

Software Engineering and Testing (315332)

Model Question Paper — Set C · Model Answers

Q.1

a) Software and its characteristics.

2 M

Software is (1) **instructions** (computer programs) that when executed provide desired features, function and performance; (2) **data structures** that enable the programs to manipulate information; and (3) **documentation** that describes the operation and use of the programs.

- Software is **developed or engineered**; it is not manufactured in the classical sense.
- Software **does not wear out**, but it **deteriorates** due to changes.
- Most software is still **custom built**, although reusable components are increasing.

(Definition 1 M, two characteristics 1 M)

b) Reasons for SRS.

2 M

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 four, ½ M each)

c) Classification of risks.

2 M

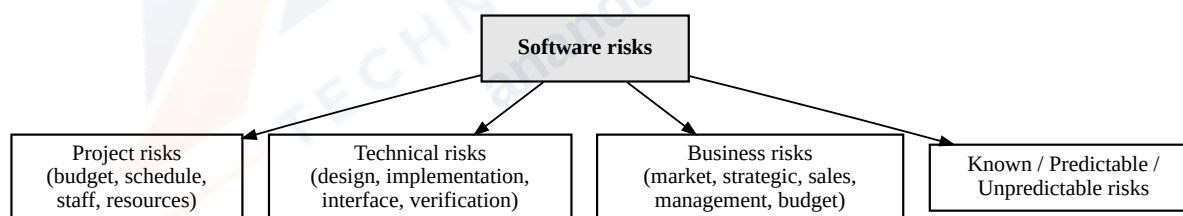


Fig.: Classification of software risks

- **Project risks** — threaten the project plan.
- **Technical risks** — threaten quality and timeliness of the software.
- **Business risks** — threaten the viability of the software (market, strategic, sales, management, budget).
- By how easily they are found: **known**, **predictable** and **unpredictable** risks.

(Classification 2 M)

d) Bug and failure.

2 M

- **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.
- **Bug:** common (informal) name for a fault / defect in a program that makes it behave unexpectedly.

A bug in the code causes a failure only when that part of the code is executed under particular conditions.

(Each definition 1 M)

e) Agile process and models.

2 M

Agility is the ability to respond effectively to change. An **agile process** is an iterative and incremental approach that delivers working software in short increments, encourages continuous customer collaboration, uses small self-organising teams and welcomes changing requirements.

Agile models: Extreme Programming (XP), Scrum, Adaptive Software Development (ASD), DSDM, Crystal, Feature Driven Development (FDD).

(Definition 1 M, two models 1 M)

f) Test case and its fields.

2 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).

Fields: Test case ID, Test scenario / objective, Pre-conditions, Test steps, Test data (input), Expected result, Actual result, Status (Pass / Fail).

(Definition 1 M, any four fields 1 M)

g) Acceptance testing.

2 M

Acceptance testing is formal testing done by the customer / end users to determine whether the system satisfies the acceptance criteria and business needs, so that the customer can decide whether to accept the system.

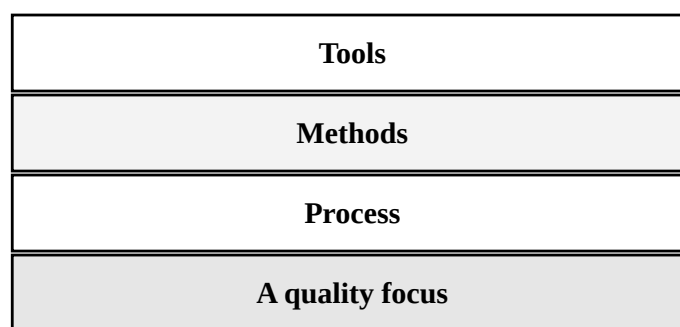
- **Alpha testing:** done at the developer's site by users, with the developer watching and recording problems (controlled environment).
- **Beta testing:** done at one or more users' sites without the developer; users report problems found in a real environment.

(Definition 1 M, two types 1 M)

Q.2**a) Layers of software engineering.**

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, four layers 2 M)

b) Scrum process.

4 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.

