1. Demonstrate Unit Testing on a Login Module.
= Introduction
Unit testing is the process of testing individual functions or modules to ensure they work as expected. Here, we demonstrate unit testing on a Login Module that checks username and password validation.
Demonstration Points

1. Create the Login Function
   Define a simple login() function that checks if the username and password are correct or empty, returning suitable messages.
2. Use the unittest Framework
   Import the unittest module in Python to write and organize test cases systematically.
3. Write Multiple Test Cases
   Include cases for successful login, invalid credentials, and empty fields to cover all possible scenarios.
4. Use Assertions
   Apply assertEqual() to compare the expected output with the actual function output for each test case.
5. Run and Verify Results
   Execute the test file. If all tests pass, it confirms the login function behaves correctly in all tested conditions.

2. Apply Black Box Testing on an E-Commerce Cart.
= Introduction
Black Box Testing checks the system's functionality without seeing its internal code. In an E-commerce Cart, it ensures user actions work properly.
Points

1. Add to Cart – Verify products are added correctly.
2. Remove Item – Ensure removed items no longer appear.
3. Update Quantity – Check quantity changes update totals.
4. Cart Total – Validate total price with discounts and tax.
5. Empty Cart – Confirm it shows "Cart is empty" after all removals.

3. Use Equivalent Partitioning to test data inputs.
= Introduction
Equivalent Partitioning is a Black Box Testing technique that divides input data into valid and invalid groups. Testing one value from each group reduces repetition while ensuring full coverage.
Points

1. Identify Input Ranges – Divide input values into valid and invalid classes.
2. Select Test Cases – Pick one representative value from each class.
3. Example – For age input (1–100): valid = 25, invalid = -5, 150.
4. Reduce Redundancy – Fewer test cases are needed but still ensure all scenarios are covered.
5. Check System Response – Verify correct handling of both valid and invalid inputs.

4. Illustrate how Validation Testing is performed.
= Introduction
Validation Testing ensures that the final software meets user needs and business requirements. It checks whether the product built is the right product as per the intended use.
Points

1. Requirement Review – Verify that all user requirements are clearly defined and testable before testing begins.
2. System Testing – Test the complete system to ensure all integrated modules work together as expected.
3. Acceptance Testing – Conducted by end users or clients to confirm the system satisfies real-world needs.
4. User Interface Testing – Check if the interface is user-friendly, consistent, and meets design expectations.
5. Performance and Reliability Checks – Validate that the system performs efficiently under expected load and remains stable.

5. Apply Regression Testing after Bug Fixing.

= Introduction

Regression Testing ensures that recent bug fixes or code changes do not break existing features. It is done after modifications to verify overall system stability.

Points

1. Identify Fixed Areas – Locate modules or functions where bugs were corrected.
2. Select Affected Test Cases – Choose and rerun test cases related to the modified code.
3. Re-execute Previous Tests – Run earlier successful tests to ensure other features still work properly.
4. Automate Tests – Use automated testing tools for faster and consistent regression checks.
5. Analyze Results – Compare outputs before and after bug fixes to detect any new or recurring issues.

6. Analysis the difference between varification and validation.

=

Point	Verification	Validation
1.	Ensures the product is built correctly.	Ensures the right product is built.
2.	Checks if software meets design specifications.	Checks if software meets user needs.
3.	Done through reviews and inspections.	Done through testing and evaluation.
4.	Performed during the development phase.	Performed after development completion.
5.	Answers “Are we building it right?”	Answers “Are we building the right thing?”

7. Distinguish between Alpha and Beta Testing.

=

Point	Alpha Testing	Beta Testing
1.	Conducted by internal testers within the organization.	Conducted by real users outside the organization.
2.	Performed in a controlled lab environment.	Performed in the real user environment.
3.	Aims to find bugs before public release.	Aims to gather user feedback and usability issues.
4.	Occurs before Beta Testing.	Occurs after Alpha Testing.
5.	Issues are fixed immediately by developers.	Feedback is collected for future improvements.

