

Software Engineering and Testing (315332)

Model Question Paper — Set A · Model Answers

Q.1

a) Define software engineering.

2 M

Software engineering is the application of a **systematic, disciplined, quantifiable approach** to the development, operation and maintenance of software; that is, the application of engineering to software. (*IEEE*)

OR: Software engineering is the establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines. (*Fritz Bauer*)

(Correct definition 2 M)

b) Need of SRS.

2 M

SRS is a document that completely describes what the proposed software should do, without describing how it will do it.

1. **Agreement between customer and developer:** it records exactly what the software must do, so both sides have the same understanding.
2. **Reduces development effort and rework:** careful review of the SRS finds omissions and misunderstandings early, before design and coding.
3. **Basis for estimating cost and schedule:** size, effort and time are estimated from the documented requirements.
4. **Baseline for verification and validation:** test plans and test cases are written from the SRS; the product is accepted against it.

(Any two needs, 1 M each)

c) Types of software risks.

2 M

1. **Project risks** — threaten the project plan: budget, schedule, personnel, resources, customer and requirement problems.
2. **Technical risks** — threaten quality and timeliness: design, implementation, interface, verification and maintenance problems.
3. **Business risks** — threaten the viability of the software: market risk, strategic risk, sales risk, management risk, budget risk.
4. **Known risks** — uncovered after careful evaluation of the project plan (e.g. unrealistic delivery date).
5. **Predictable risks** — extrapolated from past project experience (e.g. staff turnover).
6. **Unpredictable risks** — can occur, but are extremely difficult to identify in advance.

(Any four, ½ M each)

d) Error and failure.

2 M

- **Error (mistake):** a human action that produces an incorrect result, e.g. a programmer types < instead of <=.
- **Failure:** the inability of the system or component to perform its required function within the specified requirements; it is observed when a fault is executed.

Chain: an *error* by a person introduces a *fault* (defect) in the code, which on execution may cause a *failure*.

(Each definition 1 M)

e) Scrum and its roles.

2 M

Scrum is an agile software development method (Jeff Sutherland and Ken Schwaber) in which work is done in short, time-boxed iterations called **sprints** (typically 2–4 weeks / 30 days). Work is taken from a prioritised **product backlog**, the team meets daily in a short **scrum meeting**, and each sprint delivers a potentially shippable increment.

Scrum roles: (1) **Product owner** — owns and prioritises the product backlog; (2) **Scrum master** — facilitates the process, conducts the daily scrum and removes obstacles; (3) **Development team** — self-organising team that builds the increment.

(Definition 1 M, two roles 1 M)

f) Benefits of a test plan.

2 M

1. Gives a clear roadmap (direction) for the whole testing activity.
2. Defines the scope — features to be tested and not to be tested.
3. Helps estimate effort, resources, schedule and cost of testing.
4. Defines roles and responsibilities of the test team.
5. Defines entry / exit (pass/fail) and suspension criteria.
6. Identifies testing risks and contingency plans early.

(Any four, ½ M each)

g) Unit testing.

2 M

Unit testing is the testing of the smallest testable unit of software — an individual module, function, procedure or class — in isolation, to verify that it works as per its detailed design. It is usually done by the developer using white-box techniques; *drivers* and *stubs* are used to replace the calling and called modules.

Tested in a unit: module interface, local data structures, boundary conditions, independent paths and error-handling paths.

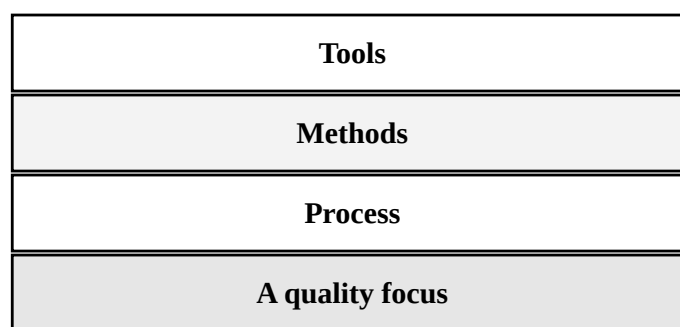
(Definition 2 M)

Q.2

a) Software engineering — a layered technology.

4 M

Software engineering is a **layered technology**: any engineering approach (including software engineering) must rest on an organisational commitment to quality.



Software engineering — a layered technology

1. **A quality focus:** the bedrock. Any engineering approach must rest on an organisational commitment to quality; total quality management, Six Sigma and similar philosophies foster continuous process improvement.
2. **Process:** the foundation layer and the glue that holds the technology layers together. It defines the framework of key process areas, the context in which methods are applied, work products are produced, milestones are established, quality is ensured and change is managed.
3. **Methods:** provide the technical “how-to’s” for building software — communication, requirements analysis, design modelling, program construction, testing and support.
4. **Tools:** provide automated or semi-automated support for the process and methods (CASE tools); when tools are integrated, information created by one tool is used by another.

(Diagram 2 M, explanation 2 M)

b) Extreme Programming (XP).

4 M

Extreme Programming (XP), proposed by Kent Beck, is the most widely used agile process. It uses an object-oriented approach and follows four framework activities — planning, design, coding and testing.

