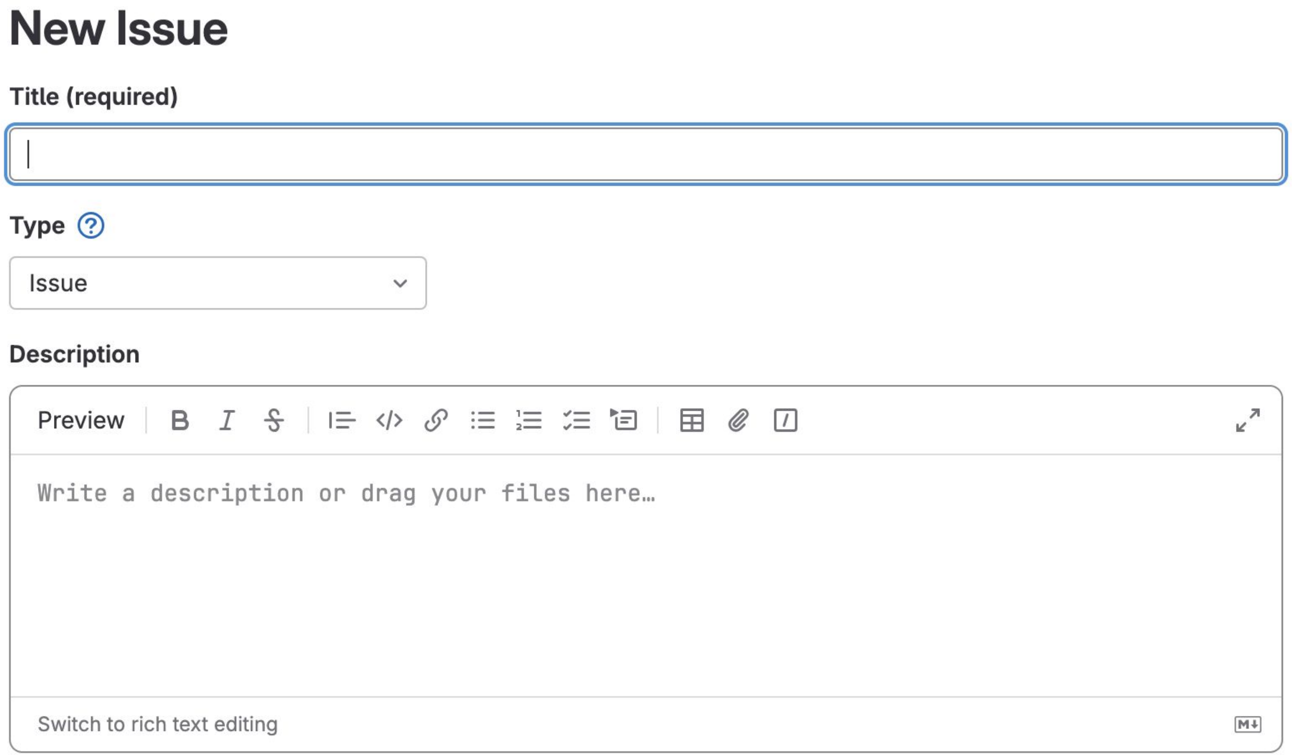
Title

Add a description

Add your description here...

Defect Tracking and New Build Process

Gitlab system



New Issue

Title (required)

Type

Issue

Description

Write a description or drag your files here…

Test Program Status Tracking

Test Program Status Tracking

- The test team manager is responsible for **ensuring** that **tests** are **executed according to schedule.**
- Test **personnel** are **allocated** and **redirected** when necessary to **handle problems** that arise during the test effort.
- To perform this oversight function effectively, the test manager must **conduct test program status tracking and management reporting.**
- The test engineer will need to provide **meaningful reports** based on the **measures** and **metrics** defined within the test plan.
- The test engineer reviews the test coverage report to ascertain whether complete **(100%) test procedure execution coverage** has been achieved.
- Whether test **coverage criteria** have been **met** or whether these criteria should be **modified**.

