3.8.2 Earned Value Analysis

Q. 3.8.3 What is Earned Value Analysis in Project tracking concept? (Ref. sec. 3.8.2) (5 Marks)

- Earned Value Analysis (EVA) is the key technique used in Project Management.
- The purpose of Earned Value Analysis is to understand how the project is progressing.
- EVA used to estimate the progress of a project based on earnings or money and schedules as calculated on the basis of EVA.
- In simple words Earned Value is a measure of 'Progress' to evaluate 'Percentage of Completeness'

3.8.2(A) Features of EVA

- Earned Value Analysis's objective is to measure project performance in terms of scope, cost and time.
- EVA is used to evaluate project health and project performance.
- Earned Value Analysis is also used for monitoring the progress of a software project / team which involves tasks allocated to the Project Schedule.
- Total time required to complete the project is calculated and each task is given an Earned Value based on its estimated (%) out of the total.

3.8.2(B) Need for EVA

- EVA provides the measures of project process of different tasks associated with project, so it is a single way of measuring each and every point in the project.
- It also provides a signal for corrective action in the project. The types of signals can be the following :

  1. **Time to recover**

     If the project is not going as per plan and it is found that there is some delay in the project. Then at this time, situation is needed to be taken care by finding out the reasons that are causing delay and by taking the required corrective actions.

  2. **Request for additional funds/resources**

     While there is time to recover, the need for extra resources or money can be calculated with an early warning.

Key Elements of EVA

1. **Planned Value (PV)**

   The allocated cost for the project which is approved. It was known as Budgeted Cost of Work Scheduled (BCWS).

2. **Earned Value (EV)**

   The budgeted value of the completed work packages. It used to be known as Budgeted Cost of Work Performance at a specified point (BCWP).

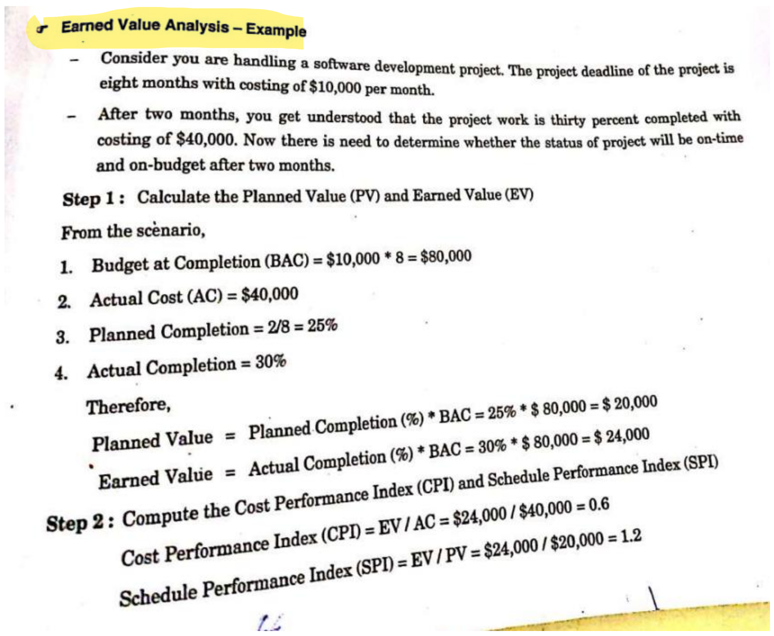
3. **Actual Cost (AC)**

   The actual cost involved during the execution of project work. It was previously called Actual Cost of Work Performed (ACWP).

Tools and techniques

There are several software packages available which are listed are as follows :

1. Schedule maker
2. Planisware OPX2
3. RiskTrak



Earned Value Analysis – Example

- Consider you are handling a software development project. The project deadline of the project is eight months with costing of $10,000 per month.
- After two months, you get understood that the project work is thirty percent completed with costing of $40,000. Now there is need to determine whether the status of project will be on-time and on-budget after two months.

Step 1 : Calculate the Planned Value (PV) and Earned Value (EV)

From the scenario,

1. Budget at Completion (BAC) = $10,000 * 8 = $80,000
2. Actual Cost (AC) = $40,000
3. Planned Completion = 2/8 = 25%
4. Actual Completion = 30%

Therefore,

Planned Value = Planned Completion (%) * BAC = 25% * $ 80,000 = $ 20,000

Earned Value = Actual Completion (%) * BAC = 30% * $ 80,000 = $ 24,000

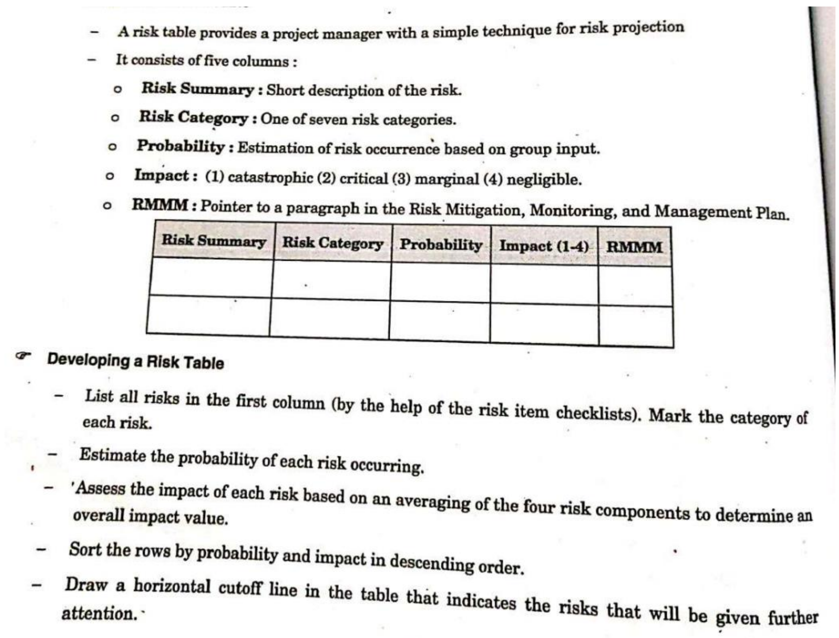
Step 2 : Compute the Cost Performance Index (CPI) and Schedule Performance Index (SPI)

