STANNS COLLEGE OF ENGINEERING AND TECHNOLOGY
SOFTWARE TESTING METHODOLOGIES

Unit-5

Efficient Test Suite Management: Test case deisgn, Why does a test suite grow, Minimizing the test suite and its benefits, test suite prioritization, Types of test case prioritization, prioritization techniques, measuring the effectiveness of a prioritized test suite

Software Quality Management: Software Quality metrics, SQA models

Debugging: process, techniques, correcting bugs, Basics of testing management tools, test link and Jira

Test Case Deisgn:

--Basically test design is the act of creating and writing test suites for testing a software.
--Test analysis and identifying test conditions gives us a generic idea for testing which covers quite a large range of possibilities.
--But when we come to make a test case we need to be very specific. In fact now we need the exact and detailed specific input.
--But just having some values to input to the system is not a test, if you don't know what the system is supposed to do with the inputs, you will not be able to tell that whether your test has passed or failed.
--Test cases can be documented as described in theIEEE 829 Standard for Test Documentation.
--The IEEE 829 Standard for Test Documentation consists of different documents covering: *Test plan, Test Design Specification, Test Case Specification, Test Procedure Specification, Test Item Transmittal Report, Test Log, Test Incidental Report, Test Summary Report.*

Why does a test suite grow?

--There may be unnecassary test cases in the test suite including both obsolete and redundant test cases.
--For example a change in a program causes a test case to become obsolete.
--A test case is redundant if other test cases in test suite provide the same coverage of the program.
--Thus due to obsolete and redundant test cases, the size of a test suite continues to grow unnecassarily.
--Test engineers measures the extent to which a criterion is satisfied in terms of *Coverage*.
--Coverage is measured in terms of the requirements that are imposed.
--Partial Coverage is defined as the percentage of requirements that are satisfied.
-- Coverage Criteria is used as a stopping point to decide when a program is sufficiently tested.

Minimizing the test suite and its benefits:

A test suite can sometimes grow to an extent that it is nearly impossible to execute. In this case, it becomes necessary to minimize the test cases such that they are executed for maximum coverage. The reasons why minimization is important are:
-->Release date of the product is near.
-->Limited staff to execute all the test cases.
-->Limited test equipments or unavailability of testing tools

Minimizing the test suite has the following Benefits:
-- Redundant test cases will be eliminated.
-- Lower costs by reducing a test suite to a minimal subset.

--A reduction in the size of test suite decreases both the overhead of maintaining the test suite and the number of test cases that must be rerun after changes are made to the software, thereby reducing the cost of regression testing.

Test Suite Prioritization:

The purpose of prioritization is to reduce the number of test cases based on some criteria, while aiming to select the most appropriate tests. The different priorities are:

Priority1: *The test cases that must be executed* otherwise there may be worse consequences for the release of the product.

Priority 2: *The test cases may be executed, if time permits.*

Priority 3: *The test case is not important prior to the current release*. It may be tested shortly after the release of the current version of the software.

Priority 4: *The test case is never important as its impact is nearly negligible.*

Types of test case prioritization:

General Test Case Prioritization:

In this prioritization, we prioritize the test cases that will be useful over a succession of subsequent modified versions of P (Program or Product), without any knowledge of the modified versions.

Version-Specific Test Case Prioritization:

Here, we prioritize the test cases such that they will be useful on a specific version P' of P.

Prioritization Techniques:

Coverage Based Test Prioritization

Risk Based Prioritization

Prioritization using Relevant Slices

Prioritization based on Requirements

Coverage Based Test Prioritization:

Total Statement based Prioritization: This prioritization orders the test cases based on the total number of statements covered. It counts the number of statements covered by test cases and orders them in descending order. For example, if T1 covers 5 statements, T2 covers 7 statements and T3 covers 14 statements, then the order of prioritization is T3, T3, T1.

Additional Statement Coverage Prioritization:

Statement	Statement Coverage		
	Test case 1	Test case 2	Test case 3
1	X	X	X
2	X	X	X
3		X	X
4			X
5			
6		X	
7	X	X	
8	X	X	
9	X	X	

This technique iteratively selects a test case T1, that yields the greatest statement coverage, then selects a test case which covers a statement uncovered by T1.

Total Branch coverage Prioritization: In this prioritization, the criteria to order is to consider condition branches in a program instead of statements. Thus it is the coverage of each possible outcome of a condition in a predicate. The test case which will cover maximum branch outcomes will be ordered first. For example, in the following digram the order will be test cases 1,2,3.

Branch Statements	Branch Coverage		
	Test case 1	Test case 2	Test case 3
Entry to while	X	X	X
2-true	X	X	X
2-false	X		
3-true		X	
3-false	X		

Total Fault-Exposing-Potential (FEP) Prioritization: Statement and branch coverage prioritization ignore a fact about test cases and faults. Thus the ability of a test case to uncover a fault is called the ***fault exposing potential***. To obtain an approximation of a test case, an approach was adopted using manual analysis. The approach is:

-->Given program P and test suite T, First create a set of mutants N = {n1, n2,…..nm} for P, noting which statement s_j in P contains each mutant.