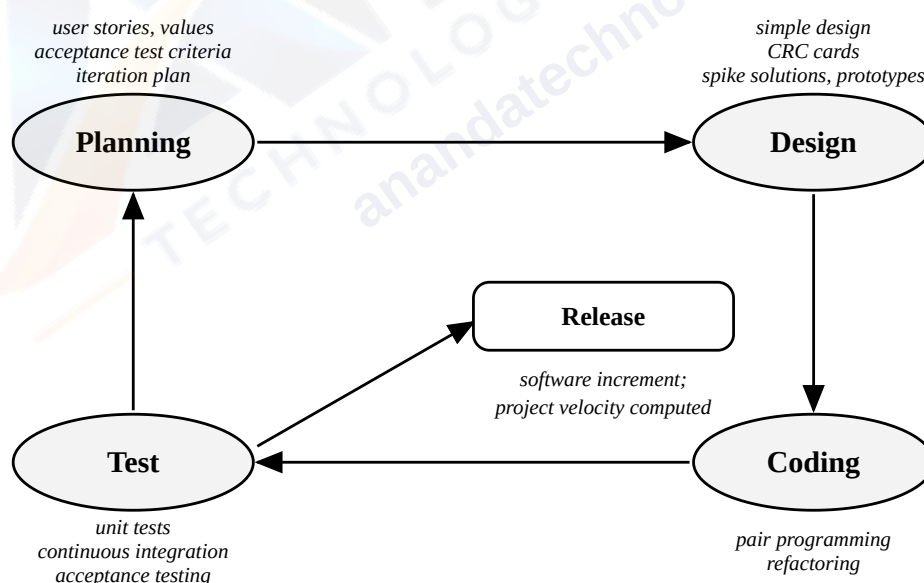


Fig.: Extreme Programming (XP) process

1. **Planning:** begins with *user stories* written by the customer on index cards, each with a value (priority). The team estimates the cost (in weeks) of each story; stories are grouped into a release. After the first release, *project velocity* (stories implemented per release) is computed and used to plan later releases.
2. **Design:** follows the KIS (keep it simple) principle. *CRC cards* (class–responsibility–collaborator) identify classes; for a difficult design problem a *spike solution* (operational prototype) is built. *Refactoring* improves internal design without changing behaviour.
3. **Coding:** unit tests are written *before* coding. **Pair programming** — two people work together at one workstation (real-time problem solving and quality assurance). Code is integrated continuously.
4. **Testing:** unit tests are automated and run daily (regression); *acceptance tests* (customer tests) are specified by the customer and check overall features visible to the customer.

(Diagram 2 M, four activities 2 M)

c) Software Testing Life Cycle (STLC).

4 M

STLC is the sequence of specific activities carried out during the testing process to ensure that quality goals are met. Each phase has defined entry and exit criteria.

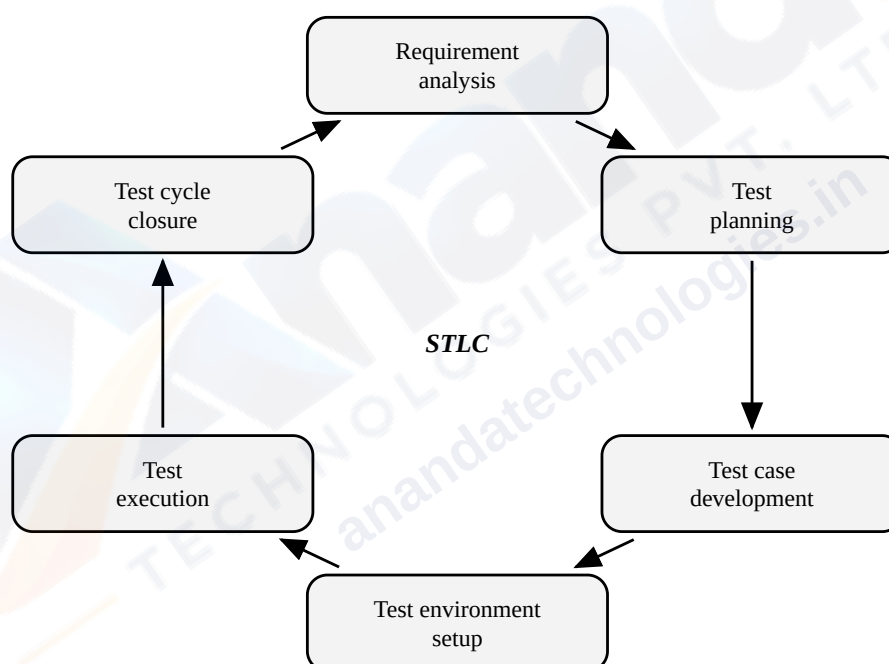


Fig.: Software Testing Life Cycle (STLC)

1. **Requirement analysis:** Testers study the SRS to identify testable requirements; a Requirement Traceability Matrix (RTM) is prepared; automation feasibility is checked.
2. **Test planning:** Test manager prepares the test plan / strategy — scope, effort and cost estimation, resources, tools, schedule, risks, entry/exit criteria.
3. **Test case development:** Test cases, test scripts and test data are written and reviewed; RTM is updated.

4. **Test environment setup:** Hardware, software, network and test data are prepared; a smoke test checks that the build is ready for testing.
5. **Test execution:** Test cases are executed; actual results are compared with expected; failures are logged as defects, tracked and retested after fixing.
6. **Test cycle closure:** Exit criteria are evaluated; test summary report and metrics (coverage, defects, cost) are prepared; lessons learned are recorded.

(Diagram / list 1 M, explanation 3 M)

d) 4 P's of management spectrum.

4 M

Effective software project management focuses on the **four P's**: People, Product, Process and Project.