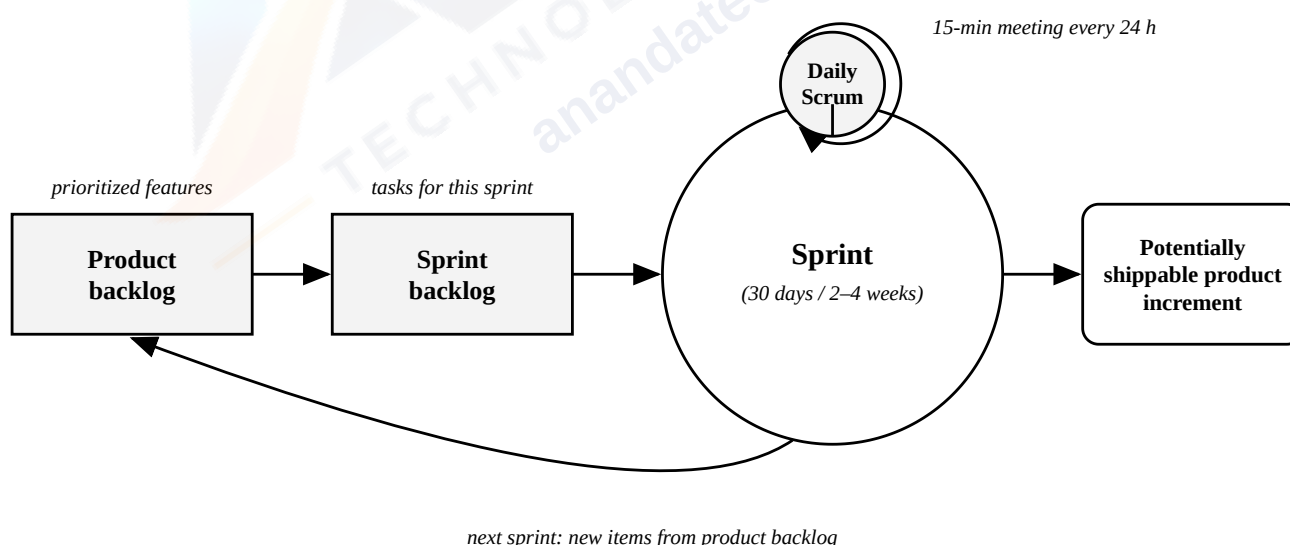


Fig.: Scrum process flow

1. **Product backlog:** prioritised list of requirements / features that provide business value; items can be added at any time by the product owner.

2. **Sprint planning & sprint backlog:** the team selects backlog items that can be finished in one sprint; changes are not introduced during the sprint (stable work environment).
3. **Sprint:** work units required to achieve a requirement in a fixed time-box (usually 30 days).
4. **Daily scrum meeting:** 15-minute stand-up meeting; each member answers — What did you do since the last meeting? What obstacles are you facing? What will you do by the next meeting? The *Scrum master* leads it and removes obstacles.
5. **Demo / sprint review:** the increment is demonstrated to the customer for evaluation; a *sprint retrospective* improves the process for the next sprint.

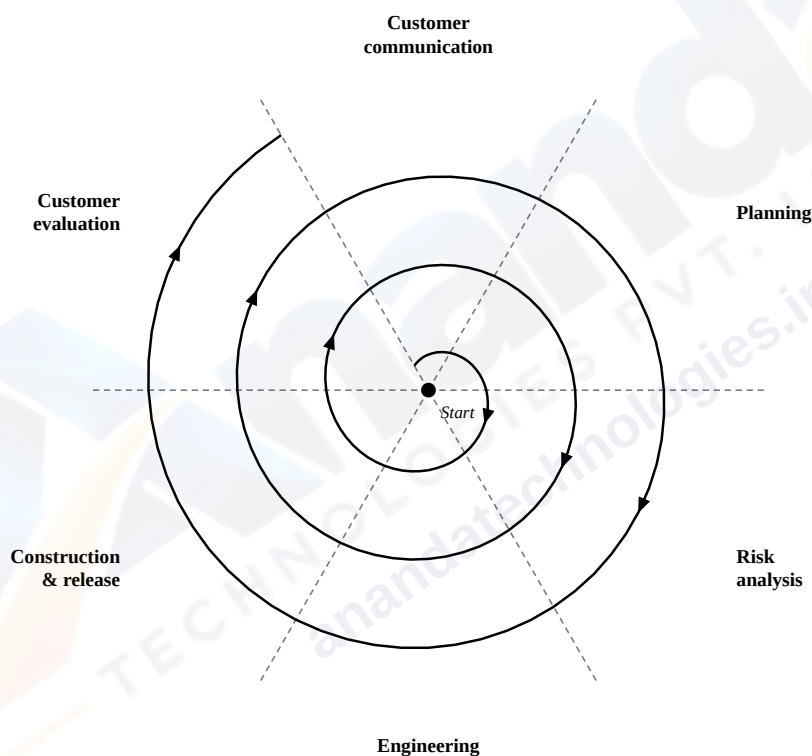
Roles: Product owner, Scrum master, Development team.