Cost Performance Index (CPI) = EV / AC = $24,000 / $40,000 = 0.6

Schedule Performance Index (SPI) = EV / PV = $24,000 / $20,000 = 1.2

1.2

Module 5:Risk Analysis

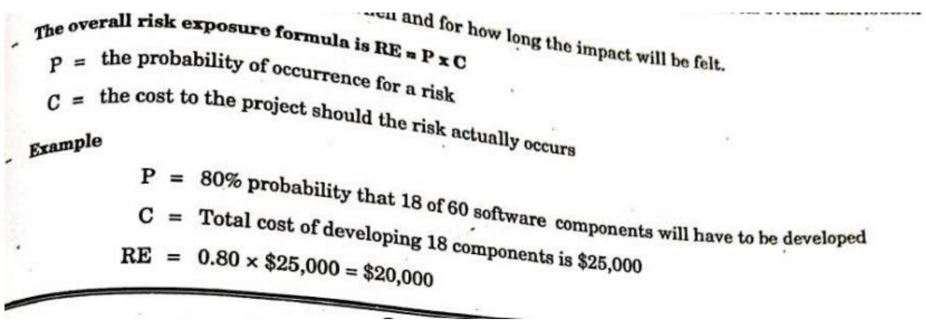


- A risk table provides a project manager with a simple technique for risk projection
- It consists of five columns :
  - **Risk Summary :** Short description of the risk.
  - **Risk Category :** One of seven risk categories.
  - **Probability :** Estimation of risk occurrence based on group input.
  - **Impact :** (1) catastrophic (2) critical (3) marginal (4) negligible.
  - **RMMM :** Pointer to a paragraph in the Risk Mitigation, Monitoring, and Management Plan.

Risk Summary	Risk Category	Probability	Impact (1-4)	RMMM

Developing a Risk Table

- List all risks in the first column (by the help of the risk item checklists). Mark the category of each risk.
- Estimate the probability of each risk occurring.
- 'Assess the impact of each risk based on an averaging of the four risk components to determine an overall impact value.
- Sort the rows by probability and impact in descending order.
- Draw a horizontal cutoff line in the table that indicates the risks that will be given further attention.



and for how long the impact will be felt.

The overall risk exposure formula is RE = P x C

P = the probability of occurrence for a risk

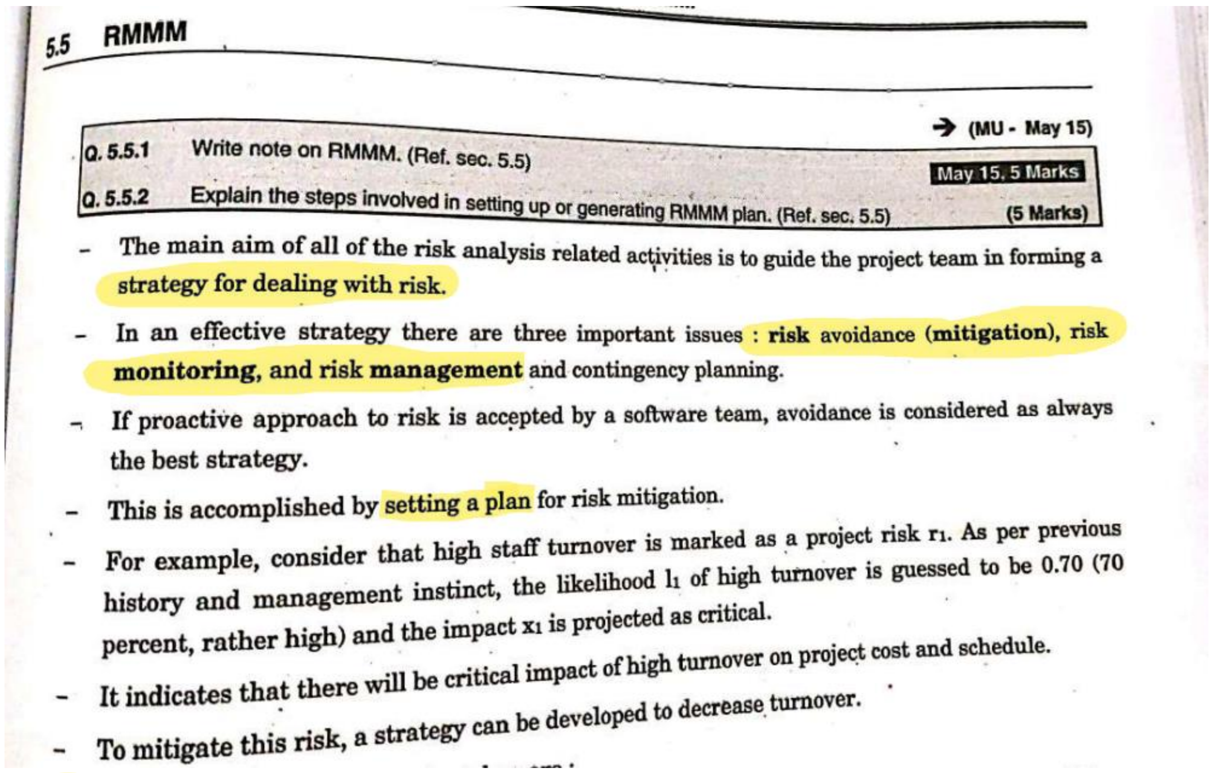
C = the cost to the project should the risk actually occurs