Test Program Status Tracking

- The test team further needs to **decide** whether **additional test requirements** and test procedures are needed to satisfy test coverage or test completion criteria.
- The test manager needs to implement an earned value approach to test progress status effort => Implementing ***an earned value management system (EVMS)*** is one of the best ways of tracking the test program status.
- The test manager needs to **collect** other measurements of test **performance**, related to test **coverage**, **predictions of time to release AUT**, and **quality of the software at time of release.**
- **Time limitations** often **restrict** the test team's **ability** to **collect**, **track**, and **analyze** such measurements.

Test Program Status Tracking

Earned Value Management System

Earned value analysis involves:

- **Tracking** the value of **completed** work.
- **Comparing** it to **planned costs** and **actual costs** => **provide** a **true measure** of schedule and cost status => **enable** the **creation** of **effective corrective actions.**

Test Program Status Tracking

Earned Value Management System

The earned value process includes four steps:

1. Identify short tasks (functional test phase)
2. Schedule each task (task start date and end date)
3. Assign a budget to each task (task will require 3,100 hours using four test engineers)
4. Measure the progress of each task, enabling the engineer to calculate schedule and cost variance.

=> The use of earned value calculations **requires the collection of performance measurements.**

Test Program Status Tracking

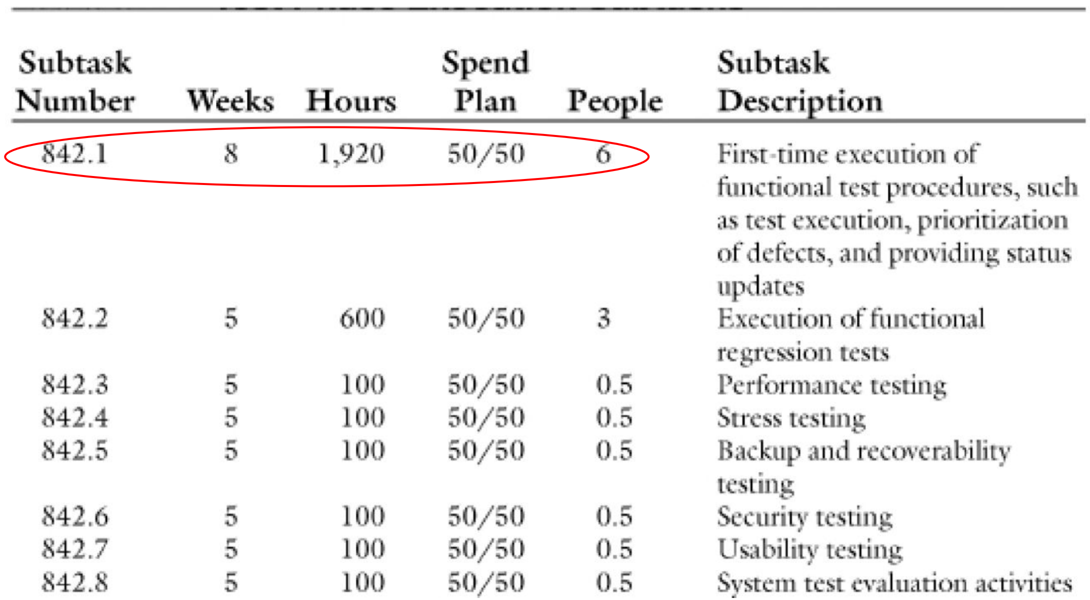
Earned Value Management System

Two key earned value calculations pertain to the assessment of cost and schedule variance:

Earned value for work completed − planned budget = schedule variance

Earned value for work completed − actual cost = cost variance

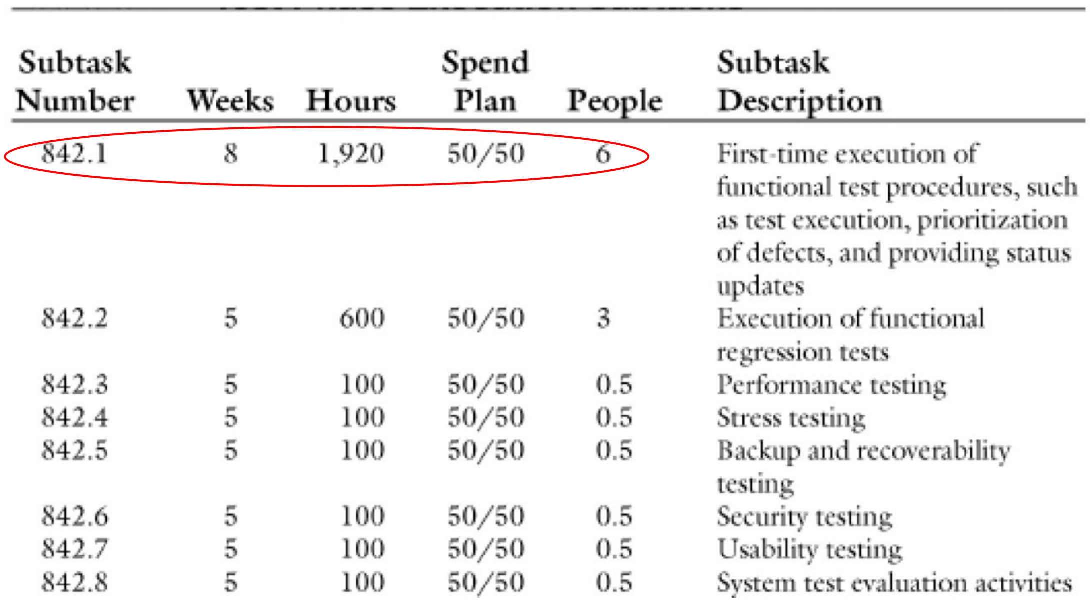
Test Program Status Tracking



Subtask Number	Weeks	Hours	Spend Plan	People	Subtask Description
842.1	8	1,920	50/50	6	First-time execution of functional test procedures, such as test execution, prioritization of defects, and providing status updates
842.2	5	600	50/50	3	Execution of functional regression tests
842.3	5	100	50/50	0.5	Performance testing
842.4	5	100	50/50	0.5	Stress testing
842.5	5	100	50/50	0.5	Backup and recoverability testing
842.6	5	100	50/50	0.5	Security testing
842.7	5	100	50/50	0.5	Usability testing
842.8	5	100	50/50	0.5	System test evaluation activities

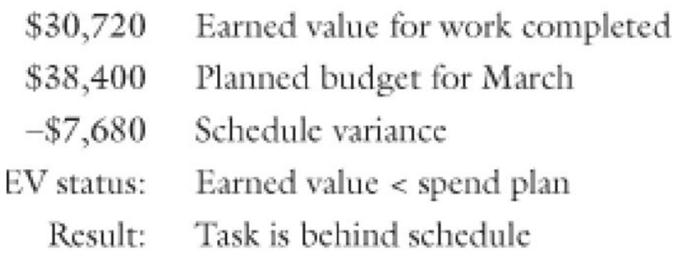
- **600 functional test procedures** must be executed.
- **50%** of the test procedures (300) for **March** <=> **960** planned hours.
- **50%** of the test procedures (300) for **April** <=> **960** planned hours.