(Diagram 2 M, explanation 2 M)

c) Spiral model.

4 M

The **spiral model** (Barry Boehm) is an evolutionary, **risk-driven** process model that combines the iterative nature of prototyping with the controlled, systematic aspects of the waterfall model. Software is developed in a series of evolutionary releases, one per loop of the spiral.



Each loop (circuit) of the spiral = one iteration / one more complete version

Fig.: Spiral model (Boehm) with six task regions

1. **Customer communication** — establish communication between developer and customer.
2. **Planning** — define resources, time-lines and other project information.
3. **Risk analysis** — assess technical and management risks and plan how to reduce them.
4. **Engineering** — build one or more representations (models, prototypes).
5. **Construction & release** — construct, test, install and support.
6. **Customer evaluation** — obtain customer feedback; cost and schedule are adjusted after each circuit.

Use: large, high-risk projects. **Limitation:** needs risk-assessment expertise; can be costly for small projects.

(Diagram 2 M, explanation 2 M)

d) Function point metric with calculation.

4 M

The **function point (FP)** metric (Albrecht) measures software size from the functionality delivered to the user, independent of the programming language. It uses five information domain values — external inputs, external outputs, external inquiries, internal logical files and external interface files — each weighted as simple, average or complex. **FP = count total** $\times$ **[0.65 + 0.01 $\times$ ΣF_i]**, where F_i are the 14 value adjustment factors (0–5 each).

Information domain value	Simple	Average	Complex
External inputs (EI)	3	4	6
External outputs (EO)	4	5	7
External inquiries (EQ)	3	4	6
Internal logical files (ILF)	7	10	15
External interface files (EIF)	5	7	10

Calculation (weight chosen as per given complexity):

Information domain value	Count	Weight	Count $\times$ weight
External inputs (EI)	10	3	30
External outputs (EO)	8	5	40
External inquiries (EQ)	6	3	18
Internal logical files (ILF)	3	10	30
External interface files (EIF)	2	10	20
Count total (UFP)			138

$$\Sigma F_i = 38$$

$$\text{CAF (value adjustment)} = 0.65 + 0.01 \times 38 = \mathbf{1.03}$$

$$\text{FP} = \text{UFP} \times \text{CAF} = 138 \times 1.03 = \mathbf{142.14}$$

Answer: FP $\approx$ 142

(Explanation 2 M, calculation 2 M)

Q.3

a) Use case diagram — Online Movie Ticket Booking.