8. Examine how quality affects testing.

= Introduction

Quality directly influences how software testing is planned, executed, and evaluated. High-quality standards ensure reliable, efficient, and user-satisfying software.

Points

1. Defines Testing Standards – Quality goals determine what to test and how much testing is needed to meet expectations.
2. Improves Test Coverage – High-quality requirements lead to clear, detailed test cases that cover all features.
3. Reduces Defects – A focus on quality during development lowers the number of bugs found during testing.
4. Enhances Reliability – Quality-based testing ensures the software performs consistently under various conditions.
5. Increases Customer Satisfaction – Delivering a well-tested, high-quality product builds user trust and confidence.

9. Explain how ISO 9000 standards contribute to software quality improvement.
= Introduction
ISO 9000 is a set of international standards for quality management systems. In software development, it helps ensure that processes are well-defined, consistent, and focused on continuous improvement.
Points

1. Standardized Processes – ISO 9000 defines clear procedures for software development, ensuring uniform quality across all projects.
2. Quality Assurance Framework – It establishes guidelines for planning, monitoring, and verifying software quality at every stage.
3. Continuous Improvement – Regular audits and feedback help identify weak areas and improve processes over time.
4. Customer Focus – The standards emphasize meeting customer requirements and enhancing satisfaction through reliable software.
5. Documentation and Accountability – Proper documentation ensures traceability, making it easier to maintain and improve software quality.

10. Describe the five levels of Capability Maturity Model (CMM) and their importance in software improvement.
= Introduction
The Capability Maturity Model (CMM) defines five levels of process maturity in software development. It helps organizations improve software quality, predictability, and efficiency through structured process enhancement.
Points

1. Level 1 – Initial
   Processes are unpredictable and unorganized. Success depends on individual effort rather than a defined system.
2. Level 2 – Repeatable
   Basic project management practices are established, allowing similar projects to be repeated with consistent results.
3. Level 3 – Defined
   Processes are documented, standardized, and integrated across the organization for better control and understanding.
4. Level 4 – Managed
   Detailed measurements and metrics are used to monitor software quality and process performance quantitatively.
5. Level 5 – Optimizing
   Continuous process improvement is emphasized through innovation, feedback, and proactive problem prevention.

11. Explain the need for proper Software Project Management to avoid project failure.
= Introduction
Software Project Management ensures that software projects are completed on time, within budget, and meet user requirements. Without proper management, projects often face delays, cost overruns, or complete failure.
Points

1. Clear Planning and Scheduling – Proper management helps define goals, timelines, and milestones to keep the project on track.
2. Resource Management – Ensures effective use of manpower, tools, and budget to prevent shortages or wastage.
3. Risk Control – Identifies potential risks early and plans strategies to minimize their impact on the project.
4. Quality Assurance – Maintains standards through regular testing and reviews to deliver reliable software.
5. Effective Communication – Promotes coordination among team members, clients, and stakeholders to avoid misunderstandings.

12. Explain how the size of software is estimated during project planning.
= Introduction
Estimating software size during project planning helps determine effort, time, and resources needed for development. It provides a basis for cost estimation and project scheduling.
Points
1. Lines of Code (LOC) – Size is measured by counting the total number of source code lines expected in the final product.
2. Function Point Analysis (FPA) – Estimates size based on the number of user functions like inputs, outputs, files, and interfaces.
3. Use Case Points (UCP) – Measures size by analyzing the number and complexity of use cases and actors in the system.
4. Expert Judgment – Experienced developers give estimates based on past similar projects and technical understanding.
5. Historical Data and Tools – Past project data and estimation tools help predict size more accurately using statistical methods.