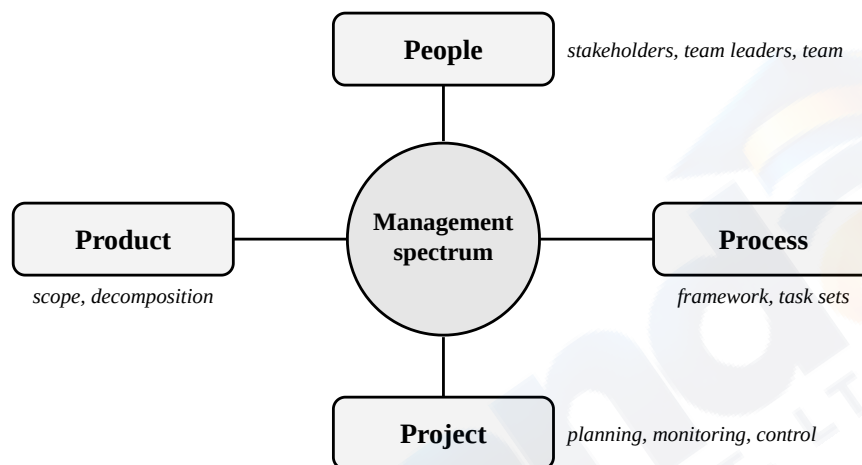


Fig.: The 4 P's of the management spectrum

- 1. People:** The most important element. Stakeholders are senior managers, project (technical) managers, practitioners, customers and end users. Team leaders must motivate, organise and innovate; the team structure (democratic, controlled, agile team) and coordination/communication issues are decided.
- 2. Product:** The software to be built. Before planning, the *software scope* (context, information objectives, function and performance) is established and the problem is *decomposed* into smaller functions so that estimation is possible.
- 3. Process:** The software process provides the framework (communication, planning, modeling, construction, deployment) from which a comprehensive plan is made. A suitable process model and task set is chosen, and umbrella activities (SQA, configuration management, measurement) are added.
- 4. Project:** A planned and controlled software project is the only known way to manage complexity. The project manager plans, monitors and controls work, avoids the common signs of failure and uses the W5HH principle (Why, What, When, Who, Where, How, How much).

(Each P 1 M)

Q.3

a) Use case diagram — Online Food Ordering System.

4 M

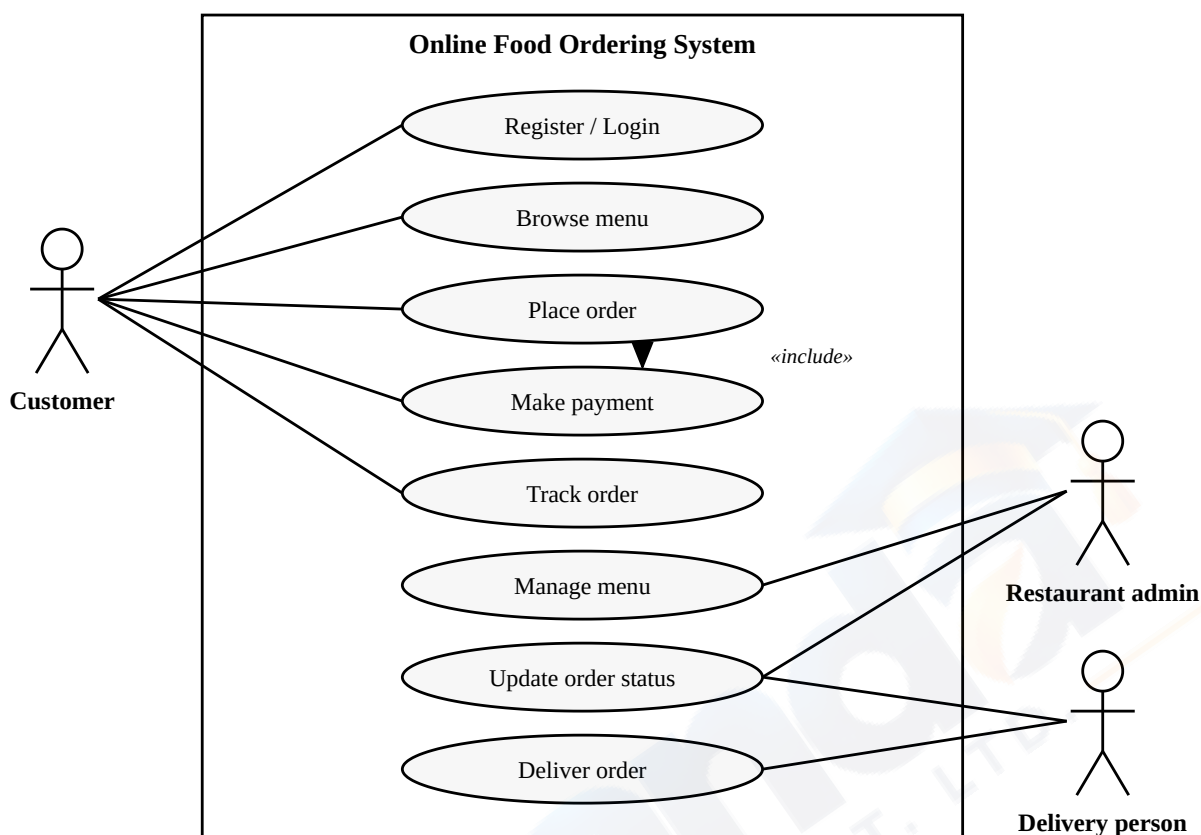


Fig.: Use case diagram — Online Food Ordering System

- **Actors:** Customer (primary), Restaurant admin and Delivery person (secondary).
- **Use cases** are shown as ellipses inside the system boundary; lines show which actor participates in which use case.
- **Place order «include» Make payment** — payment is always done as part of placing an order.

(Actors 1 M, use cases 2 M, relationships / boundary 1 M)

b) Time-line chart — Hospital Management System (10 days).

4 M

A **time-line chart** (Gantt-type chart) shows each work task as a horizontal bar against a calendar. Here the Hospital Management System is planned for **10 working days** = 2 weeks × 5 days a week, with SDLC phases as the work tasks.