Example

P = 80% probability that 18 of 60 software components will have to be developed

C = Total cost of developing 18 components is $25,000

RE = 0.80 × $25,000 = $20,000



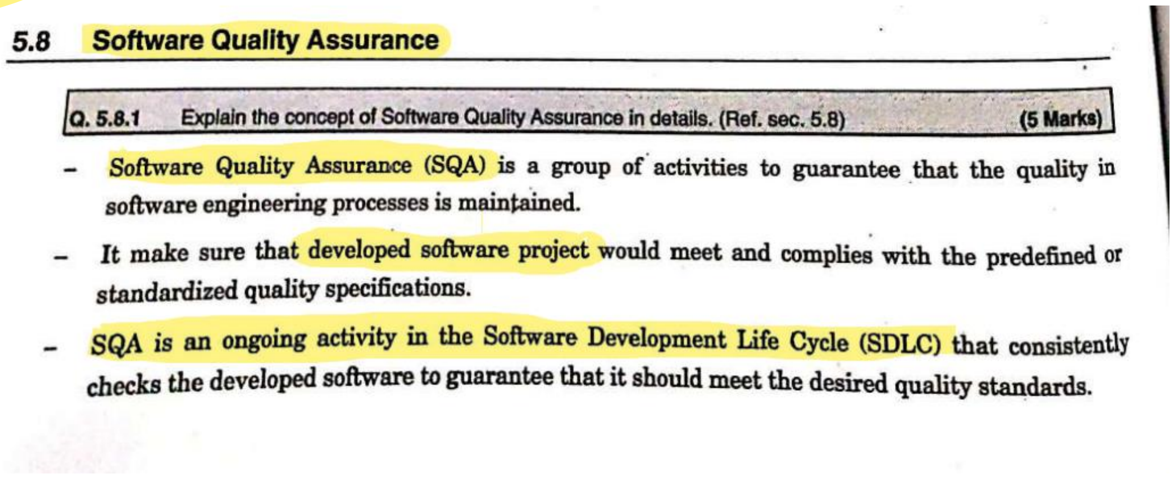
5.5 RMMM

> → (MU - May 15)

Q. 5.5.1	Write note on RMMM. (Ref. sec. 5.5)	May 15, 5 Marks
Q. 5.5.2	Explain the steps involved in setting up or generating RMMM plan. (Ref. sec. 5.5)	(5 Marks)

- The main aim of all of the risk analysis related activities is to guide the project team in forming a strategy for dealing with risk.
- In an effective strategy there are three important issues : **risk** avoidance (**mitigation**), risk **monitoring**, and risk **management** and contingency planning.
- If proactive approach to risk is accepted by a software team, avoidance is considered as always the best strategy.
- This is accomplished by setting a plan for risk mitigation.
- For example, consider that high staff turnover is marked as a project risk r_1. As per previous history and management instinct, the likelihood l_1 of high turnover is guessed to be 0.70 (70 percent, rather high) and the impact x_1 is projected as critical.
- It indicates that there will be critical impact of high turnover on project cost and schedule.
- To mitigate this risk, a strategy can be developed to decrease turnover.

SQA



5.8 Software Quality Assurance

Q. 5.8.1	Explain the concept of Software Quality Assurance in details. (Ref. sec. 5.8)	(5 Marks)

- Software Quality Assurance (SQA) is a group of activities to guarantee that the quality in software engineering processes is maintained.
- It make sure that developed software project would meet and complies with the predefined or standardized quality specifications.
- SQA is an ongoing activity in the Software Development Life Cycle (SDLC) that consistently checks the developed software to guarantee that it should meet the desired quality standards.

- SQA practices are adapted in most types of software development industries.
- SQA incorporates and implements software testing methodologies to test the software. Rather than every time checking for quality after completion of each project, SQA processes test the quality in each phase of development.
- With SQA, the software development activities moves into the next level only if the present/prior level complies with the required quality standards.
- SQA generally works on one or more engineering standards that assist in building software quality guiding principle and execution strategies.
- Once the processes have been defined and implemented, Quality Assurance has the following tasks :
  (a) To identify the weaknesses in the processes,
  (b) Correct those weaknesses to constantly improve the process.

It includes the following activities

(1) Process definition and accomplishment
(2) Auditing
(3) Guidance

ocesses could be

1) Software Development method
2) Project Management
) Configuration Management
) Requirements Development/Management
Estimation
Software Design
Testing, etc.

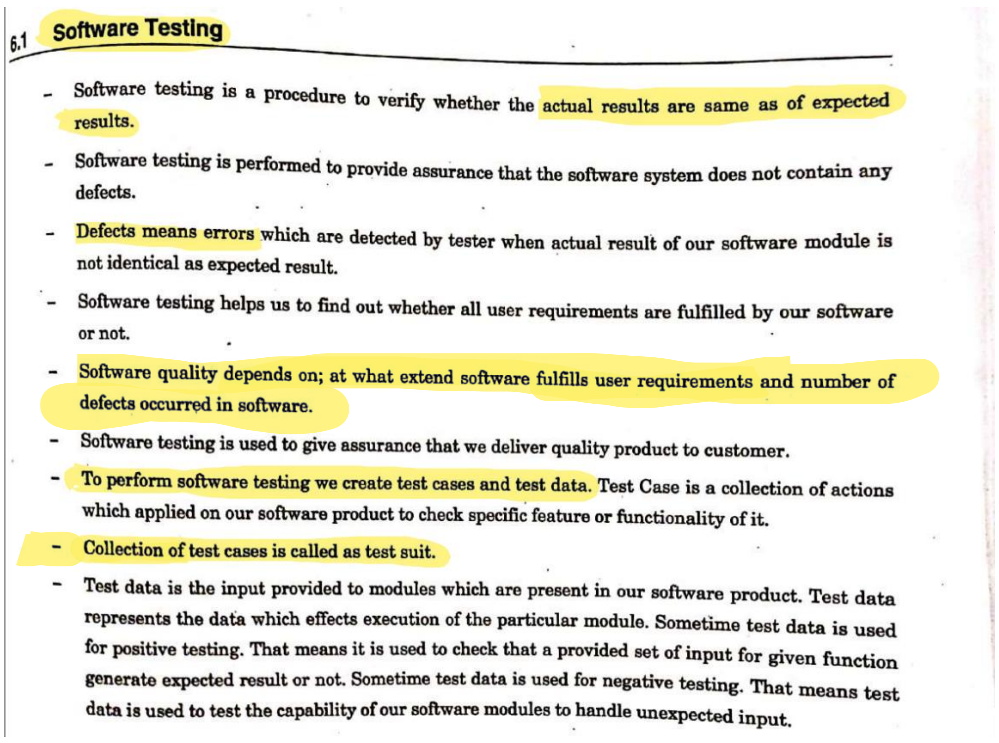
5.15 Formal Technical Review (FTR)