13. Summarize the steps involved in Risk Mitigation, Monitoring, and Management.
= Introduction
Risk Mitigation, Monitoring, and Management aim to identify, control, and minimize risks that may affect a software project's success. Proper handling of risks ensures stability and timely delivery.
Points
1. Risk Identification – Recognize potential risks such as budget issues, delays, or technical failures early in the project.
2. Risk Analysis – Assess each risk's likelihood and impact to prioritize which ones need more attention.
3. Risk Mitigation Planning – Develop strategies and backup plans to reduce the chances or effects of critical risks.
4. Risk Monitoring – Continuously track identified risks and check for new ones during the project lifecycle.
5. Risk Control and Management – Implement mitigation plans and take corrective actions when risks occur.

14. Explain how Earned Value Analysis (EVA) helps track the progress of a software project.
= Introduction
Earned Value Analysis (EVA) is a project management technique that measures project progress by comparing planned work, actual work, and the value of work completed. It helps track performance in terms of cost and schedule.
Points
1. Planned Value (PV) – Represents the budgeted cost for the work planned up to a specific date.
2. Earned Value (EV) – Indicates the budgeted value of the actual work completed so far.
3. Actual Cost (AC) – The real cost incurred for the work performed to date.
4. Performance Indicators – EVA calculates key metrics like Cost Performance Index (CPI) and Schedule Performance Index (SPI) to measure efficiency.
5. Project Forecasting – Helps predict future costs, timelines, and performance trends to support corrective actions if needed.

15. Describe the components of a Network Scheduling Diagram and its role in project planning.
= Introduction
A Network Scheduling Diagram is a visual tool used in project planning to show the sequence, timing, and dependency of project activities. It helps managers plan, monitor, and control project progress effectively.
Points
1. Activities – Represented as nodes or arrows, they define specific tasks or work units in the project.
2. Events (Milestones) – Indicate the start or completion points of one or more activities.
3. Dependencies – Show the logical relationships between tasks, such as which activity must finish before another begins.
4. Critical Path – The longest path in the diagram that determines the minimum project completion time.
5. Slack or Float Time – Represents the amount of time an activity can be delayed without affecting the project's overall timeline.

16. Differentiate between Function Point Analysis (FPA) and Lines of Code (LOC) based estimation methods.
=

Point	Function Point Analysis (FPA)	Lines of Code (LOC)
1.	Measures software size based on functional requirements.	Measures software size based on number of code lines.
2.	Language-independent method.	Language-dependent method.
3.	Focuses on user functions like inputs, outputs, and files.	Focuses on programmer effort in writing code.
4.	Can be estimated early in the project, before coding.	Can be estimated only after coding begins.
5.	Suitable for project management and productivity analysis.	Suitable for technical and effort estimation.

17. Describe the significance of Quality Assurance (QA) in the Software Development Life Cycle (SDLC).
= Introduction
Quality Assurance (QA) ensures that software products meet defined quality standards throughout the Software Development Life Cycle (SDLC). It focuses on process improvement to prevent defects rather than just finding them.
Points

1. Process Standardization – QA establishes consistent processes and guidelines to ensure quality at every SDLC phase.
2. Defect Prevention – By following standards and regular reviews, QA helps identify and eliminate errors early.
3. Improved Reliability – Ensures the software performs correctly and consistently under various conditions.
4. Customer Satisfaction – QA helps deliver reliable, user-friendly software that meets client expectations.
5. Continuous Improvement – Promotes regular process evaluation to enhance quality and efficiency over time.

18. Design a Gantt Chart for Software Project Scheduling.
= Introduction
A Gantt Chart is a visual scheduling tool that displays project activities, their duration, and progress over time. It helps in tracking tasks, dependencies, and deadlines in a software project.

Example Gantt Chart for Software Project Scheduling

Task	Start Date	End Date	Duration	Status
1. Requirement Analysis	Jan 1	Jan 10	10 days	✓ Completed
2. System Design	Jan 11	Jan 20	10 days	✓ Completed
3. Coding and Development	Jan 21	Feb 15	25 days	🔄 In Progress
4. Testing	Feb 16	Feb 28	13 days	⏳ Pending
5. Deployment	Mar 1	Mar 5	5 days	⏳ Pending
6. Maintenance	Mar 6	Mar 20	15 days	⏳ Pending

Points

1. Each task is represented as a horizontal bar along a timeline.
2. Overlapping bars indicate parallel tasks.
3. Color or symbols show task status (e.g., completed, in progress).
4. Helps visualize dependencies and deadlines.
5. Useful for monitoring progress and adjusting schedules if needed.