Work task (SDLC phase)	Start day	End day	Days	Week / day
Requirement gathering	1	2	2	W1 D1 – W1 D2
Planning	3	3	1	W1 D3 – W1 D3
Analysis & design	4	5	2	W1 D4 – W1 D5
Coding	6	8	3	W2 D1 – W2 D3
Testing	9	9	1	W2 D4 – W2 D4
Deployment	10	10	1	W2 D5 – W2 D5

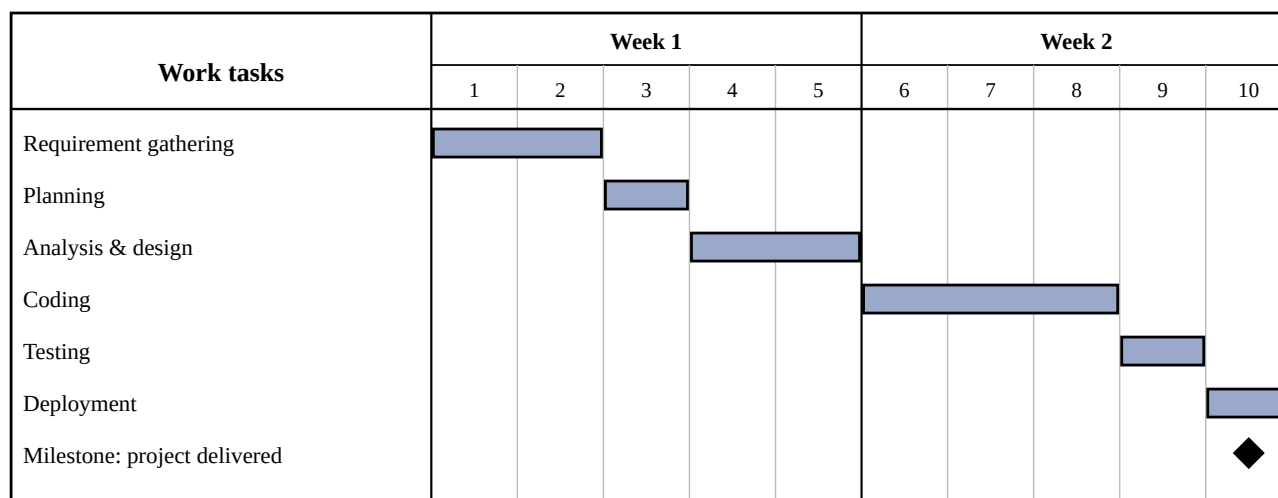


Fig.: Time-line chart — Hospital Management System (10 working days, ◆ = milestone)

(Table / allocation 1 M, chart 3 M)

c) Test cases — login page of online banking.

4 M

Test case: a set of input values, execution preconditions, expected results and execution post-conditions developed for a particular objective or test condition, such as to exercise a particular program path or to verify compliance with a specific requirement (IEEE 829).

TC ID	Test scenario	Test steps / test data	Expected result
TC_01	Login with valid user ID and valid password	User ID = ravi2025, Password = Ravi@123; click Login	Login successful; account dashboard is displayed
TC_02	Login with valid user ID and wrong password	User ID = ravi2025, Password = abc123; click Login	Message “Invalid user ID or password”; user stays on login page
TC_03	Login with unregistered user ID	User ID = xyz999, any password; click Login	Message “Invalid user ID or password”
TC_04	Login with blank fields	Leave user ID and password empty; click Login	Message “Please enter user ID and password”
TC_05	Account lock after repeated wrong attempts	Enter wrong password 3 times	Account is temporarily locked; OTP / reset link option shown
TC_06	Password field masking	Type password in the password field	Characters are shown as dots / asterisks

Actual result and status (Pass/Fail) columns are filled in during test execution.

(Any four valid test cases, 1 M each)

d) Defect life cycle.

4 M

The **defect life cycle** (bug life cycle) is the set of states a defect goes through from the time it is found until it is closed.

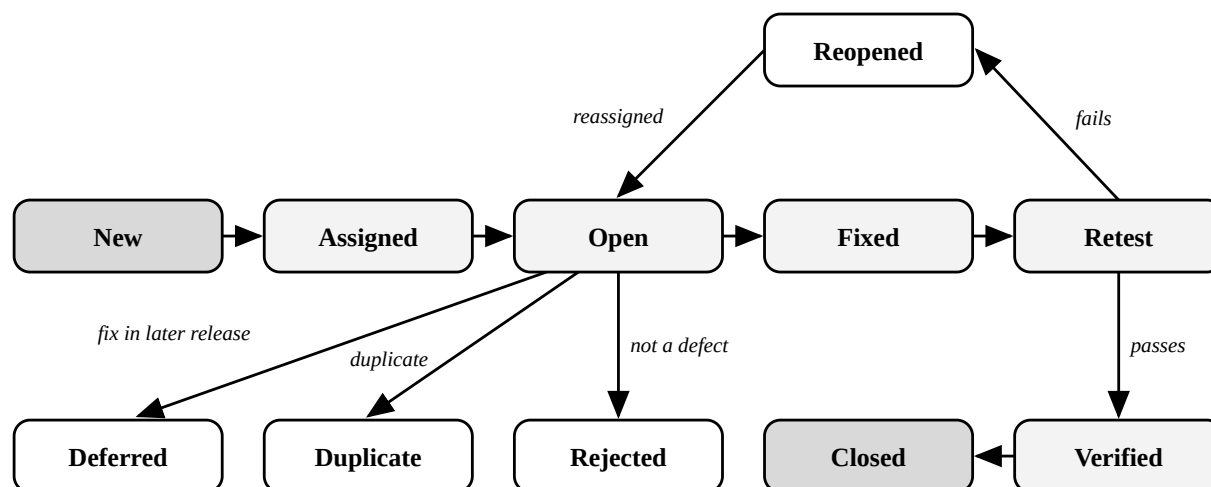


Fig.: Defect (bug) life cycle

1. **New:** defect is logged for the first time by the tester; not yet approved.
2. **Assigned:** test lead / project lead approves it and assigns it to a developer.
3. **Open:** developer starts analysing and working on the defect. If it is not genuine it may be marked *Rejected*, *Duplicate* or *Deferred* (fix in a later release).
4. **Fixed:** developer makes the code change, verifies it and marks the defect fixed.
5. **Retest (pending retest):** the fixed build is given to the tester, who retests the defect.
6. **Reopened:** if the defect still exists after retesting, the tester reopens it and it goes through the cycle again.
7. **Verified:** tester confirms the defect is no longer reproducible.
8. **Closed:** defect is fixed, retested and approved; the life cycle ends.

(Diagram 2 M, working of states 2 M)

Q.4

a) Prescriptive vs agile process model.