Q. 5.15.1 What is FTR in SQA? What are its objectives? Explain steps in FTR. (Ref. sec. 5.15) → (MU - Dec. 15, May 16, May 17, May 18) Dec. 15, May 17, 10 Marks

Q. 5.15.2 Explain FTR. (Ref. sec. 5.15) May 16, May 18, 3 Marks

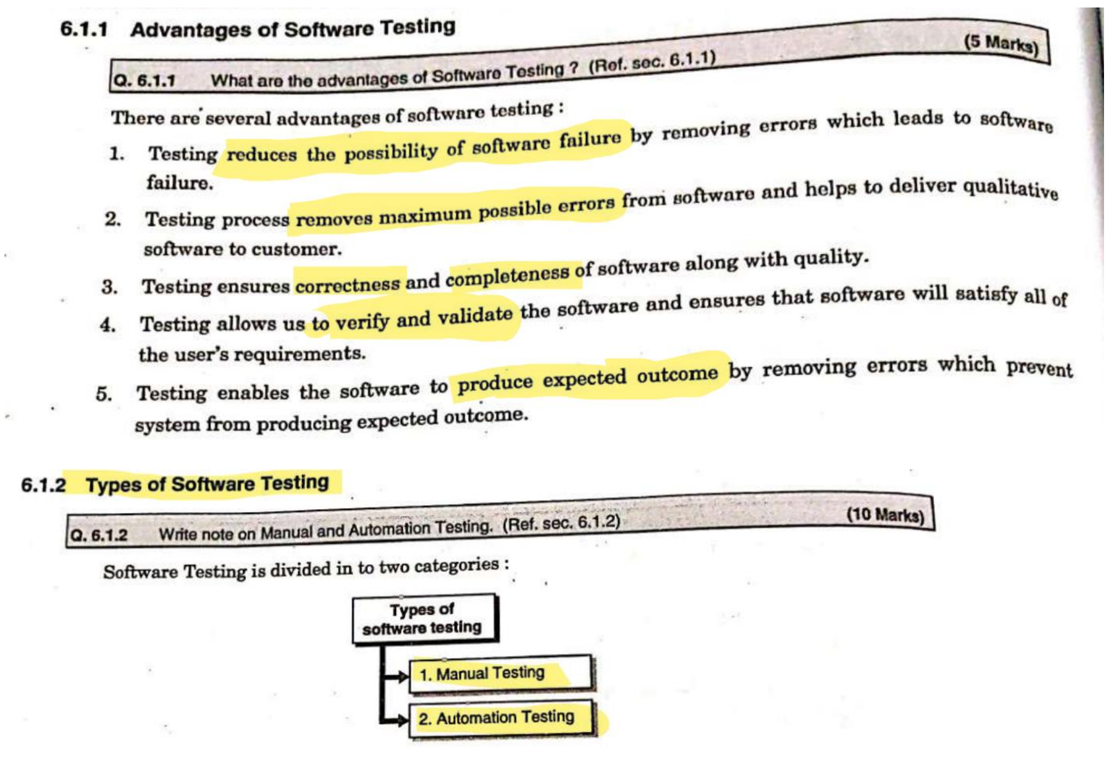
- A formal technical review is a process performed to give assurance of software quality.
- This testing activity is done by software engineers and other persons.
- The **objectives** of the Formal Technical Review are :
  (1) To find out errors in function, logic, or coding in the software.
  (2) To check that the software fulfils the requirements for which it is built.
  (3) To give assurance that software has been designed as per customer requirements.
  (4) To create software developed in uniform order.
  (5) To create more manageable project.
- The Formal Technical Review is used while providing training and it is also used as an example for junior developers to study different ways for software analysis, design, and development.
- The Formal Technical Reviews is a class of reviews that contains walkthroughs, inspections, round-robin reviews and other small technical assessments of software.
- Every Formal Technical Review is performed as a meeting.
- If Formal Technical Reviews is planned, controlled, and attended in correct way then only it becomes successful.

Module 6:Software Testing



6.1 Software Testing

- Software testing is a procedure to verify whether the actual results are same as of expected results.
- Software testing is performed to provide assurance that the software system does not contain any defects.
- Defects means errors which are detected by tester when actual result of our software module is not identical as expected result.
- Software testing helps us to find out whether all user requirements are fulfilled by our software or not.
- Software quality depends on; at what extend software fulfills user requirements and number of defects occurred in software.
- Software testing is used to give assurance that we deliver quality product to customer.
- To perform software testing we create test cases and test data. Test Case is a collection of actions which applied on our software product to check specific feature or functionality of it.
- Collection of test cases is called as test suit.
- Test data is the input provided to modules which are present in our software product. Test data represents the data which effects execution of the particular module. Sometime test data is used for positive testing. That means it is used to check that a provided set of input for given function generate expected result or not. Sometime test data is used for negative testing. That means test data is used to test the capability of our software modules to handle unexpected input.