-->Next, for each test case t_i ćT, execute each mutant version n_k of P on t_i , noting whether t_i kills that mutant.

-->Calculate the FEP(s,t) as the ratio : Mutants of s_j killed / Total number of mutants of s_j.

To perform total FEP prioritization, given these FEP(s,t) values, calculate an *award value* for each test case ti ćT, by summing the FEP(sj, ti) values for all statements sj in P. Given these award values, we prioritize test cases by sorting them in order of descending award value.

Statement	FEP(s,t) values		
	Test case 1	Test case 2	Test case 3
1	0.5	0.5	0.3
2	0.4	0.5	0.4
3		0.01	0.4
4		1.3	
5			
6	0.3		
7	0.6		0.1
8		0.8	0.2
9			0.6
Award values	1.8	3.11	2.0

Risk Based Prioritization:

Risk Based technique is to prioritize the test cases based on some potential problems which may occur during the project. The two components of risk based technique are:

Probability of occurence / fault likelihood: It indicates the probability of occurence of a problem.

Severity of impact / failure impact: If the problem has occurred, how much impact does it have on the software.

A risk analysis table consists of the following columns: Problem ID, Potential Problem, Uncertainity factor, Severity of impact, Risk Exposure. For example, the problems in the given table can be prioritized in the order of P5, P4, P2, P3 and P1 based on the risk exposure.

Problem ID	Potential Problem	Uncertainty Factor	Risk Impact	Risk Exposure
P_1	Specification ambiguity	2	3	6
P_2	Interface problems	5	6	30
P_3	File corruption	6	4	24
P_4	Databases not synchronized	8	7	56
P_5	Unavailability of modules for integration	9	10	90

Prioritization using Relevant Slices:

Execution Slice: The set of statements executed under a test case is called *Execution Slice* of the program.

Example:

```
S1:    read(a,b);
S2:    result;
S2:    if(a>=100 || b>=200)
S3:        result=a+b;
S4:    else
S5:        result=a-b;
S6:    return result;
```

Test Cases	a	b	result
1	150	250	400
2	50	200	250
3	200	50	250
4	20	20	0

Dynamic Slice: The set of statements executed under a test case and have an effect on the program output under that test case is called *Dynamic Slice* of the program with respect to the output variables.

Example:

```
S1:    read(a,b);
```

```
S2:     sum=0, I;
S2:     if(a==0)
S3:             print a;
S4:     else
S5:             print b;
S6:     else if
S7:             {
S8:                     sum=a+b;
S9:                     I=50;
S10:                    Print(I);
S11:            }
S12:    return(sum);
```

Relevant Slice: The set of statements that were executed under a test case and did not affect the output, but have potential to affect the output produced by a test case is known as *Relevant Slice* of the program. For example, in the above code, statements S2 , S4 have the potential to affect the output, if modified.

Prioritization based on Requirements:

Hema Srikanth et al. [136] have applied requirement engineering approach for prioritizing the system test cases. It is known as PORT (Prioritization of Requirements for Test). They have considered the four factors for analyzing and measuring the criticality of requirements:

--*Customer-Assigned priority of requirements*
--*Requirement Volatility*
--*Developer*-perceived implementation complexity
--*Fault proneness of requirements*

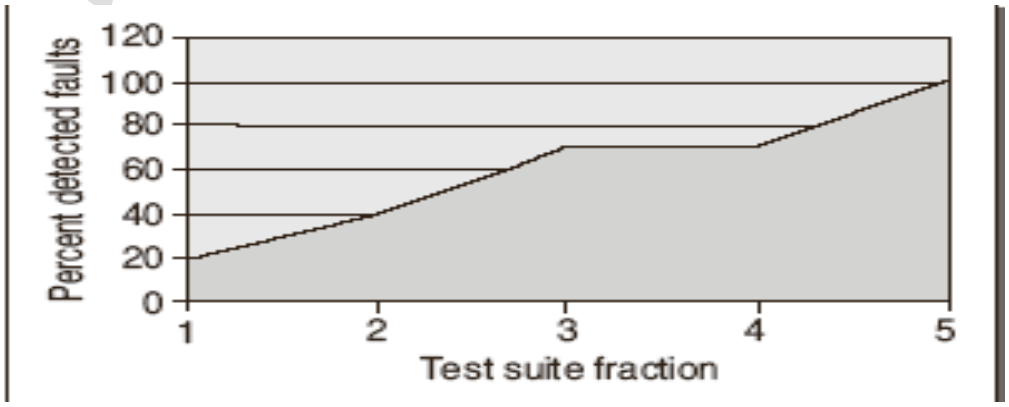
Measuring the effectiveness of a Prioritized Test Suite:

Elbaum et al. [133,134] developed APFD (Average percentage of faults detected) metric that measures the weighted average of the percentage of faults detected during the execution of the test suite. Its value ranges from 0 to 100 where higher value means faster fault detection rate.