4 M

Point	Prescriptive process model	Agile process model
Approach	Plan-driven; activities follow a defined sequence	Iterative and incremental; adapts to change
Requirements	Fixed and fully defined at the start	Evolve; changes welcome even late
Customer involvement	Mainly at the start (requirements) and end (delivery)	Continuous, customer is part of the team
Delivery	Complete product delivered at the end	Working software delivered frequently (2–4 week increments)
Documentation	Heavy, formal documentation	Light; working software valued more
Team	Large, role-based, managed by plan	Small, self-organising, collaborative
Suitable for	Well-understood, stable, large projects	Projects with changing / unclear requirements
Examples	Waterfall, incremental, RAD, prototyping, spiral	XP, Scrum, ASD, DSDM, Crystal

(Any four points, 1 M each)

b) Six Sigma and DMAIC.

4 M

Six Sigma is a widely used strategy for statistical quality assurance, popularised by Motorola in the 1980s. It uses data and statistical analysis to measure and improve a company's operational performance by identifying and eliminating defects. Six Sigma quality means not more than **3.4 defects per million occurrences**.

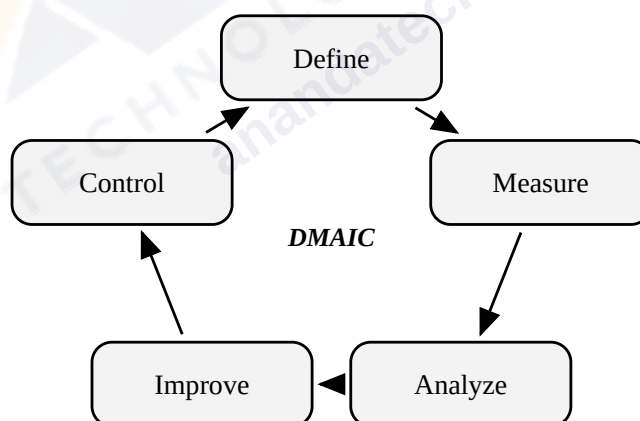


Fig.: DMAIC cycle (Six Sigma — improving an existing process)

DMAIC (for improving an existing process):

- 1. Define** — customer requirements, deliverables and project goals through well-defined methods of customer communication.

2. **Measure** — the existing process and its output to determine current quality performance (collect defect metrics).
3. **Analyze** — defect metrics and determine the vital few causes.
4. **Improve** — the process by eliminating the root causes of defects.
5. **Control** — the process to ensure that future work does not reintroduce the causes of defects.

(Six Sigma 1 M, DMAIC 3 M)

c) Verification vs validation.

4 M

Point	Verification	Validation
Question answered	“Are we building the product right?”	“Are we building the right product?”
Meaning	Checks that the work product of a phase meets the conditions set at the start of that phase (conforms to specification)	Checks that the final software meets the customer’s actual needs and requirements
Type of testing	Static — code is not executed	Dynamic — code is executed
Methods	Reviews, walkthroughs, inspections, desk checking	Black-box and white-box testing, unit to acceptance testing
Timing	Done before validation, in every phase	Done after verification, on the executable product
Done by	QA team / developers / reviewers	Testing team, with users in acceptance testing
Finds	Defects early (cause), cheap to fix	Failures (effect) when the program runs

(Any four points, 1 M each)

d) Flow graph and cyclomatic complexity.

4 M

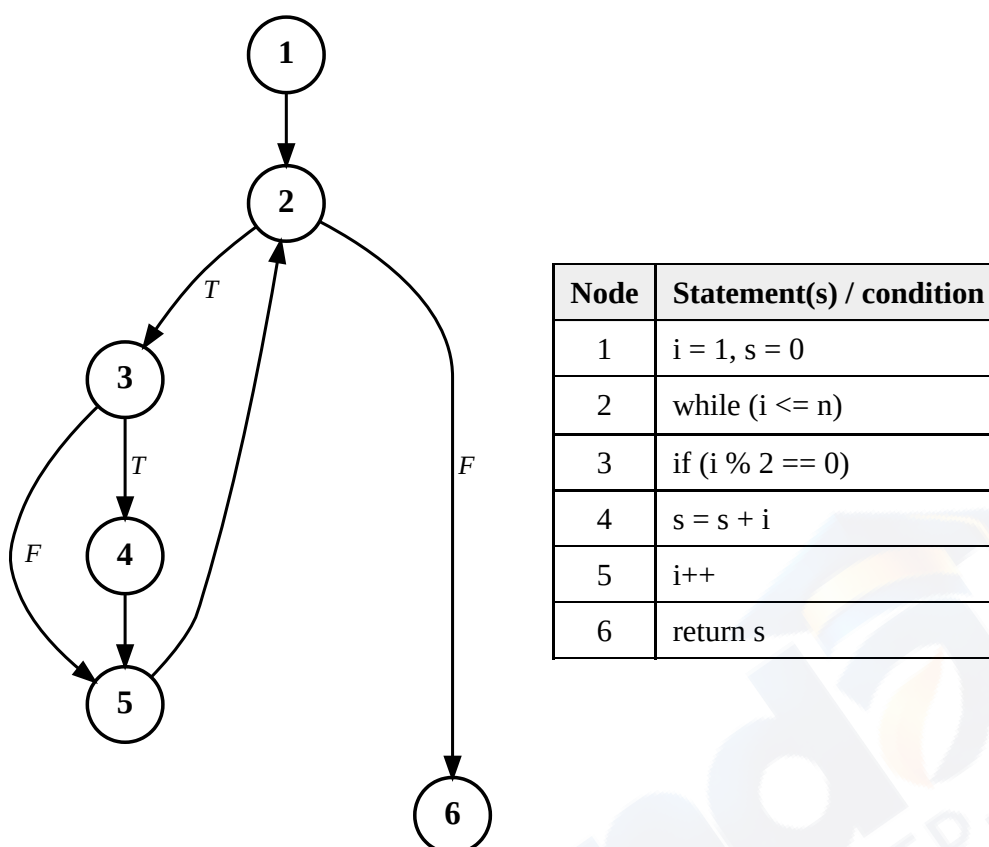


Fig.: Flow graph of sum_even()