6.1.1 Advantages of Software Testing

Q. 6.1.1 What are the advantages of Software Testing ? (Ref. sec. 6.1.1) **(5 Marks)**

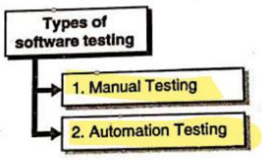
There are several advantages of software testing :

1. Testing reduces the possibility of software failure by removing errors which leads to software failure.
2. Testing process removes maximum possible errors from software and helps to deliver qualitative software to customer.
3. Testing ensures correctness and completeness of software along with quality.
4. Testing allows us to verify and validate the software and ensures that software will satisfy all of the user's requirements.
5. Testing enables the software to produce expected outcome by removing errors which prevent system from producing expected outcome.

6.1.2 Types of Software Testing

Q. 6.1.2 Write note on Manual and Automation Testing. (Ref. sec. 6.1.2) **(10 Marks)**

Software Testing is divided in to two categories :



→ 1. **Manual Testing**

- Manual Testing is one of the type of Software Testing in which Tester treat himself as end user and use every module of application from login module to log out module to check application behavior according to requirement provided by user.
- In manual testing, tester manually runs all the test cases without help of any automation testing tools.
- Manual Testing is the very basic type of all testing types and supports to search the bug (error) in the software system.
- First we do manual testing for any new application before going for automation testing. We perform feasibility study of automation testing in manual testing i.e. do the analysis of benefits of performing automation testing for our project to take the decision that whether to do automation testing or not.
- For performing manual testing, we do not need knowledge of any testing tool.

- List of manual testing is as follow :

(i) Unit testing

(ii) Integration testing

(iii) System testing

(iv) Acceptance testing

2. **Automation Testing**

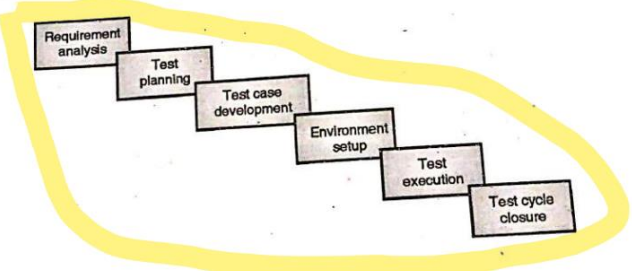
- Automation Testing is a process in which we use automation tools to run our test cases to find out bugs from our software product.
- The automation software is able to enter test data into the software on which we perform testing and it also matches the expected and actual results to create test reports.
- Automation testing requires considerable amount of money and resources such as employees, testing tools etc.
- After change in requirement or occurrence of error, we need to make changes in code and we require executing same test suite again and again.
- We can record test suite by using test automation tools and re-play it when needed.
- Automation testing does not require human interaction to perform testing like manual testing.
- Aim of Automation testing is to decrease number of test cases to be executed manually but not replace Manual Testing completely.
- To perform automation testing we require generation of test cases, test data and test script.
- Automation testing is performed for project if requirements of projects are stable at some extent i.e. requirements are not frequently changing.
- List of some automation tools

(i) Selenium	(ii) Mentis
(iii) Quality Test Professional (QTP)	(iv) Buxila
(v) HP ALM (Application Lifecycle Management)	

6.1.6 Software Testing Process

Q. 6.1.6 Explain the software testing process in detail. **(10 Marks)**
(Ref. sec. 6.1.6)

- Software testing process is a sequence of various activities which helps to certify software system.
- Software testing process activities are : Requirement analysis, Test planning, Test case development, environment set up, Test execution, Test cycle closures.
- Each of the activity of software testing process has defined entry and exit criteria.



Unit Testing

- Unit Testing is a level of software testing where individual units / components of a software are tested.
- The purpose is to validate that each unit of the software performs as designed.
- A unit is the smallest testable part of any software. It usually has one or a few inputs and usually a single output.
- In procedural programming, a unit may be an individual program, function, procedure, etc.
- In object-oriented programming, the smallest unit is a method, which may belong to a base / super class, abstract class or derived/ child class. (Some treat a module of an application as a unit.

→ **(MU - Dec. 15, Dec. 16)**

Q. 6.4.1 Explain integration testing. (Ref. sec. 6.4) **Dec. 15, Dec. 16, 5 Marks**

- Integration testing is a method of software testing in which all units of software under test are integrated and tested as a single group.
- It is always done after unit testing is applied.
- It mainly focuses on testing data communication among all units of system.

and we need
so developer
ule A. This

Module B
vill return

Marks)

- It makes use of mythologies as Big bang approach and incremental approach.
- Incremental approach can be either top-down, bottom-up or combination of both.
- The main objective of integration testing is to determine faults in communication between, integrated units.

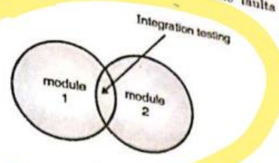


Fig. 6.4.1

Following are reasons for performing Integration testing :

- A Module is developed by software developer whose level of understanding and programming logic may not be same as of other software developers from project team.
- So, while performing Integration Testing, we required to check whether software module works efficiently with other modules or not.
- While developing single module, there is risk of changes in requirements by the clients.
- We may not perform unit testing for this new requirements and so performing integration testing of software is needed.

6.5.3 Alpha and Beta Testing

Q. 6.5.2 Write note on Alpha and Beta Testing. (Ref. sec. 6.5.3) **(10 Marks)**

(1) Alpha testing

- Alpha Testing is software testing which is done to find out bugs before deploying the software application to end user.
- Alpha testing is a type of acceptance testing.
- This testing is called as an alpha testing since it is performed when development phase of software application is near to Beta Testing.
- The objective of alpha testing is to refine the software product by finding (and fixing) the bugs that were not discovered through previous tests.
- Alpha testing is typically performed by in-house software engineers or QA staff.
- It is the final testing stage before the software is released into the real world.
- Alpha testing is performed in two phases :
  - (i) In first phase software is tested by development team members. They perform debugging of software to catch bugs quickly.
  - (ii) In second phase software is tested by software quality analyst team for additional testing in actual user's environment setup.