APFD is calculated as given below.

APFD = 1- ((TF1+TF2+…….+ TFm)/nm) + 1/2n

where: --TFi is the position of the first test in Test suite T that exposes fault i
-- m is the total number of faults exposed in the system or module under T
--n is the total number of test cases in T



For example:

	T1	T2	T3	T4	T5
F1			X	X	
F2	X		X		X
F3		X		X	
F4	X		X	X	
F5		X		X	X

APFD= 1-((3+1+2+1+2)/ 5*5) + (1/2)*5

Software Quality Management

Software Quality metrics, SQA models

Software Quality Metrics:

Software Quality Metrics a subset of software metrics that focus on the quality aspects of the product, process and project. Software Quality Metrics can be divided into 3 groups:
-->Product Quality Metrics
-->In-Process Quality Metrics
--> Metrics for Software Maintenance

Product Quality Metrics:

Mean time to failure (MTTF): MTTF metric is an estimate of the average or mean time until a product's first failure occurs. It is calculated by using:

MTTF=total operating time of the units tested / total number of failures encountered.

Defect Density Metrics: It measures the defects relative to the software size.

Defect Density = Number of Defects / Size of Product

Customer problem metrics: This metric measures the problems which customers face while using the product.the problems may be valid defects or usability problems, unclear documentation etc. The problems metric is usuallly expressed in terms of Problems per User Month (PUM).

PUM = Total problems reported by the customer for a time period / Total number of licensed months of the software during the period.

Customer Satisfaction metrics: Customer satisfaction is usually measured through various methods of customer surveys via the five point scale: *Very Satisfied, Satisfied, Neutral, Dissatisfied, Very Satisfied.*

In-Process Quality Metrics:

Defect Density during testing: Higher defect rates found during testing is an indicator that the software has experienced higher error injection during its development process.

Defect Arrival pattern during Testing: The pattern of defect arrivals or the time between consecutive failures gives more information.

Defect Removal Efficiency:

DRE = (Defects removed during a development phase / Defects latent in the product) X 100%

Metrics for Software Maintenance:

Fix backlog and backlog management index: Fix backlog metric is the count of reported problems that remain open at the end of a month or a week. BMI (Backlog Mananement Metric) is used to express the index:

BMI = (Number of problems closed during the month / Number of problem arrivals during the month) x 100 %

Fix response time and fix responsiveness: While fixing the probelms, time-limit also matters according to the severity of the problems.

Percent Delinquent fixes: For each fix, if the turn-around time greatly exceeds the required response time, then it is classified as delinquent.

Percent Delinquent fixes = (Number of fixes that exceeded the response time criteria by severity level / Number of fixes delivered in a specified time) X 100%

Fix Quality: It is the metric to measure the number of fixes that turn out to be defective.

Software Quality Assurance (SQA) Models:

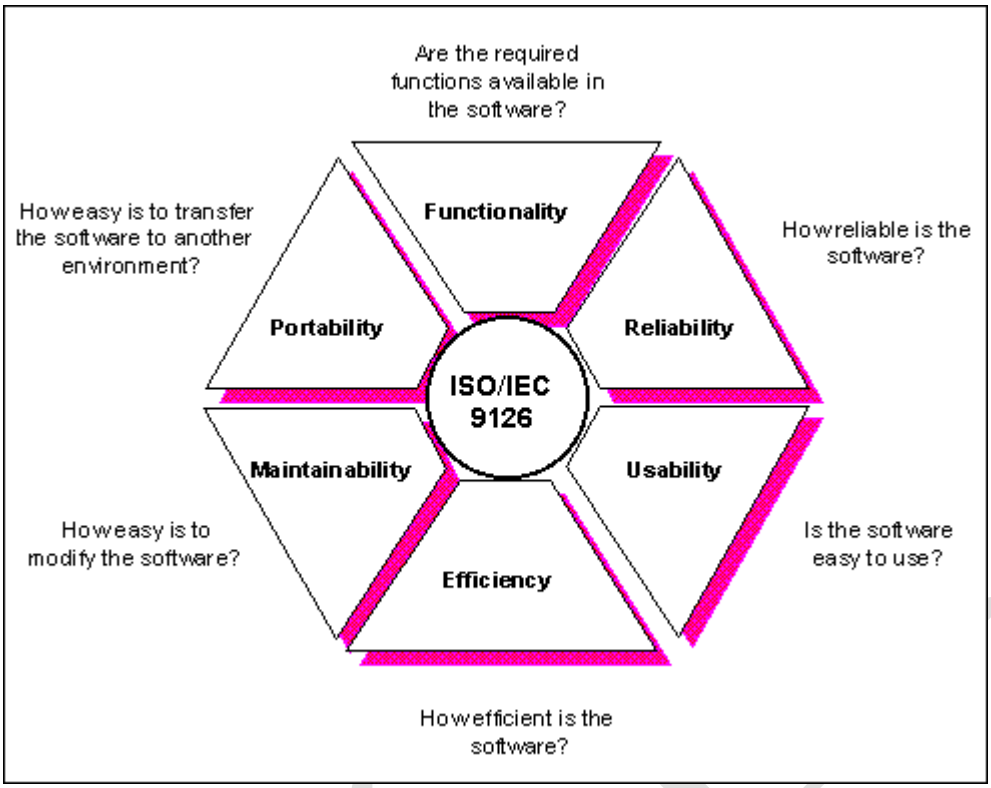
ISO 9126:

--Software systems are large and complex with lots of maintenance problems, while users expect high quality. However, it is hard to assess and assure quality.
--The ISO/IEC 9126 standard has been developed to address software quality issues.
-- It specifies software product quality characteristics and sub-characteristics and proposes metrics for their evaluation.
--It is generic, and can be applied to any type of software product by being tailored to a specific purpose.
-- It aims at eliminating any misunderstanding between customers and developers.
-- ISO 9126 provides a set of six quality characteristics:

- Functionality
- Efficiency
- Maintainability
- Reliability
- Usability
- Portability.

--The proposed new edition of ISO/IEC 9126 will be divided into three parts:
ISO/IEC 9126-1: Information technology - Software quality characteristics and metrics - Part 1: Quality characteristics and subcharacteristics.
ISO/IEC 9126-2: Information technology - Software quality characteristics and metrics - Part 2: External metrics

ISO/IEC 9126-3: Information technology - Software quality characteristics and metrics - Part 3: Internal metrics.



ISO 9126 defines the following guidelines for implementation of this model:
Quality requirements definition: First comes the quality requirements definition, which takes as input a set of stated or implied needs, relevant technical documentation and the ISO Standard itself and produces a quality requirement specification.

Evaluation preparation: The second stage is that of evaluation preparation, which involves the selection of appropriate metrics, a rating level definition and the definition of assessment criteria. Metrics, in ISO 9126, typically give rise to quantifiable measures mapped on to scales. The rating levels definition determines what ranges of values on those scales count as satisfactory or unsatisfactory. The assessment criteria definition involves preparing a procedure for summarising the results of the evaluation.

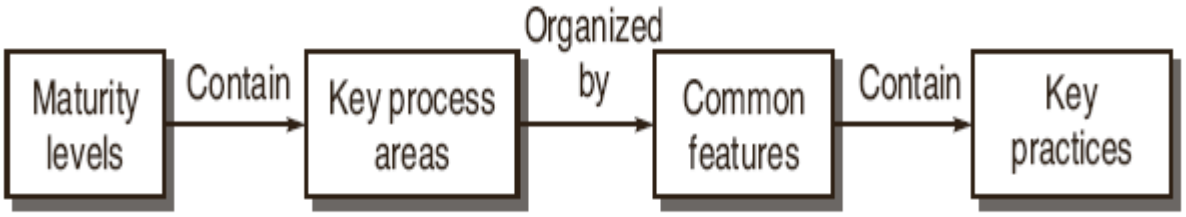
Evaluation procedure: The final stage is the evaluation procedure, which is refined into three steps: measurement, rating and assessment. In measurement, the selected metrics are applied to the software product and values on the scales of the metrics obtained. Subsequently, for each measured value, the rating level is determined. Assessment is the final step of the software evaluation process, where a set of rated levels are summarised. The result is a summary of the quality of the software product.

Capability Maturity Model (CMM):

Capability Maturity Model is a bench-mark for measuring the maturity of an organization's software process. It is a methodology used to develop and refine an organization's software development process. CMM can be used to assess an organization against a scale of five process maturity levels based on certain Key Process Areas (KPA). It describes the maturity of the company based upon the project the company is dealing with and the clients. Each level ranks the organization according to its standardization of processes in the subject area being assessed.