4 M

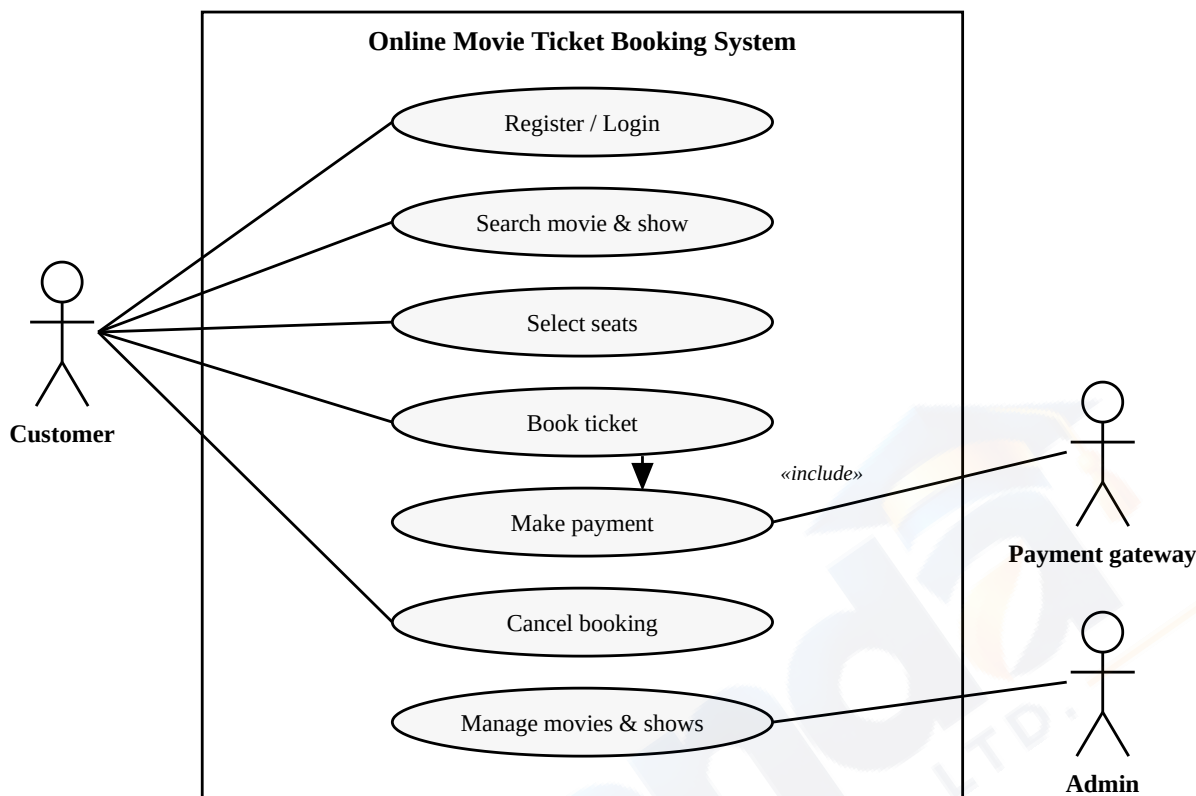


Fig.: Use case diagram — Online Movie Ticket Booking System

- **Actors:** Customer (primary), Admin, Payment gateway (external system).
- **Book ticket «include» Make payment** — a booking is confirmed only after payment.

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

b) Level 0 and level 1 DFD — Student Result Management System.