Cyclomatic complexity V(G) (McCabe) gives the number of **independent paths** in the basis set, i.e. the minimum number of test cases that execute every statement and every branch at least once. From the flow graph: number of nodes N = 6, number of edges E = 7, predicate (decision) nodes P = 2 (nodes 2, 3).

i) $V(G) = E - N + 2 = 7 - 6 + 2 = 3$

ii) $V(G) = P + 1 = 2 + 1 = 3$

iii) $V(G) = \text{number of regions of the flow graph (including the outer region)} = 3$

Basis set of independent paths:

1. 1 → 2 → 6
2. 1 → 2 → 3 → 5 → 2 → 6
3. 1 → 2 → 3 → 4 → 5 → 2 → 6

Test cases for the basis paths (outputs obtained by running the function):

Basis path	Input n	Output (sum)
Path 1	0	0
Path 2	1	0
Path 3	2	2

(Flow graph 2 M, V(G) calculation 2 M)

e) Test summary report.

4 M

A **test summary report** is prepared at the end of a test cycle / project. It summarises the testing activities and results and helps management decide whether the product can be released. Contents (IEEE 829):

1. **Test summary report identifier:** unique ID of the report.
2. **Summary:** items tested with versions, test environment, references to test plan, test cases and incident reports.
3. **Variances:** differences from the test plan / test design and the reasons (e.g. features not tested, schedule changes).
4. **Comprehensiveness assessment:** how far testing covered the planned features; features or combinations not sufficiently tested.
5. **Summary of results:** number of test cases run, passed, failed; defects found, fixed and still open, with severity.
6. **Evaluation:** overall assessment of each test item and its quality; risk of release.
7. **Summary of activities:** major testing activities, effort, time and resources used versus planned.
8. **Recommendation and approvals:** whether the product is ready for release; names and signatures of people who approve the report.

(Any four contents with explanation, 1 M each)

Q.5

a) Requirements — College Admission System.

6 M

i) Student registration module

- Student shall register with name, date of birth, category, mobile number, e-mail and a password.
- System shall verify mobile number / e-mail using OTP.
- Student shall fill the application form: personal details, SSC/HSC marks, preferred courses.
- Student shall upload documents (photo, mark sheets, caste / domicile certificate) in PDF/JPG within a size limit.
- Student shall view and edit the application until the last date and download an acknowledgement.

ii) Fee payment module

- System shall calculate the fee according to course and category (e.g. open / reserved / scholarship).
- Student shall pay online through debit card, credit card, UPI or net banking.
- System shall generate a receipt with transaction ID and send it by e-mail / SMS.
- Accountant shall view daily fee collection reports and record offline (cash / DD) payments.
- Failed or pending transactions shall be reconciled and refunds tracked.

iii) Merit list module

- System shall compute merit marks (percentage of qualifying examination) for every confirmed application.

- System shall sort applicants by merit marks; ties broken by a defined rule (e.g. higher marks in Mathematics, then date of birth).
- Admin shall generate course-wise and category-wise merit lists as per seat intake.
- Merit list shall be published on the portal and downloadable as PDF; students can view their rank.
- Admin shall open a grievance window and publish the final merit list after corrections.

Non-functional requirements (common to all modules):

- Security — passwords stored encrypted; role-based access (student, admin, accountant).
- Performance — portal should handle peak load near the last date with response time under 3 seconds.
- Availability — 24 × 7 during the admission period; daily database backup.

(Each module 2 M (any four requirements))

b) Basic COCOMO — 300 KLOC, semi-detached.

6 M

COCOMO (Constructive Cost Model) was proposed by Barry Boehm (1981). It is an empirical (heuristic) model that estimates effort, development time and staff from the size of the software in KLOC. It has three levels — **Basic**, **Intermediate** (adds 15 cost drivers) and **Detailed** (cost drivers per phase).

Project modes:

- **Organic:** small, simple project (about 2–50 KLOC), small experienced team, familiar environment, e.g. payroll, inventory.
- **Semi-detached:** medium size (about 50–300 KLOC), mixed team experience, e.g. compiler, DBMS.
- **Embedded:** large/complex (over 300 KLOC) with tight hardware, software and operational constraints, e.g. air-traffic control, ATM network.

Basic COCOMO equations: Effort $E = a \times (\text{KLOC})^b$ person-months · Development time $D = c \times (E)^d$ months · Staff = E / D persons

Mode	a	b	c	d
Organic	2.4	1.05	2.5	0.38
Semi-detached	3.0	1.12	2.5	0.35
Embedded	3.6	1.2	2.5	0.32

Given: Size = 300 KLOC = 300,000 LOC

Semi-detached mode ($a = 3.0$, $b = 1.12$, $c = 2.5$, $d = 0.35$):

i) Effort $E = a \times (\text{KLOC})^b = 3.0 \times (300)^{1.12} = 3.0 \times 594.81 = \mathbf{1784.42 \text{ person-months}}$

ii) Development time $D = c \times (E)^d = 2.5 \times (1784.42)^{0.35} = \mathbf{34.35 \text{ months}}$

iii) Average staff size $SS = E / D = 1784.42 / 34.35 = \mathbf{51.94 \approx 52 \text{ persons}}$

iv) Productivity $P = \text{KLOC} / E = 300 / 1784.42 = \mathbf{0.1681 \text{ KLOC/PM}}$ (= 168.1 LOC per person-month)

Mode	Effort (PM)	Time (months)	Staff	Productivity (LOC/PM)
Semi-detached	1784.42	34.35	51.9	168.1

(Formulae 2 M, effort 1 M, time 1 M, staff 1 M, productivity 1 M)

c) Software process framework.

6 M

A **process framework** establishes the foundation for a complete software engineering process by identifying a small number of **framework activities** applicable to all software projects, together with a set of **umbrella activities** applicable across the entire process. Each framework activity is populated by a set of software engineering actions; each action has a *task set* (work tasks, work products, quality assurance points and milestones).