A maturity model provides:
--A place to start
--The benefit of a community's prior experiences
--A common language and a shared vision
--A framework for prioritizing actions
--A way to define what improvement means for your organization

CMM Structure:



Maturity levels	Indicate	Process capability
Key process areas	Achieve	Goals
Common features	Address	Implementation/Institutionalization
Key practices	Describe	Infrastructure/Activities

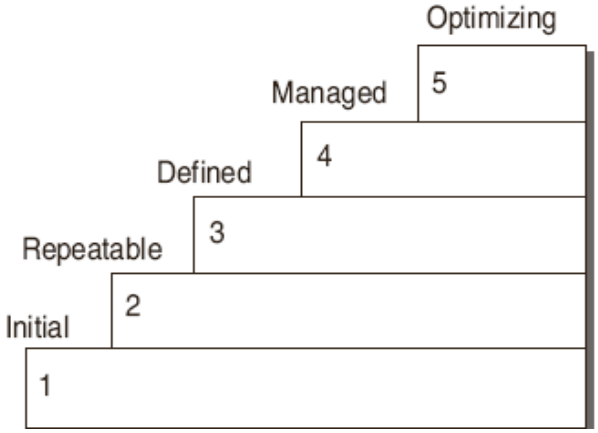
*Maturity Levels:*It is a layered framework providing a progression to the discipline needed to engage in continuous improvement

Key Process Areas: A Key Process Area (KPA) identifies a cluster of related activities that, when performed collectively, achieve a set of goals considered important.

Common Features: Common features include practices that implement and institutionalize a key process area. These five types of common features include: Commitment to Perform, Ability to Perform, Activities Performed, Measurement and Analysis, and Verifying Implementation.

Key Practices: The key practices describe the elements of infrastructure and practice that contribute most effectively to the implementation and institutionalization of the key process areas.

Maturity Levels:



***Initial*:** It is a basis for comparison with the next levels. In an organization at the initial level, conditions are not stable for the development of quality software.

***Repeatable*:** At this level, project management technologies have been introduced in a company.

That project planning and management is based on accumulated experience and there are standards for produced software (these standards are documented) and there is a special quality management group.

***Defined*:** Here, standards for the processes of software development and maintenance are introduced and documented (including project management). During the introduction of standards, a transition to more effective technologies occurs.

***Managed*:** There are quantitative indices (for both software and process as a whole) established in the organization. Better project management is achieved due to the decrease of digression in different project indices.

***Optimizing*:** Improvement procedures are carried out not only for existing processes, but also for evaluation of the efficiency of newly introduced innovative technologies. The main goal of an organization on this level is permanent improvement of existing processes.

Maturity Level	Process Capability
1	-----
2	Disciplined Process
3	Standard Consistent Process
4	Predictable Process
5	Continously Improving Proces

Key Process Areas:

Table 13.2 KPAs for level 2

KPAs, at this level, focus on the project's concerns related to establishing basic project management controls.	
KPA	***Goals***
Requirement Management	Document the requirements properly. Manage the requirement changes properly.
Software Project Planning	Ensure proper project plan including estimation and listing of activities to be done.
Software Project Tracking and Oversight	Evaluate the actual performance of the project against the plans during project execution. Action, if there is deviation from plan.
Software Sub-contract Management	Selects qualified software sub-contractors. Maintain ongoing communications between prime contractor and the software sub-contractor. Track the software sub-contractor's actual results and performance against its commitments.
Software Quality Assurance	Plan the SQA activities. Ensure that there are proper processes by conducting review and audits. Take proper actions, if projects fail.
Software Configuration Management	Identify work products and documents to be controlled in the project. Control the changes in the software.

Table 13.3 KPAs for level 3

KPAs, at this level, address both project and organizational issues, as the organization establishes an infrastructure that institutionalizes effective software engineering and management processes across all projects.	
KPA	***Goals***
Organization Process Focus	Coordinate process development and improvement activities across the organization. Compare software processes with a process standard. Plan organization-level process development and improvement activities.
Organization Process Definition	Standard software processes are defined and documented.
Training Program	Identify training needs of various team members and implementation of training programs.
Integrated Software Management	Tailor the project from the standard defined process. Manage the project according to defined process.
Software Product Engineering	Define, integrate, and perform software engineering tasks. Keep the work products consistent especially if there are changes.
Inter-group Coordination	Ensure coordination between different groups.
Peer Reviews	Plan peer review activities. Identify and remove defects in the software work products.

Table 13.4 KPAs for level 4

KPAs, at this level, focus on establishing a quantitative understanding of both the software process and the software work products being built.	
KPA	***Goals***
Quantitative Process Management	Plan the quantitative process management activities. Control the performance of the project's defined software process quantitatively. The process capability of the organization's standard software process is known in quantitative terms.
Software Quality Management	Set and plan quantitative quality goals for the project. Measure the actual performance of the project with quality goals and compare them with plans.

Table 13.5 KPAs for level 5

KPAs, at this level, cover the issues that both the organization and the projects must address to implement continuous and measurable software process improvement.	
KPA	***Goals***
Defect Prevention	Plan the defect prevention activities. Identify, prioritize, and eliminate the common causes of bugs.
Technology Change Management	Plan the technology changes to be incorporated in the project, if any. Evaluate the effect of new technologies on quality and productivity.
Process Change Management	Plan the process improvement activities such that an organization-wide participation is there. Measure the change of improvement in the process.