4 M

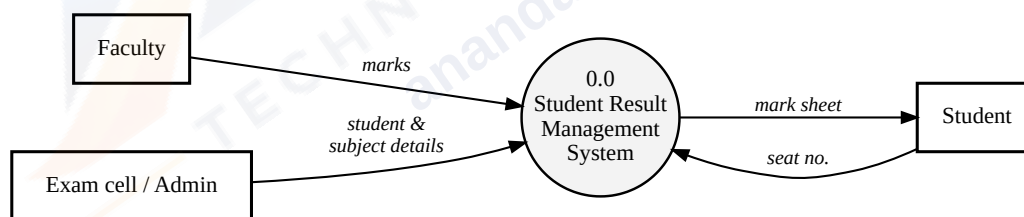


Fig.: Level 0 DFD (context diagram) — Student Result Management System

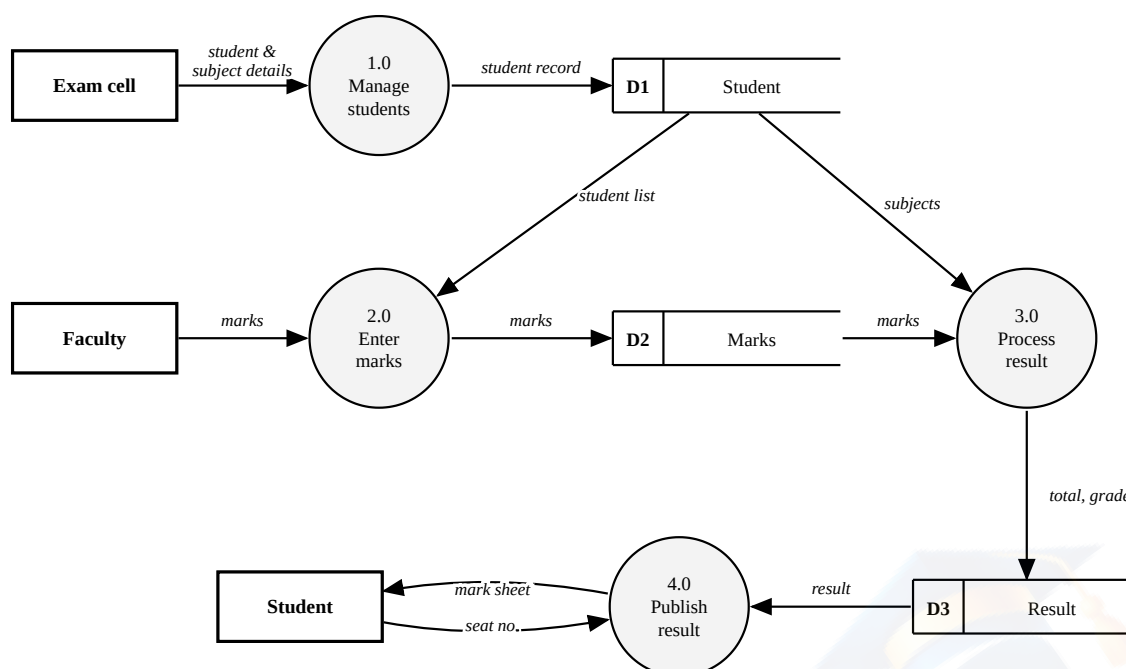


Fig.: Level 1 DFD — Student Result Management System

(Level 0 DFD 2 M, level 1 DFD 2 M)

c) Top-down and bottom-up integration.

4 M

Integration testing combines unit-tested modules according to the program structure and tests them to uncover errors associated with interfacing.

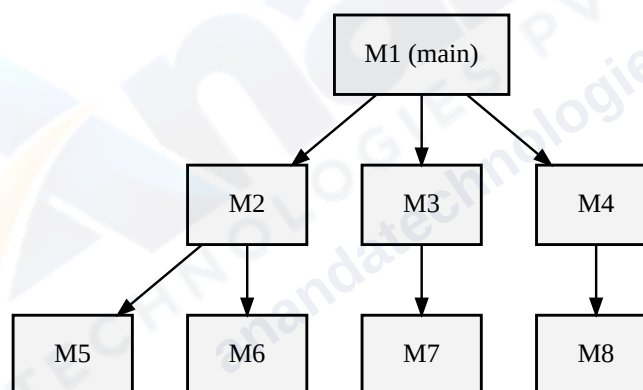


Fig.: Program structure (module call hierarchy) used for integration

Top-down integration: starts with the main module M1; modules subordinate to it are integrated one by one, either *depth-first* (M1, M2, M5, M6, M3, M7 ...) or *breadth-first* (M1, M2, M3, M4, M5 ...). **Stubs** (dummy called modules) replace modules not yet integrated; as each real module is added a stub is replaced and regression tests are run. Advantage: major control and decision points are verified early.

Bottom-up integration: starts with the lowest-level (atomic) modules M5, M6, M7, M8. They are combined into *clusters (builds)*; a **driver** (dummy calling program) supplies test input and collects output for each cluster. Drivers are removed and clusters are combined moving upward until M1 is reached. No stubs are needed.

(Diagram 1 M, top-down 1½ M, bottom-up 1½ M)

d) CPM network, critical path and duration.

4 M

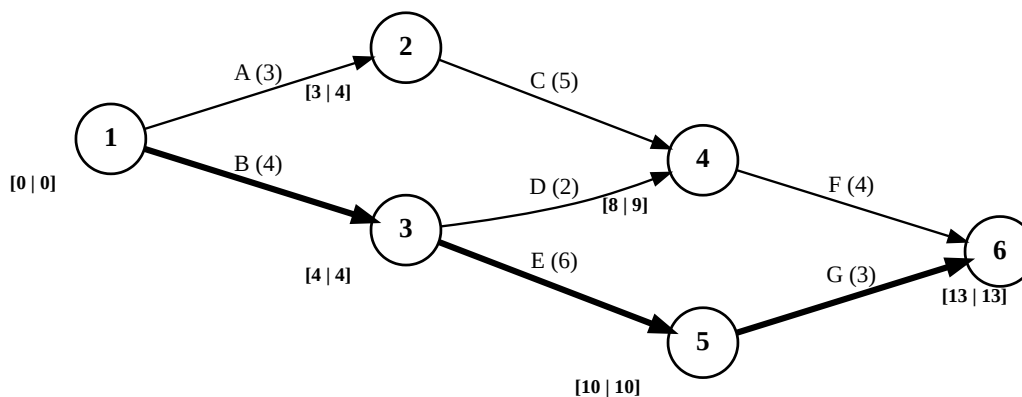


Fig.: Activity network (AOA); $[E | L]$ beside each event; thick arrows = critical path

Forward pass (earliest event time, $E_j = \max(E_i + t_{ij})$) and **backward pass** (latest event time, $L_i = \min(L_j - t_{ij})$):