Test Program Status Tracking



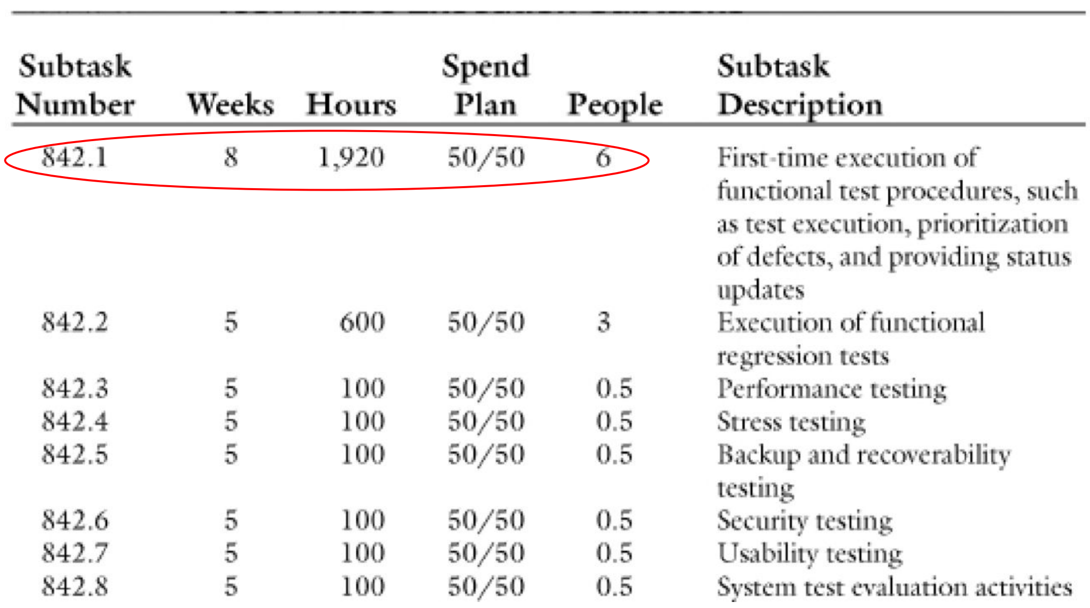
Subtask Number	Weeks	Hours	Spend Plan	People	Subtask Description
842.1	8	1,920	50/50	6	First-time execution of functional test procedures, such as test execution, prioritization of defects, and providing status updates
842.2	5	600	50/50	3	Execution of functional regression tests
842.3	5	100	50/50	0.5	Performance testing
842.4	5	100	50/50	0.5	Stress testing
842.5	5	100	50/50	0.5	Backup and recoverability testing
842.6	5	100	50/50	0.5	Security testing
842.7	5	100	50/50	0.5	Usability testing
842.8	5	100	50/50	0.5	System test evaluation activities

- **80%** (240 procedures) **actually completed**.
- **740 hrs** for March **actually**
- Budgeted to cost **$76,800.**



$30,720	Earned value for work completed
$38,400	Planned budget for March
–$7,680	Schedule variance
EV status:	Earned value < spend plan
Result:	Task is behind schedule

Test Program Status Tracking



Subtask Number	Weeks	Hours	Spend Plan	People	Subtask Description
842.1	8	1,920	50/50	6	First-time execution of functional test procedures, such as test execution, prioritization of defects, and providing status updates
842.2	5	600	50/50	3	Execution of functional regression tests
842.3	5	100	50/50	0.5	Performance testing
842.4	5	100	50/50	0.5	Stress testing
842.5	5	100	50/50	0.5	Backup and recoverability testing
842.6	5	100	50/50	0.5	Security testing
842.7	5	100	50/50	0.5	Usability testing
842.8	5	100	50/50	0.5	System test evaluation activities

How to overcome and meet the completion deadline by end of April?

- Overtime? *(may affect employee morale and contribute to employee turnover if working 9hrs/day).*
- **Other solutions?**

Test Program Status Tracking

Test Metrics Collection and Analysis

- **Test metrics** can provide the test manager with **key indicators** of the test **coverage, progress, and the quality** of the test effort.
- **Be careful** to choose that set of metrics that best **serves its performance** concerns => **time-consuming** if gathering so many metrics and **reduce** the number of **hours** that are **spent actually** performing actual test activities.
- The team should take care to measure whether **the rate of change** in one particular area is much larger than that observed in other areas.

Test Program Status Tracking

Test Metrics Collection and Analysis

Basic elements and prerequisites of a software metric process are structured as:

- **Goals** and **objectives** are set **relative** to the product and software (test) management process.
- **Measurements** are defined and selected to ascertain the degree to which the **goals** and **objectives** are being **met**.
- The **data collection** process and **recording mechanism** are defined and used.