19. Formulate a Risk Management Plan for an E-Learning Platform.
= Introduction
A Risk Management Plan for an E-Learning Platform helps identify, assess, and control potential risks that could affect development, security, or user experience. It ensures project stability and continuity.
Points

1. Risk Identification
   Identify possible risks like server downtime, data breaches, low user engagement, or content delivery delays.
2. Risk Analysis
   Assess each risk based on its probability and potential impact on learning operations and business goals.
3. Risk Mitigation Strategies
   - Use backup servers to handle downtime.
   - Implement strong encryption for user data.
   - Schedule regular content updates and performance reviews.
4. Risk Monitoring
   Continuously track system performance, feedback, and analytics to detect new or recurring risks early.
5. Risk Control and Review
   Evaluate mitigation effectiveness, update strategies regularly, and maintain communication with all stakeholders.

20. Describe the purpose of Regression Testing and when it is applied in a project.
= Introduction
Regression Testing ensures that recent code changes, bug fixes, or feature updates do not negatively affect the existing functionality of a software system. It helps maintain software stability throughout development.
Points

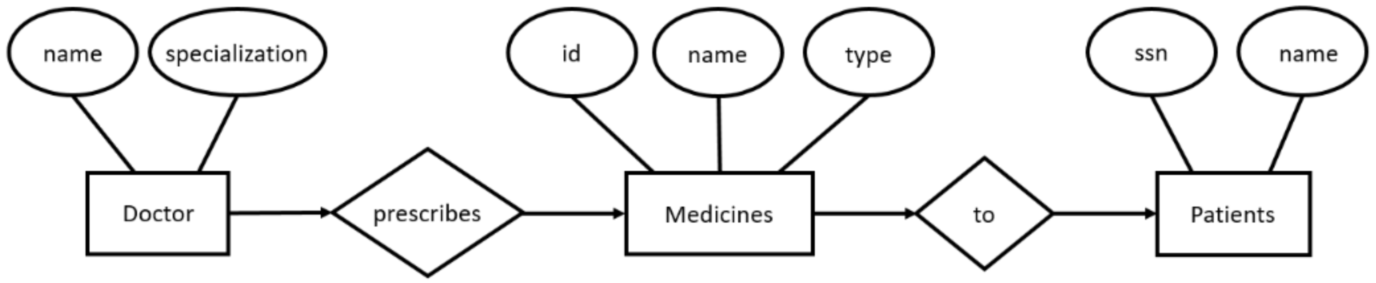
1. Purpose – To verify that modifications or enhancements do not introduce new errors in previously working parts of the software.
2. After Bug Fixes – Applied whenever bugs are fixed to ensure that fixing one issue doesn't break other functions.
3. After Feature Updates – Performed when new features are added to confirm that existing features still work correctly.
4. After Code Integration – Conducted after merging multiple modules or code changes to check system consistency.
5. Before Release – Ensures overall system reliability and performance before deployment or delivery.

21. Develop a Project Scheduling Plan for a Mobile App Project.
= Introduction
A Project Scheduling Plan defines the timeline, sequence, and duration of tasks for developing a Mobile App Project. It ensures efficient use of resources and timely project delivery.
Points

1. Requirement Analysis (Week 1–2)
   Gather user needs, define app objectives, and finalize technical requirements.
2. UI/UX Design (Week 3–4)
   Create wireframes, design prototypes, and finalize the user interface.
3. App Development (Week 5–10)
   Develop front-end and back-end components, integrate APIs, and ensure responsiveness.
4. Testing and Debugging (Week 11–12)
   Conduct functional, performance, and usability testing to fix errors and ensure quality.
5. Deployment and Maintenance (Week 13–14)
   Launch the app on Android and iOS platforms, monitor performance, and provide updates.