Event	1	2	3	4	5	6
Earliest time E	0	3	4	8	10	13
Latest time L	0	4	4	9	10	13

Activity times ($ES = E_i$, $EF = ES + t$, $LF = L_j$, $LS = LF - t$, Total float = $LS - ES$):

Activity	i-j	Duration	ES	EF	LS	LF	Total float
A	1-2	3	0	3	1	4	1
B *	1-3	4	0	4	0	4	0
C	2-4	5	3	8	4	9	1
D	3-4	2	4	6	7	9	3
E *	3-5	6	4	10	4	10	0
F	4-6	4	8	12	9	13	1
G *	5-6	3	10	13	10	13	0

Activities marked * have zero total float, so they are critical.

Critical path: 1 → 3 → 5 → 6 (B-E-G)

Project duration = 13

(Network 1 M, forward/backward pass 2 M, critical path and duration 1 M)

Q.4

a) Principles of good planning.

4 M

- 1. Understand the scope of the project:** Scope gives the team a destination; plans cannot be made without knowing it.
- 2. Involve stakeholders in the planning activity:** Stakeholders define priorities and constraints; they help to fix delivery dates and order of increments.

3. **Recognize that planning is iterative:** A plan is never fixed; it is adjusted as work proceeds and feedback is received.
4. **Estimate based on what you know:** Effort, cost and duration are estimated from the team's current understanding of the work.
5. **Consider risk as you define the plan:** High-impact, high-probability risks need contingency planning in the schedule.
6. **Be realistic:** People do not work 100 % of the time; changes and mistakes will occur — build this into the plan.

(Any four, 1 M each)

b) QA vs QC.

4 M

Point	Quality assurance (QA)	Quality control (QC)
Focus	Process oriented	Product oriented
Aim	Prevent defects	Find (detect) and correct defects
Approach	Proactive	Reactive
Activities	Process definition, standards, audits, training, process checklists	Reviews of work products, inspections, testing
Related to	Verification	Validation
When	Throughout the SDLC, before and during development	Mainly after a work product / build is ready
Responsibility	Whole team and management; SQA group	Mainly testing / QC team
Example	Defining a coding standard and auditing that it is followed	Executing test cases on the billing module

(Any four points, 1 M each)

c) Classification of defects.

4 M

Defect: A defect is a **variance (deviation) from the desired product attribute** — i.e. a flaw in a work product (requirement, design or code) because of which the software does not meet its specification or the customer's expectation. A defect found during testing is commonly called a *bug*.

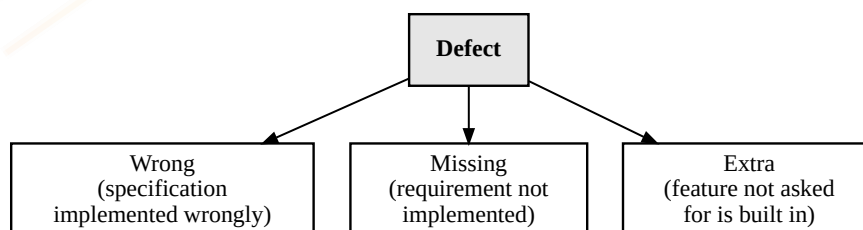


Fig.: Defect categories (variance from specification)

1. **By variance from specification:** *Wrong* — a requirement is implemented incorrectly; *Missing* — a specified requirement is not built; *Extra* — something not specified is built into the product.
2. **By severity:** *Critical* (crash, data loss, no workaround), *Major / high* (main function fails), *Minor / medium* (workaround exists), *Low / cosmetic* (spelling, alignment).

3. By priority: P1 urgent (fix immediately), P2 high, P3 medium, P4 low.

4. By origin (phase): requirement defects, design defects, coding defects, testing / documentation defects.

(Any four bases with explanation, 1 M each)

d) When to start and stop testing.

4 M

When to start testing:

- Testing should **start as early as possible** in the SDLC — from the requirements phase itself (reviews of SRS and design are static tests).
- Defects found early are much cheaper to fix than defects found after coding or after delivery.
- Test planning starts with project planning; test cases are designed as soon as requirements are base-lined.

When to stop testing (testing can never be complete; it stops when exit criteria are met):

- Planned test cases are executed and the required pass percentage is achieved.
- Required code / requirement coverage is reached.
- Defect discovery rate falls below an agreed level and no open critical / high-severity defects remain.
- Testing deadline (release date) is reached or testing budget is exhausted.
- Alpha / beta testing period ends.
- Management decides that the remaining risk is acceptable (exit criteria in the test plan are met).