(2) Beta testing

- Beta testing is testing which is performed at the location of customer. In this testing actual as well as intended users will test the software to determine whether the software is satisfying their needs and expectations.
- Beta testing allows users to test software before it is released to public.
- Beta testing minimizes the product failure risks and delivers qualitative product through customer validation. It gives direct feedback from the user.
- It ensures reliability, security, robustness etc. from user's perspective.
- Types of Beta testing are : Traditional, public, technical, focused and past release.
- Beta testing is also called as User Acceptance Testing, Customer acceptance Testing, Customer Validation testing and Pre-Release testing.
- Beta testing gives assurance that we deliver quality software to our customers by testing the product under various situations in real world environment that can't be created in a lab setting.

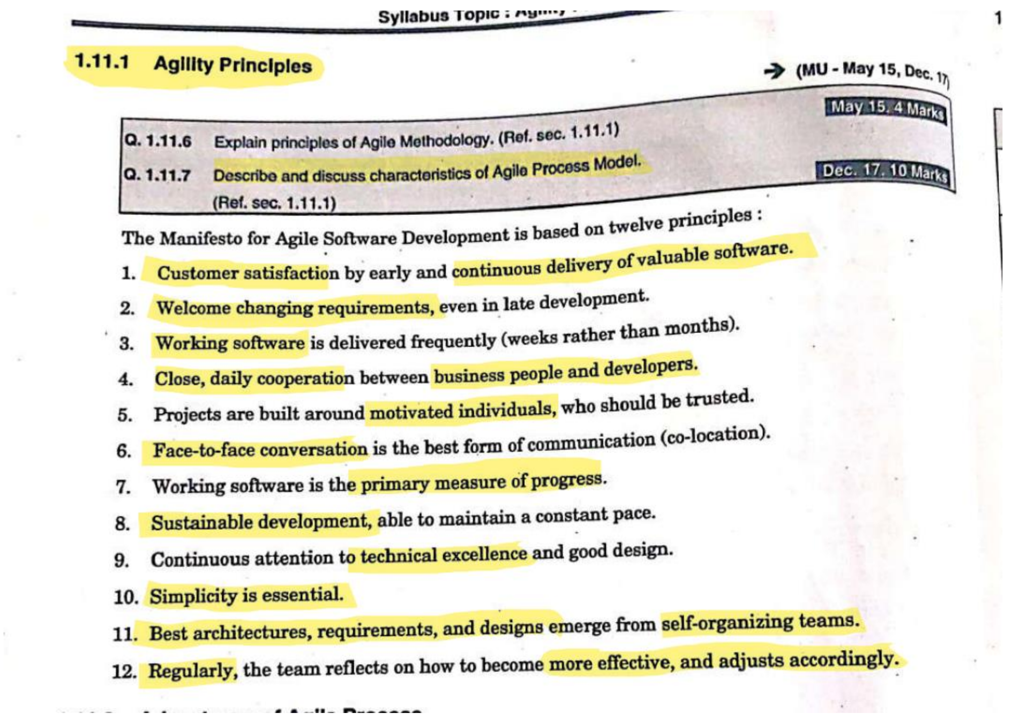
Fig. 6.6.1 : Software testing methods

→ **1. Black Box Testing**

In this type of testing, we check working of our software project by running it. We do not need to check code of our software in this testing type. **System testing** is included in black box testing

→ **2. White Box Testing**

In white box testing we check internal working of our code. So in this type of testing tester need deep knowledge of programming language which is used in our software product.



1.11.1 Agility Principles

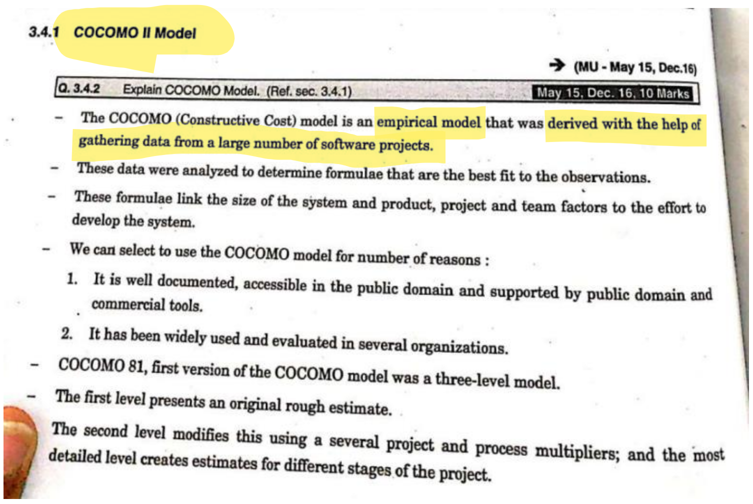
→ (MU - May 15, Dec. 17)

Q. 1.11.6	Explain principles of Agile Methodology. (Ref. sec. 1.11.1)	May 15, 4 Marks
Q. 1.11.7	Describe and discuss characteristics of Agile Process Model. (Ref. sec. 1.11.1)	Dec. 17, 10 Marks

The Manifesto for Agile Software Development is based on twelve principles :

1. Customer satisfaction by early and continuous delivery of valuable software.
2. Welcome changing requirements, even in late development.
3. Working software is delivered frequently (weeks rather than months).
4. Close, daily cooperation between business people and developers.
5. Projects are built around motivated individuals, who should be trusted.
6. Face-to-face conversation is the best form of communication (co-location).
7. Working software is the primary measure of progress.
8. Sustainable development, able to maintain a constant pace.
9. Continuous attention to technical excellence and good design.
10. Simplicity is essential.
11. Best architectures, requirements, and designs emerge from self-organizing teams.
12. Regularly, the team reflects on how to become more effective, and adjusts accordingly.