22. Use an ER diagram to design a hospital system.

=



23. Demonstrate Logical DFD for an ATM system.
= Introduction
A Logical Data Flow Diagram (DFD) for an ATM system shows how data moves between processes, data stores, and external entities — focusing on *what* happens, not *how* it is implemented.
Logical DFD (Level 0 – Context Level)
Entities & Processes:

1. Customer – interacts with the ATM using a card and PIN.
2. ATM System – processes user requests and manages transactions.
3. Bank Database – stores account and transaction information.

Data Flow Summary:

- Customer → (Card, PIN, Request) → ATM System
- ATM System → (Authentication Request) → Bank Database
- Bank Database → (Validation/Balance Info) → ATM System
- ATM System → (Cash, Receipt, Status Message) → Customer

24. Illustrate the structure of a software requirement specification.
= Introduction
A Software Requirement Specification (SRS) defines the complete description of a system's functionality, performance, and constraints. It acts as a communication bridge between users and developers.
Points

1. Introduction – Includes purpose, scope, definitions, and overall system goals.
2. Overall Description – Describes product perspective, functions, user needs, and constraints.
3. Specific Requirements – Details functional and non-functional requirements like input, output, and performance.
4. External Interface Requirements – Defines how the system interacts with hardware, software, and users.
5. Appendices and References – Contains supporting information, diagrams, and related documents.

25. Apply functional and Non-functional requirements to a banking system.
= Introduction
In a Banking System, requirements are divided into Functional and Non-Functional to ensure both correct operation and good performance.

1. Functional Requirements
   - User login and authentication.
   - Fund transfer between accounts.
   - Account balance inquiry and transaction history.
   - Bill payment and loan management.
   - Admin control for user and transaction monitoring.
2. Non-Functional Requirements
   - Performance: Fast response time during peak hours.
   - Security: Strong encryption and multi-factor authentication.
   - Reliability: 24/7 system availability with data backup.
   - Usability: Easy-to-navigate interface for all users.
   - Scalability: Ability to handle increasing users and transactions.

26. Formulate a complete SRS for an online exam portal.
= SRS for Online Exam Portal
1. Introduction
The Online Exam Portal allows students to take exams online and admins to manage exams, questions, and results securely.
2. Overall Description
- Purpose: To automate exam management and evaluation.
- Users: Admin (manages system) and Student (takes exams).
- Scope: User registration, exam scheduling, result generation.

3. Functional Requirements
1. User login and registration.
2. Exam and question management.
3. Automatic grading and result display.
4. Timer and answer auto-save.
5. Report generation.

4. Non-Functional Requirements
- Performance: Fast and reliable.
- Security: Data encryption and authentication.
- Usability: Simple and responsive UI.
- Scalability: Support many users.

27. Design a data dictionary for a Student management.
=

Entity	Attribute	Type	Description
Student	Student_ID (PK)	Int	Unique ID
	Name	Varchar	Student name
	Email	Varchar	Contact email
Course	Course_ID (PK)	Int	Course ID
	Course_Name	Varchar	Course title
Enrollment	Enroll_ID (PK)	Int	Enrollment record
	Student_ID (FK)	Int	Linked student
	Course_ID (FK)	Int	Linked course

28. Construct a logical DFD that models the core processes of an online ticket booking system.
= Introduction
A Logical DFD for an Online Ticket Booking System shows how data flows between users, system processes, and databases without focusing on implementation details.
Points
1. Customer – Registers, logs in, and requests ticket booking.
2. Booking Process – Verifies seat availability and processes payments.
3. Payment Gateway – Handles secure payment transactions.
4. Database – Stores user details, tickets, and payment records.
5. System Output – Generates e-ticket and confirmation message for the customer.

Data Flow Summary:
- Customer → (Booking Request) → System
- System → (Payment Info) → Payment Gateway
- Payment Gateway → (Confirmation) → System
- System → (E-Ticket) → Customer
- Database ↔ System for storing and retrieving data