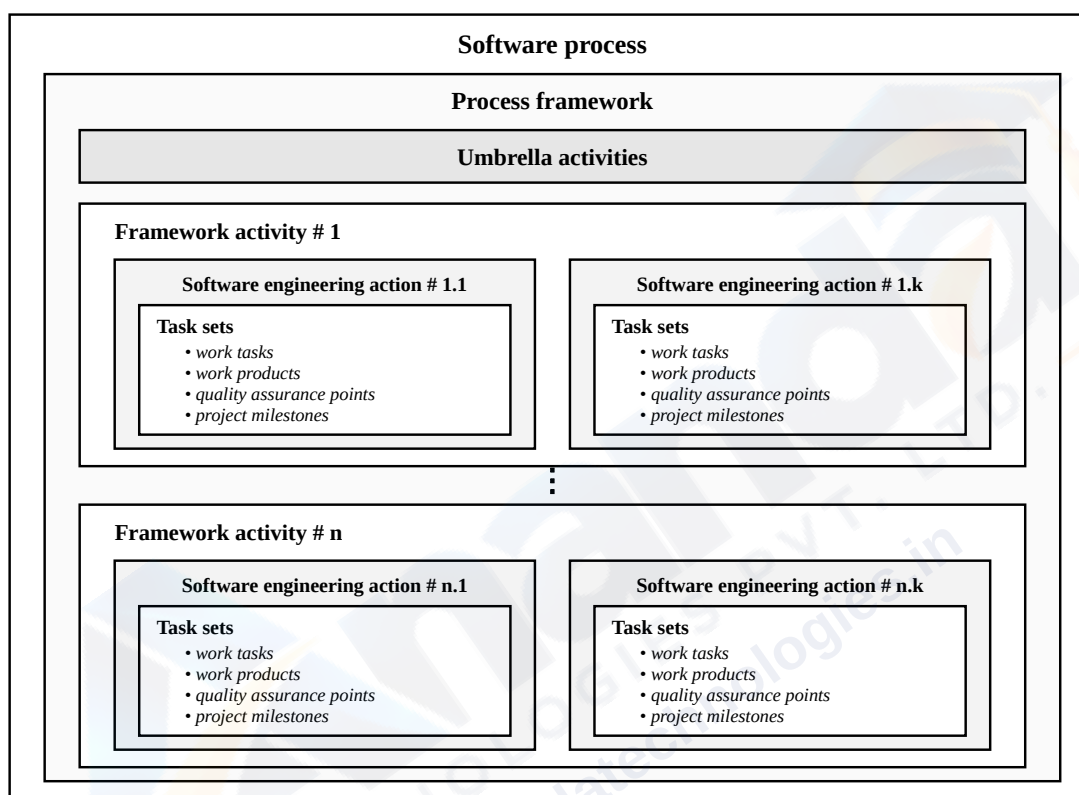


Fig.: A software process framework

Generic framework activities:

1. **Communication:** communicate and collaborate with the customer and stakeholders to understand objectives and gather requirements.
2. **Planning:** create a “map” (software project plan) — technical tasks, risks, resources, work products and schedule.
3. **Modeling:** create models (analysis and design) to understand requirements and the design that will achieve them.
4. **Construction:** code generation (manual or automated) and testing to uncover errors in the code.
5. **Deployment:** deliver the software (complete or increment) to the customer, who evaluates it and gives feedback.

Umbrella activities:

1. **Software project tracking and control:** assess progress against the plan and take action to maintain the schedule.
2. **Risk management:** assess risks that may affect the outcome or quality of the project.
3. **Software quality assurance:** define and conduct activities to ensure software quality.
4. **Technical reviews:** assess work products to uncover and remove errors before they move to the next activity.
5. **Measurement:** define and collect process, project and product measures.
6. **Software configuration management:** manage the effects of change throughout the process.
7. **Reusability management:** define criteria for reuse of work products and establish mechanisms to achieve it.
8. **Work product preparation and production:** create models, documents, logs, forms and lists.

(Diagram 2 M, framework activities 2 M, umbrella activities 2 M)

Q.6

a) Size estimation techniques; function point calculation.

6 M

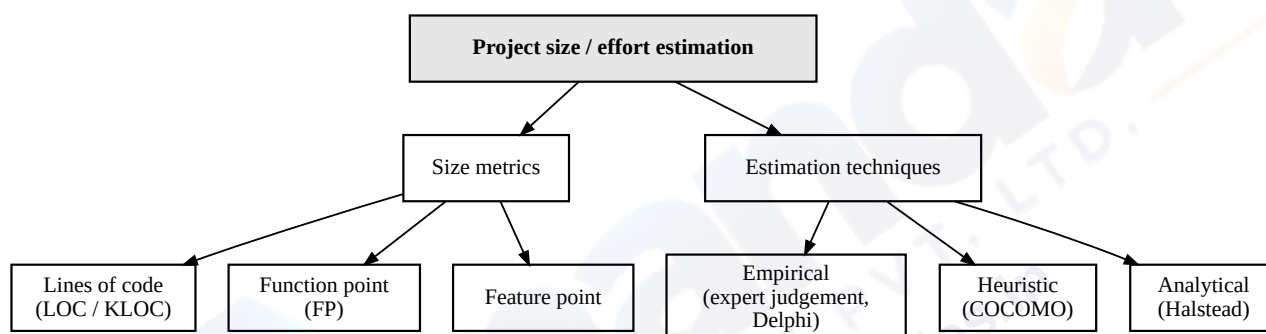


Fig.: Project size estimation techniques

1. **Lines of Code (LOC):** size is measured as the number of delivered source lines (excluding comments and blank lines), usually in KLOC. Simple to count, but language-dependent and known only after coding.
2. **Function Point (FP):** size is measured from the functionality delivered to the user (inputs, outputs, inquiries, files, interfaces); language-independent and can be computed from the SRS.
3. **Feature point:** an extension of FP that also counts algorithms — suitable for real-time and embedded software.