(Start 2 M, stop 2 M)

e) Manual vs automation testing.

4 M

Point	Manual testing	Automation testing
Execution	Test cases executed by a human tester	Test cases executed by a tool / script
Speed	Slow, time-consuming	Fast; can run unattended (overnight)
Reliability	Less reliable — human errors, fatigue	More reliable and repeatable
Initial cost	Low (no tool needed)	High (tool licences, script development)
Best for	Exploratory, usability, ad hoc testing; frequently changing UI	Regression, load / performance, repeated data-driven tests
Skill	No programming knowledge needed	Scripting / programming knowledge needed
Examples	Tester follows steps in test case document	Selenium, JMeter, Appium, QTP/UFT

(Any four points, 1 M each)

Q.5

a) Requirements — Hotel Room Booking System.

6 M

i) Room booking module

- Customer shall search available rooms by check-in date, check-out date, number of guests and room type.
- System shall show room types with photos, facilities and tariff per night.
- Customer shall book a room; the system shall block the room for those dates so that it cannot be double-booked.
- Customer shall modify or cancel a booking as per the cancellation policy.
- Receptionist shall record walk-in bookings, check-in and check-out.

ii) Customer module

- Customer shall register / log in with name, mobile number, e-mail and password.
- Customer shall manage profile and ID proof details (required at check-in).
- Customer shall view booking history and download invoices.
- Customer shall give feedback / rating after check-out.
- System shall send booking confirmation and reminders by SMS / e-mail.

iii) Billing module

- System shall calculate room charges = tariff $\times$ number of nights, plus extra services (food, laundry) and GST.
- System shall accept advance payment online (card / UPI / net banking) and balance at check-out.
- System shall generate an invoice with bill number and payment details.
- System shall apply refund rules on cancellation.
- Manager shall view daily / monthly revenue reports.

Non-functional requirements (common to all modules):

- Security — encrypted passwords and secure payment gateway.
- Availability — online booking 24×7 .
- Usability — booking completed in a few simple steps on mobile and desktop.

(Each module 2 M)

b) Basic COCOMO — 60 KLOC, organic and embedded.

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 = 60 KLOC = 60,000 LOC

Organic mode ($a = 2.4$, $b = 1.05$, $c = 2.5$, $d = 0.38$):

- i) Effort $E = a \times (\text{KLOC})^b = 2.4 \times (60)^{1.05} = 2.4 \times 73.63 = \mathbf{176.71 \text{ person-months}}$
- ii) Development time $D = c \times (E)^d = 2.5 \times (176.71)^{0.38} = \mathbf{17.86 \text{ months}}$
- iii) Average staff size $SS = E / D = 176.71 / 17.86 = \mathbf{9.89 \approx 10 \text{ persons}}$
- iv) Productivity $P = \text{KLOC} / E = 60 / 176.71 = \mathbf{0.3395 \text{ KLOC/PM}}$ (= 339.5 LOC per person-month)

Embedded mode ($a = 3.6$, $b = 1.2$, $c = 2.5$, $d = 0.32$):

- i) Effort $E = a \times (\text{KLOC})^b = 3.6 \times (60)^{1.2} = 3.6 \times 136.08 = \mathbf{489.87 \text{ person-months}}$
- ii) Development time $D = c \times (E)^d = 2.5 \times (489.87)^{0.32} = \mathbf{18.15 \text{ months}}$
- iii) Average staff size $SS = E / D = 489.87 / 18.15 = \mathbf{27.00 \approx 27 \text{ persons}}$
- iv) Productivity $P = \text{KLOC} / E = 60 / 489.87 = \mathbf{0.1225 \text{ KLOC/PM}}$ (= 122.5 LOC per person-month)

Mode	Effort (PM)	Time (months)	Staff	Productivity (LOC/PM)
Organic	176.71	17.86	9.9	339.5
Embedded	489.87	18.15	27.0	122.5

(Formulae 1 M, organic 2½ M, embedded 2½ M)

c) Time-line chart — Online Food Delivery App (20 days).