Test Program Status Tracking

Test Metrics Collection and Analysis

Basic elements and prerequisites of a software metric process are structured as:

- **Measurements** and **reports** are part of a **closed-loop** system that provides current (operational) and historical information to technical staff and management.
- **Data on post-software product** life measurement are retained for analysis that could lead to improvements for future product and process management.

Test Program Status Tracking

Test Metrics Collection and Analysis

White-Box Testing Metrics:

- **White-box testing** techniques target the **application's internal workings.**
- **White-box metrics collection has the same focus.**
- The test engineer measures the **depth** of testing by collecting data related to **path coverage and test coverage.**
- The **white-box testing** metric is called coverage analysis.

Test Program Status Tracking

Test Metrics Collection and Analysis

White-Box Testing Metrics:

- Source code analysis and code profiling help discern the **code quality.**
- The objective of **source code complexity analysis** is to identify complex areas of the source code.
- **High-complexity areas** of source code can be sources of high risk.
- **Unnecessary code complexity** can **decrease code reusability** and **increase code maintenance.**
- Testing efforts need to **focus** on **high-complexity code.**

Test Program Status Tracking

Test Metrics Collection and Analysis

White-Box Testing Metrics:

- Another **white-box** test metric of interest is **fault density.**
- **Predict** the remaining faults by comparing the measured fault density with the expected fault density => determine the amount of testing.
- **Fd = Nd/KSLOC**
  - Nd: the number of defects.
  - KSLOC: the number of non-comment lines of source code.

Test Program Status Tracking

Test Metrics Collection and Analysis

Black-Box Testing Metrics:

- Metrics collection focuses on the **breadth of testing.**
- **Black-box** testing techniques are based on the **application's external considerations.**

Test Program Status Tracking

Test Metrics Collection and Analysis

Sample Black-Box Test Metrics

Metric name	Description	Classification
Test coverage	Total number of test procedures/total number of test requirements. The test coverage metric indicates planned test coverage.	Coverage
System coverage analysis	The system coverage analysis measures the amount of coverage at the system interface level.	Coverage
Test procedure execution status	Executed number of test procedures/total number of test procedures. This test procedure execution metric indicates the extent of the testing effort still outstanding.	Progress

Test Program Status Tracking

Test Metrics Collection and Analysis

Sample Black-Box Test Metrics

Metric name	Description	Classification
Error discovery rate	Number of total defects found/number of test procedures executed. The error discovery rate metric uses the same calculation as the defect density metric. It is used to analyze and support a rational product release decision.	Progress
Defect Aging	Date defect was opened versus date defect was fixed. The defect aging metric provides an indication of turnaround of the defect.	Progress
Defect fix retest	Date defect was fixed and released in new build versus date defect was retested. The defect fix retest metric provides an idea about whether the test team is retesting the fixes fast enough so as to get an accurate progress metric.	Progress

Test Program Status Tracking

Test Metrics Collection and Analysis

Sample Black-Box Test Metrics

Metric name	Description	Classification
Defect trend analysis	Number of total defects found versus number of test procedures executed over time. Defect trend analysis can help determine the trend of defects found. Is the trend improving as the testing phase winds down?	Progress
Current quality ratio	Number of test procedures successfully executed (without defects) versus the number of test procedures. The current quality ratio metric provides indications about the amount of functionality that has successfully been demonstrated.	Quality

Test Program Status Tracking

Test Metrics Collection and Analysis

Sample Black-Box Test Metrics

Metric name	Description	Classification
Quality of fixes	Number of total defects reopened/total number of defects fixed. This quality of fixes metric will provide indications of development issues. Ratio of previously working functionality versus new errors introduced. This quality of fixes metric will keep track of how often previously working functionality was adversely affected by software fixes.	Quality