COCOMO II Model:



3.4.1 COCOMO II Model

→ (MU - May 15, Dec.16)

Q. 3.4.2	Explain COCOMO Model. (Ref. sec. 3.4.1)	May 15, Dec. 16, 10 Marks

- The COCOMO (Constructive Cost) model is an empirical model that was derived with the help of gathering data from a large number of software projects.
- These data were analyzed to determine formulae that are the best fit to the observations.
- These formulae link the size of the system and product, project and team factors to the effort to develop the system.
- We can select to use the COCOMO model for number of reasons :
  1. It is well documented, accessible in the public domain and supported by public domain and commercial tools.
  2. It has been widely used and evaluated in several organizations.
- COCOMO 81, first version of the COCOMO model was a three-level model.
- The first level presents an original rough estimate.
- The second level modifies this using a several project and process multipliers; and the most detailed level creates estimates for different stages of the project.

- Fig. 3.4.1 shows COCOMO II sub-models as well as where they are used.

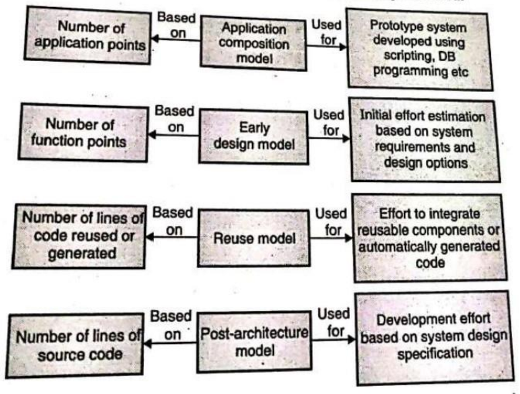

Number of application points
Based on
Application composition model
Used for
Prototype system developed using scripting, DB programming etc
Number of function points
Based on
Early design model
Used for
Initial effort estimation based on system requirements and design options
Number of lines of code reused or generated
Based on
Reuse model
Used for
Effort to integrate reusable components or automatically generated code
Number of lines of source code
Based on
Post-architecture model
Used for
Development effort based on system design specification

Fig. 3.4.1 : The COCOMO II model

The Constructive Cost Model (COCOMO)

- COCOMO is one of the most widely used software estimation models in the world
- It was developed by Barry Boehm in 1981
- COCOMO predicts the effort and schedule for a software product development based on inputs relating to the **size** of the software and a number of **cost drivers** that affect productivity

COCOMO Models

- COCOMO has three different models that reflect the **complexity**:
  - the Basic Model
  - the Intermediate Model
  - and the Detailed Model

Basic Model

- Applicable to small to medium sized software projects
- Use for a quick and rough estimates
- Three modes of software development are considered
  - Organic
  - Semi-detached
  - Embedded

Organic Mode

- A small team of experienced programmers develop software in a very familiar environment
- Require little Innovation
- Size range (0-50 KLOC)

Embedded mode

- Project has tight constraints
- Hard to find experienced persons
- Require significant Innovation
- Development environment is complex
- Size range (over 300 KLOC)

Semi-detached mode

- An intermediate mode between the organic mode and embedded mode
- Depending on the problem at hand, the team include the mixture of experienced and less experienced people
- Require medium Innovation
- Development environment is medium
- Size range (50 - 300 KLOC)

Basic COCOMO Model: Equation

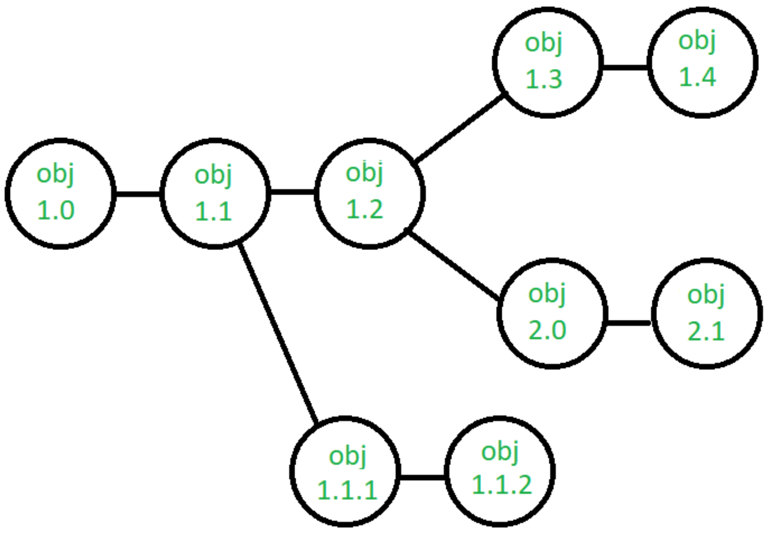
Mode	Effort	Schedule
Organic	$E=2.4*(KDSI)^{1.05}$	$TDEV=2.5*(E)^{0.38}$
Semidetached	$E=3.0*(KDSI)^{1.12}$	$TDEV=2.5*(E)^{0.35}$
Embedded	$E=3.6*(KDSI)^{1.20}$	$TDEV=2.5*(E)^{0.32}$

System configuration management(SCM)

System Configuration Management (SCM) is an arrangement of exercises which controls change by recognizing the items for change, setting up connections between those things, making/characterizing instruments for overseeing diverse variants, controlling the changes being executed in the current framework, inspecting and revealing/reporting on the changes made.

Processes involved in SCM – Configuration management provides a disciplined environment for smooth control of work products. It involves the following activities:

1. **Identification and Establishment – Identifying the configuration items from products that compose baselines at given points in time (a baseline is a set of mutually consistent Configuration Items, which has been formally reviewed and agreed upon, and serves as the basis of further development). Establishing relationship among items, creating a mechanism to manage multiple level of control and procedure for change management system.**
2. **Version control – Creating versions/specifications of the existing product to build new products from the help of SCM system. A description of version is given below:**



3. **Suppose after some changes, the version of configuration object changes from 1.0 to 1.1. Minor corrections and changes result in versions 1.1.1 and 1.1.2, which is followed by a major update that is object 1.2. The development of object 1.0 continues through 1.3 and 1.4, but finally, a noteworthy change to the object results in a new evolutionary path, version 2.0. Both versions are currently supported.**
4. **Change control – Controlling changes to Configuration items (CI). The change control process is explained in Figure below:**

System Configuration Management (SCM) is a software engineering practice that focuses on managing the configuration of software systems and ensuring that software components are properly controlled, tracked, and stored. It is a critical aspect of software development, as it helps to ensure that changes made to a software system are properly coordinated and that the system is always in a known and stable state.