Software Total Quality Management (TQM):

Total quality management is a widely used philosophy and business approach that requires all departments in an organization to participate in continuous quality improvement efforts. A TQM workplace values high performance and avoids, or at least minimizes, waste. Most companies, especially manufacturers in regulated environments, use TQM software (or TQM system) to help them instill total quality management procedures in all aspects of their operations. TQM software that exists in the market today is designed based on total quality management principles that can be found in quality standards and regulations.

TQM is defined as a quality-centered, customer-focused, fact-based, team-driven, senior-management-led process to achieve an organization's strategic imperative through continuous process improvement.

T = Total = ***everyone*** in the organization
Q = Quality = ***customer satisfaction***
M = Management = ***people*** *and* ***processes***

The elements of the TQM system can be summarized as follows:

Customer-focused*:* The customer ultimately determines the level of quality. No matter what an organization does to foster quality improvement—training employees, integrating quality into the design process, upgrading computers or software, or buying new measuring tools—the customer determines whether the efforts were worthwhile.

Process-centered**:** A fundamental part of TQM is a focus on process thinking. A process is a series of steps that take inputs from suppliers (internal or external) and transforms them into outputs that are delivered to customers (again, either internal or external). The steps required to carry out the process are defined, and performance measures are continuously monitored in order to detect unexpected variation.

Human side of Quality: Every organization has a unique work culture, and it is virtually impossible to achieve excellence in its products and services unless a good quality culture has been fostered. Thus, an integrated system connects business improvement elements in an attempt to continually improve and exceed the expectations of customers, employees, and other stakeholders.

Measuremetnt and Analysis*:* A major thrust of TQM is continual process improvement. Continual improvement drives an organization to be both analytical and creative in finding ways to become more competitive and more effective at meeting stakeholder expectations.

Six Sigma:

Six Sigma is a methodology for pursuing continuous improvement in customer satisfaction and profit. It is a management philosophy attempting to improve effectiveness and efficiency. Six Sigma is a highly disciplined process that helps us focus on developing and delivering near-perfect products and services.

Features of Six Sigma:
--Six Sigma's aim is to eliminate waste and inefficiency, thereby increasing customer satisfaction by delivering what the customer is expecting.
--Six Sigma follows a structured methodology, and has defined roles for the participants.
--Six Sigma is a data driven methodology, and requires accurate data collection for the processes being analyzed.
--Six Sigma is about putting results on Financial Statements.
--Six Sigma is a business-driven, multi-dimensional structured approach for:

- Improving Processes
- Lowering Defects
- Reducing process variability
- Reducing costs
- Increasing customer satisfaction
- Increased profits

The word *Sigma* is a statistical term that measures how far a given process deviates from perfection. The central idea behind Six Sigma: If you can measure how many "defects" you have in a process, you can systematically figure out how to eliminate them and get as close to "zero defects" as possible and specifically it means a failure rate of 3.4 parts per million or 99.9997% perfect.

Benefits of Six Sigma
--Generates sustained success
--Sets a performance goal for everyone
--Enhances value to customers
--Accelerates the rate of improvement
--Promotes learning and cross-pollination
--Executes strategic change

Six sigma processes will produce less than 3.4 defects per million opportunities. To achieve this target, it uses a methodology known as DMAIC having the following steps:

--**D**efine opportunities,
--**M**easure performance,
--**A**nalyze opportunity,
--**I**mprove performance,

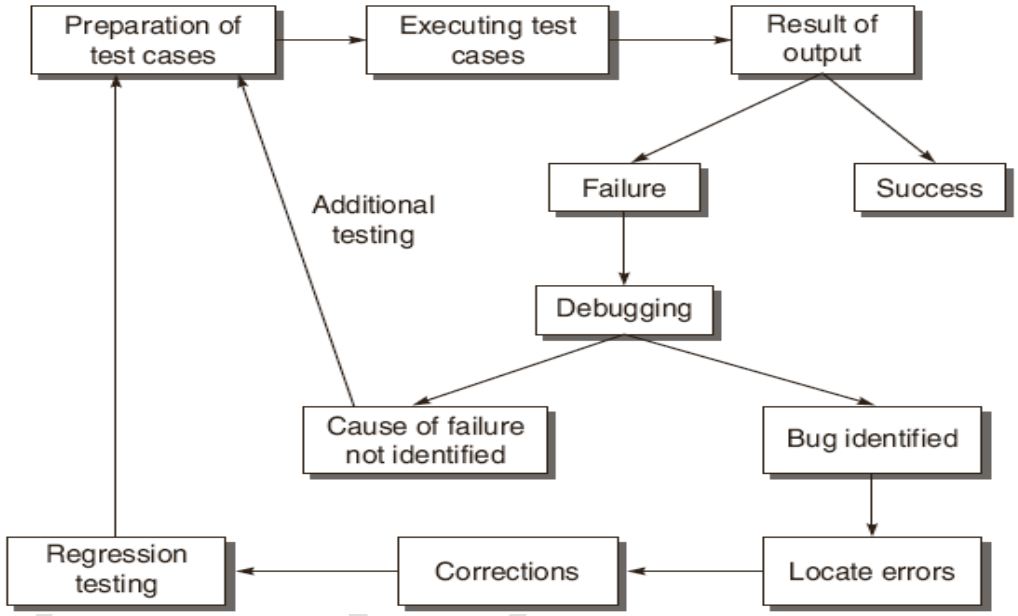
--Control performance

Debugging: Process, Techniques, Correcting bugs, Basics of testing management tools, test link and Jira