Test Program Status Tracking

Test Metrics Collection and Analysis

Sample Black-Box Test Metrics

Metric name	Description	Classification
Defect density	Number of total defects found/number of test procedures executed per functionality (that is, per use case or test requirement). Defect density can help determine if a specific high amount of defects appear in one part of functionality tested.	Quality
Problem reports	Number of software problem reports broken down by priority. The problem reports measure counts the number of software problems reported, listed by priority.	Quality
Test effectiveness	Test effectiveness needs to be assessed statistically to determine how well the test data have exposed defects contained in the product.	Quality

Test Program Status Tracking

Test Metrics Collection and Analysis / Coverage Metrics

Test Coverage:

- This measurement divides the total number of test procedures developed by the total number of defined test requirements.
- The depth of test coverage is usually based on the defined acceptance criteria.

Test Program Status Tracking

Test Metrics Collection and Analysis / Coverage Metrics

System Coverage Analysis:

- System coverage analysis measures the amount of coverage at the system interface level.

Functional Test Coverage:

- This metric can measure test coverage prior to software delivery.
- Indicates the percentage of the software tested at any point.
- Calculated by dividing the number of test requirements by the total number of test requirements.

Test Program Status Tracking

Test Metrics Collection and Analysis / Coverage Metrics

Progress Metrics:

- The test team can ascertain the number of test procedures remaining to be executed.
- The metric does **not provide an indication of the quality** of the application.
- It **provides information about the depth of the test effort** rather than an indication of its success.
- **Tool: Rational’s Test Manager.**

Test Program Status Tracking

Test Metrics Collection and Analysis / Coverage Metrics

Test Procedure Execution Status:

- During black-box testing, test engineers **collect data** that help identify test progress, so that the test team can **predict the release date** for the AUT.
- Progress metrics are collected **iteratively** during **various stages** of the test life cycle, such as weekly or monthly.

Test Program Status Tracking

Test Metrics Collection and Analysis / Coverage Metrics

Error Discovery Rate:

- This measurement divides the total number of **documented defects** by the number of test **procedures executed**. Test team review of the error discovery rate metric supports trend analysis and helps forecast product release dates.

Defect Fix Retest:

- This metric provides a measure of whether the test team is retesting the corrections at an adequate rate. It is calculated by measuring the time between when the **defect was fixed** in a new build and when the defect **was retested.**

Test Program Status Tracking

Test Metrics Collection and Analysis / Coverage Metrics

Defect Aging:

- Turnaround time - the time from when the **defect** was **identified** to the **resolution** of the defect.
- When evaluating the defect aging measure, the test team also needs to take the priority into consideration.

Defect Fix Retest:

- This metric provides a measure of whether the test team is retesting the corrections at an adequate rate.
- It is calculated by measuring the time between when the defect was fixed in a new build and when the defect was retested.

Test Program Status Tracking

Test Metrics Collection and Analysis / Coverage Metrics

Defect Trend Analysis:

- Defect trend analysis can help to determine **the trend of defects found.**
- This metric compares the total number of defects found with the number of test procedures executed over time.

Quality Metrics:

- Test Success Index (the current quality ratio): computed by taking the total number of test procedures **executed and passed** divided by the total number of test procedures executed. Provides the test team with further insight into the amount of functionality that has been successfully demonstrated.

Test Program Status Tracking

Test Metrics Collection and Analysis / Coverage Metrics

Defect Trend Analysis:

- Calculated by dividing the total number of defects found by the number of test procedures executed.

Test Effectiveness:

- Test effectiveness needs to be assessed statistically to determine how well the test data have exposed defects contained in the product.
- The test team should solicit the assistance of personnel who are experienced in the use of the application, so as to review test results and determine their correctness.