The key objectives of SCM are to:

1. Control the evolution of software systems: SCM helps to ensure that changes to a software system are properly planned, tested, and integrated into the final product.
2. Enable collaboration and coordination: SCM helps teams to collaborate and coordinate their work, ensuring that changes are properly integrated and that everyone is working from the same version of the software system.
3. Provide version control: SCM provides version control for software systems, enabling teams to manage and track different versions of the system and to revert to earlier versions if necessary.
4. Facilitate replication and distribution: SCM helps to ensure that software systems can be easily replicated and distributed to other environments, such as test, production, and customer sites.
5. SCM is a critical component of software development, and effective SCM practices can help to improve the quality and reliability of software systems, as well as increase efficiency and reduce the risk of errors.

The main advantages of SCM are:

1. Improved productivity and efficiency by reducing the time and effort required to manage software changes.
2. Reduced risk of errors and defects by ensuring that all changes are properly tested and validated.
3. Increased collaboration and communication among team members by providing a central repository for software artifacts.
4. Improved quality and stability of software systems by ensuring that all changes are properly controlled and managed.

The main disadvantages of SCM are:

1. Increased complexity and overhead, particularly in large software systems.
2. Difficulty in managing dependencies and ensuring that all changes are properly integrated.
3. Potential for conflicts and delays, particularly in large development teams with multiple contributors.

SCM Chatgpt:

Software Configuration Management (SCM) is a set of practices and tools used in software engineering to manage and control changes to software artifacts throughout the software development lifecycle. SCM helps ensure the consistency, integrity, and traceability of software components, making it an essential part of the overall software development process. Here are the key aspects and components of SCM in software engineering:

Version Control: Version control systems (VCS) are fundamental to SCM. They track changes to source code, documentation, and other artifacts over time. Popular version control systems include Git, Subversion, and Mercurial. VCS allows multiple developers to collaborate on the same project while maintaining a history of changes.

Configuration Identification: This involves identifying and labeling all the software components, such as source code files, documents, and resources, that are part of a software project. Each version or release is uniquely identified and tagged to maintain a clear record.

Change Control: Change control or change management is the process of managing and tracking changes to the software configuration items. It includes requesting, reviewing, approving, and implementing changes. Proper change control ensures that modifications are properly documented and assessed for their impact.

Configuration Control: Configuration control focuses on ensuring that the software configuration remains stable during development. It involves establishing a baseline or a reference point for the software at different stages (e.g., alpha, beta, release) to prevent unauthorized changes.

Configuration Status Accounting: This aspect deals with tracking and reporting the current status and history of each software component and the overall configuration. It helps in maintaining an audit trail for software changes.

Build Management: This involves defining the process and tools used to compile, build, and package the software. It ensures that the build process is consistent and repeatable, making it easier to reproduce a specific version of the software.

Release Management: Release management focuses on planning, scheduling, and coordinating the release of software versions to customers or end-users. It involves creating release notes, packaging the software, and deploying it to various environments.

Branching and Merging: In version control systems, branching and merging are crucial techniques for managing concurrent development efforts, bug fixes, and feature development. Branches allow different development paths without affecting the main codebase.

Integration Management: Integration management ensures that changes from different developers or teams are integrated smoothly and do not introduce conflicts or regressions. Continuous integration and automated testing are often used to support this process.

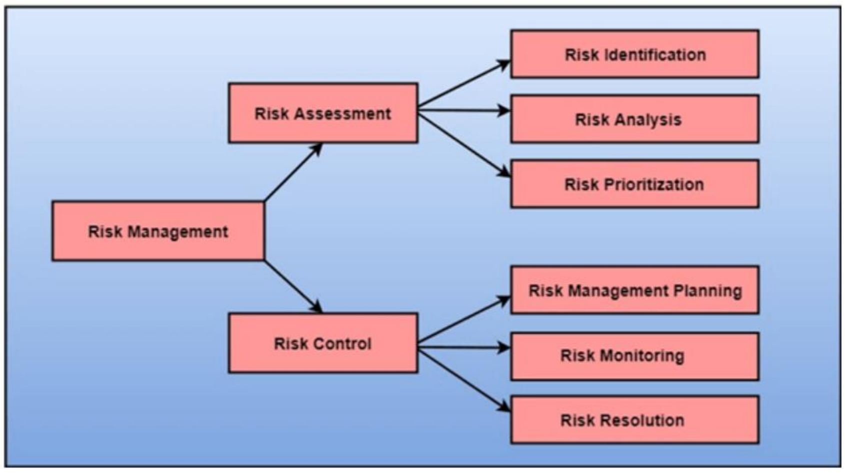
Audit and Compliance: SCM systems help ensure compliance with organizational, industry, or regulatory standards. They provide the ability to audit changes and demonstrate traceability.

Backup and Recovery: Ensuring that software artifacts and configurations are backed up regularly is a fundamental SCM practice. This helps in recovering from disasters or accidental data loss.

Documentation Management: Managing documentation, such as design specifications, user manuals, and release notes, is also an important part of SCM. It ensures that documentation is kept up-to-date and consistent with the software.

Effective SCM practices help software development teams manage complexity, control changes, reduce errors, and ensure that software is delivered in a reliable and consistent manner. It plays a critical role in ensuring the quality, reliability, and maintainability of software throughout its lifecycle.

Risk Management:



The probability of the risk might be determined as **very low (0-10%), low (10-25%), moderate (25-50%), high (50-75%) or very high (+75%).**