Function point computation (weights for average complexity: EI 4, EO 5, EQ 4, ILF 10, EIF 7):

Information domain value	Count	Weight (average)	Count × weight
External inputs (EI)	24	4	96
External outputs (EO)	46	5	230
External inquiries (EQ)	20	4	80
Internal logical files (ILF)	6	10	60
External interface files (EIF)	4	7	28
Count total (UFP)			494

$$\Sigma F_i = 4 + 3 + 5 + 3 + 2 + 4 + 4 + 3 + 5 + 2 + 3 + 1 + 2 + 4 = 45$$

$$\text{CAF (value adjustment)} = 0.65 + 0.01 \times 45 = \mathbf{1.10}$$

$$\text{FP} = \text{UFP} \times \text{CAF} = 494 \times 1.10 = \mathbf{543.40}$$

Answer: FP ≈ 543

(Techniques 2 M, UFP 2 M, CAF and FP 2 M)

b) Levels of testing.

6 M

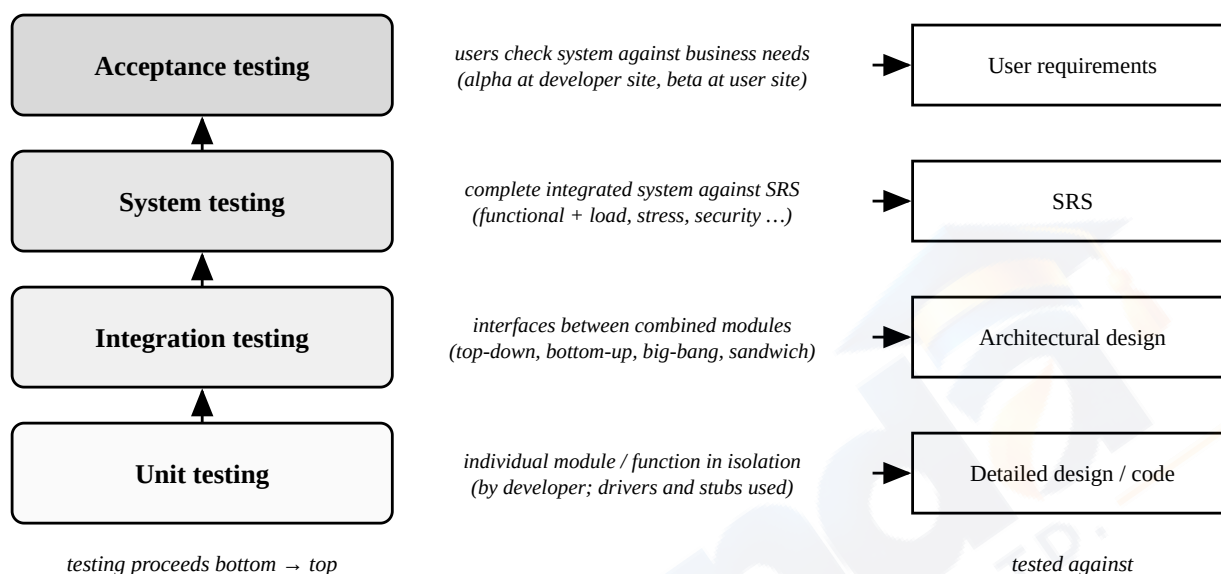


Fig.: Levels of testing

- Unit testing:** tests each module / function separately against its detailed design; done by developers with white-box techniques. Interfaces, local data structures, boundary conditions, independent paths and error-handling paths are tested. Drivers and stubs are used.
- Integration testing:** modules are combined and tested to uncover errors in their interfaces. Approaches: *top-down* (main module first, stubs for lower modules), *bottom-up* (lowest modules first, drivers used), *big-bang* and *sandwich*; regression and smoke testing are used.
- System testing:** the complete integrated system is tested in an environment close to production against the SRS — functional and non-functional tests such as recovery testing, security testing, stress testing, performance (load) testing. It is black-box testing done by an independent test team.
- Acceptance testing:** done by the customer / end users to decide whether to accept the system. *Alpha testing* — at the developer's site by users in a controlled environment; *beta testing* — at the users' sites without the developer present; user acceptance testing (UAT) against business requirements.

(Diagram 2 M, each level 1 M)

c) Defect report — online shopping cart / checkout.

6 M

A **defect report** documents a defect so that developers can reproduce, fix and track it. Using a standard defect template:

Field	Details
Defect ID	OSW-CHK-0147
Project / module	ShopEasy online shopping website / Cart and Checkout
Build / version	v2.3.1 (test build 18)
Defect title	Cart total is not updated when item quantity is changed
Description	When the quantity of an item in the cart is increased, the line total changes but the cart sub-total and payable amount on the checkout page remain at the old value.
Steps to reproduce	1. Log in as a registered customer. 2. Add “Cotton T-shirt (₹ 499)” to the cart. 3. Open the cart and change quantity from 1 to 3. 4. Click “Proceed to checkout”.
Expected result	Sub-total and payable amount = $3 \times ₹ 499 = ₹ 1,497$
Actual result	Sub-total and payable amount shown as ₹ 499 (old value for quantity 1)
Severity	High — customer would be charged a wrong amount
Priority	P1 — fix before release
Status	New
Environment	Windows 11, Chrome 129; Android 14, Chrome mobile
Reported by / date	Tester: A. Patil, 12-03-2026
Assigned to	Developer: S. Kulkarni (checkout team)
Attachment	Screenshot cart_total.png, browser console log

Severity tells how much the defect affects the system; **priority** tells how soon it must be fixed.

(Template fields 3 M, correct filled details 3 M)

Note: This is a **model question paper prepared by Ananda Technologies** only for practice and exam preparation, based on the pattern of previous MSBTE question papers and the K-scheme syllabus. It is **not an official MSBTE paper**, and it does **not** mean that these questions will come in the exam. Answers follow standard textbook points; marks shown are indicative. Free for students — anandatechnologies.in