Test Program Status Tracking

Test Metrics Collection and Analysis / Coverage Metrics

Quality of Fixes1 = Number Total Defects Reopened/Number of Total Defects Fixed

Quality of Fixes2 = Previously Working Functionality versus New Errors Introduced

Test Program Status Tracking

Test Metrics Collection and Analysis / Coverage Metrics

Defect Density:

- Calculated by taking the total number of defects found and dividing this value by the number of test procedures executed for a specific functionality or use case.
- Is this **functionality** very **complex** and therefore it would be expected that the defect density is high?
- Is there a **problem** with the **design/implementation** of the functionality?
- Were the **wrong** (or not enough) **resources assigned** to the functionality, because an inaccurate risk had been assigned to it?
- It also could be inferred that the developer responsible for this specific functionality needs more training.

Test Program Status Tracking

Test Metrics Collection and Analysis / Coverage Metrics

Problem Report - Acceptance Criteria Metric:

- Needs to be defined during the test planning phase, **before test execution begins.**
- The test engineer must ascertain whether an AUT satisfies these criteria:
  - Accept all level 1, 2, and 3 (fatal, high, and medium) resolved OR
  - Accept all level 1 and 2 (fatal and high) resolved OR
  - Accept all level 1 and 2 (fatal and high) and 90% of level 3 problem reports have been resolved.

Test Program Status Tracking

Test Metrics Collection and Analysis / Coverage Metrics

Test Automation Metric:

- Need to measure the time spent developing and executing test scripts and compare it with the results that the scripts produced.
- Especially the first time that the project uses an automated testing approach.
- E.g: during stress testing, 1,000 virtual users execute a specific functionality and the system crashes compared to 1000 users?!

Chapter summary

- When executing test procedures, the test team will need to comply with a test procedure execution schedule. Following test execution, test outcome evaluations are performed and test result documentation is prepared.
- Plans for unit, integration, system, and user acceptance testing together make up the steps that are required to test the system as a whole. During the unit testing phase, code profiling can be performed. Traditionally, profiling is a tuning process that determines whether an algorithm is inefficient or a function is called too frequently. It can uncover improper scaling of algorithms, instantiations, and resource utilization.

Chapter summary

- Integration testing focuses on the application's internal workings. During integration testing, units are incrementally integrated and tested together based on control flow. Because units may consist of other units, some integration testing (also called module testing) may take place during unit testing.
- During system testing, the test engineer examines the integration of the parts that make up the entire system. System-level tests are usually performed by a separate test team. The test team implements the test procedure execution schedule and the system test plan.

Chapter summary

- The test team must perform analysis to identify particular components or functionality that are generating a greater relative number of problem reports. As a result of this analysis, additional test procedures and test effort may need to be assigned to the components. Test results analysis can also confirm whether executed test procedures are worthwhile in terms of identifying errors.
- Each test team must perform problem reporting operations in compliance with a defined process. The documentation and tracking of software problem reports are greatly facilitated by an automated defect tracking tool.

Chapter summary

- The test team manager is responsible for ensuring that tests are executed according to schedule and that test personnel are allocated and redirected when necessary to handle problems that arise during the test effort. To perform this oversight function effectively, the test manager needs to perform test program status tracking and management reporting.
- Test metrics provide the test manager with key indicators of the test coverage, progress, and the quality of the test effort. During white-box testing the test engineer measures the depth of testing by collecting data about path coverage and test coverage. During black-box testing, metrics collection focuses on the breadth of testing, including the amount of demonstrated functionality and the amount of testing that has been performed.

Reference

[1] *Software Program Management.* Laguna Hills, CA: Humphreys and Associates, 1998.

[2] Jacobson, I. “Proven Best Practices of Software Development.” Rational ’99 Worldwide Software Symposium, Washington, DC, January 26, 1999.

[3] Florac, W.A., et al. *Software Quality Measurement: A Framework for Counting Problems and Defects.* Technical Report, CMU/SEI-92-TR-22, ESC-TR-92-022. Software Engineering Institute, Pittsburgh, PA, September 1992.

[4] McCabe, T.J. Structured Testing: *A Software Testing Methodology Using the Cyclomatic Complexity Metric*. NBS Special Publication 500-99. Washington, DC: U.S. Department of Commerce/National Institute of Standards and Technology, 1982.

[5] ANSI/IEEE Standard 982.2-1988.