Process:

The goal of debugging process is to determine the exact nature of failure with the help of symtoms identified, locate the bugs and errors, and finally corect it. The debugging process is explained in the following steps:

--> Check the result of the output produced by executing test cases prepared in the testing process. If the actual ouput matches with the expected output, it means that the results are successful.
-->Debugging is performed for the analysis of failure that has occurred, where we identify the cause of the problem and correct it. It may be possible that symtoms associated with the present failure are not sufficient to find the bug.



Debugging Techniques:

Debugging with Memory Dump:

In this technique a printout of all registers and relevant memory locations is obtained and studied. The relevant data of the program is observed through these memory locations and registers for any bug in the program. Following are the drawbacks of this method:

--There is difficulty of establishing the correspondence between storage locations and the variable in one's source program.

--It shows the state of the program at only one instant of time.

-- The Storage locations are often represented in octal or hexadecimal formats.

Debugging with Watch Points:

At a particular point of execution in the program, value of variables or other actions can be verified. These particular points of execution are known as *watch points*. Debugging with *watch points* can be implemented with the following methods:

Output Statements: In this method, output statements can be used to check the state of a condition

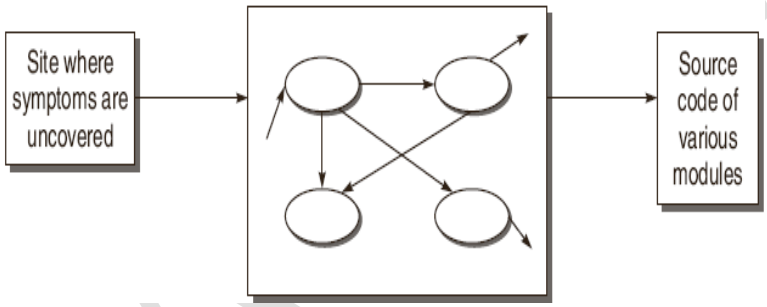
or a variable at some watch point in the program. Therefore, output statements are inserted at various watch points; program is compiled and executed with these output statements.

Breakpoint Execution: Breakpoint is actually a watch point inserted at various places in the program. Breakpoints allow the programmer to control execution of the program and specify how and where the application will stop to allow further examination. Breakpoints have an obvious advantage over output statements:
--Removing the breakpoints are easy when compared to output statements.
-- There is no need to compile the program after inserting breakpoint, while it is necessary for output statement.

Breakpoints can be categorized as:
*Unconditional Breakpoint***:** It is a simple breakpoint whitout any condition to be evaluated.
*Conditional Breakpoint***:** On the activation of this breakpoint, one expression is evaluated for its boolean value.
Temporary Breakpoint: This breakpoint is used only once in the program. When it is set, the program starts running, and once it stops, it is removed.

Single Stepping: The idea of single stepping is that the users ahould be able to watch the execution of the program after every executable statement.

Back Tracking:



An effective method for locating errors in small programs is to backtrack the incorrect results through the logic of the program until you find the point where the logic went astray. In other words, start at the point where the program gives the incorrect result—such as where incorrect data were printed. At this point you deduce from the observed output what the values of the program's variables must have been. By performing a mental reverse execution of the program from this point and repeatedly using the process of "if this was the state of the program at this point, then this must have been the state of the program up here," you can quickly pinpoint the error.

Correcting Bugs:

Before correcting the errors, we should concentrate on the following points:

(a) Highly coupled module correction can introduce many other bugs. That is why low-coupled module is easy to debug.

(b) After recognizing the influence of corrections on the other modules, plan the regression tests.

(c) Perform regression testing with every correction in the software to ensure that the corrections have not introduced new bugs.

Debugging Guidelines:
--Fresh thinking leads to good debugging
--Don't isolate the bug from your colleagues
-- Don't attempt the code modifications in first attempt

--Additional test cases are must if you don't get the symptom or clues to solve the problem
--Regression testing is a must after debugging
--Design should be referred before fixing the error

Basics of Test Management Tools:

Test management tools are used to store information on how testing is to be done, plan testing activities and report the status of quality assurance activities. The tools have different approaches to testing and thus have different sets of features. Generally they are used to maintain and plan manual testing, run or gather execution data from automated tests, manage multiple environments and to enter information about found defects. Test management tools offer the prospect of streamlining the testing process and allow quick access to data analysis, collaborative tools and easy communication across multiple project teams. Many test management tools incorporate Requirements management capabilities to streamline test case design from the requirements. Tracking of defects and project tasks are done within one application to further simplify the testing.

Test management tools give teams the ability to consolidate and structure the test process using one test management tool, instead of installing multiple applications that are designed to manage only one step of the process. Test management tools allow teams to manage test case environments, automated tests, defects and project tasks. Some applications include advanced dashboards and detailed tracking of key metrics, allowing for easy tracking of progress and bug management.

Once a project has kicked off, a test management tool tracks bug status, defects and projects tasks, and allows for collaboration across the team. When administering test cases, users can access a variety of dashboards to gain access to data instantly, making the test process efficient and accurate. The type of dashboard used is determined by the scope of the project and the information and data that needs to be extracted during the testing process. Data can be shared and accessed across multiple project teams, allowing for effective communication and collaboration throughout the testing process.

Test link and Jira:

JIRA is a tool developed by Australian Company Atlassian. It is used for **bug tracking, issue tracking,** and **project management**. The name "JIRA" is actually inherited from the Japanese word "Gojira" which means "Godzilla". The basic use of this tool is to track issues, and bugs related to your software and mobile apps. It is also used for project management. The JIRA dashboard consists of many useful functions and features which make handling of issues easy.

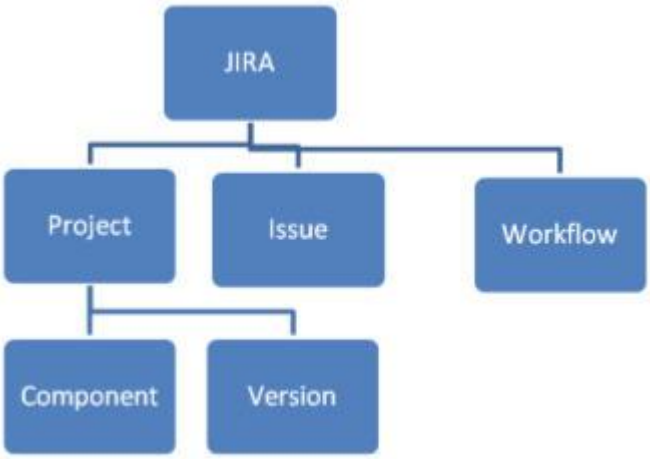
Who uses JIRA?

Software project development teams, help desk systems, leave request systems etc. Coming to its applicability to QA teams, it is widely used for bug tracking, tracking project level issues- like documentation completion and for tracking environmental issues. A working knowledge of this tool is highly desirable across the industry.

Basics about JIRA:
JIRA in its entirety is based on 3 concepts:

Issue: Every task, bug, enhancement request; basically anything to be created and tracked via JIRA is considered an Issue.
Project: a collection of issues
Workflow: A workflow is simply the series of steps an issue goes through starting from creation to

completion.



Say the issue first gets created, goes to being worked on and when complete gets closed. The work flow in this case is:

Every project has the following main attributes:
1.Name – as selected by the administrator.
2.Key- It is an identifier that all the issue names under the project are going to start with. This value is set during the creation of a project and cannot be modified later even by an administrator.
3.Components
4.Versions

For instance, take a web based application; there are 10 requirements that need to be developed. There will be 5 more features added to it later on. You can choose to create the project as "Sample_Test" version 1 and Version 2. Version1 with 10 requirements, version 2 with 5 new ones. For version 1 if 5 of the requirements belong to Module 1 and the rest of them belong to module 2. The module 1 and module 2 can be created as separate units
Note: Project creation and management in JIRA is an admin task.

Functions of System Administration:

Audit Log: Under Audit Log, you can view all the details about the issue created, and the changes made in the issues.

Issue Linking: This will show whether your issues link with any other issue that is already present or created in the project also you can de-activate Issue linking from the panel itself

Mail in JIRA: Using Mail system in admin you can mail issues to an account on a POP or IMAP mail server or messages written to the file system generated by an external mail service.

Events: An event describes the status, the default template and the notification scheme and workflow transition post function associations for the event. The events are classified in two a System event (JIRA defined events) and Custom event (User defined events).

Watch list: JIRA allows you to watch a particular issue, which tells you about the notifications of any updates relating to that issue.

Issue Collectors: In the form of JIRA issues, an issue collector allows you to gather feedback on any website.

Development Tools: You can also connect your development tools to JIRA using this admin function. You have to enter the URL of the application to connect with JIRA.

Advantages of JIRA Tutorial:

-- Easy to use Software Testing Tutorials
-- Amazing to manage and track bugs
-- Customize a workflow to fit your project need
-- Plug-ins and adaptors for your device environment
-- Better reporting for better analysis
-- Popular alternate to QC and other expensive tools