6 M

A **time-line chart** (Gantt-type chart) shows each work task as a horizontal bar against a calendar. Here the Online Food Delivery App is planned for **20 working days** = 4 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 & estimation	3	3	1	W1 D3 – W1 D3
Analysis (SRS)	4	5	2	W1 D4 – W1 D5
UI & database design	6	8	3	W2 D1 – W2 D3
Coding (app + admin)	9	14	6	W2 D4 – W3 D4
Testing	15	18	4	W3 D5 – W4 D3
Deployment (Play Store)	19	20	2	W4 D4 – W4 D5

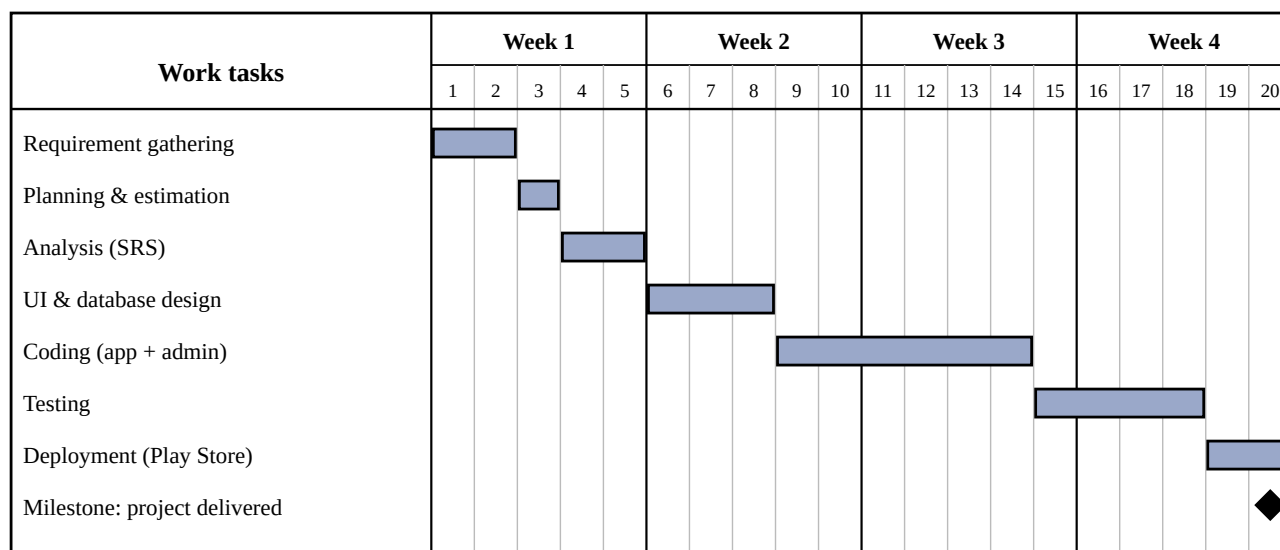


Fig.: Time-line chart — Online Food Delivery App (20 working days, ◆ = milestone)

Each bar is placed against working days; the ◆ marks the final milestone (app released). Reviews at the end of each phase act as intermediate milestones.

(Task allocation 2 M, chart 4 M)

Q.6

a) Six Sigma — DMAIC, DMADV and application.

6 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**.

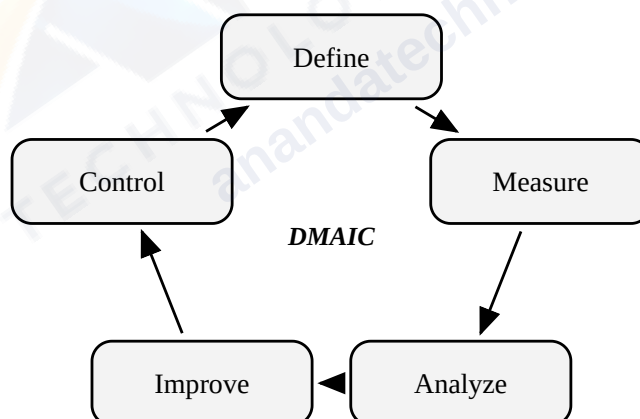


Fig.: DMAIC cycle (Six Sigma — improving an existing process)

DMAIC (improve 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.

DMADV (design a new process / product):

1. **Define** — customer requirements and project goals.
2. **Measure** — customer needs and specifications (critical-to-quality characteristics).
3. **Analyze** — process options to meet the customer requirements.
4. **Design** — the process (details) to meet the customer requirements.
5. **Verify** — the design performance and its ability to meet customer requirements.

Applying DMAIC to late deliveries (food delivery business):

Phase	Action
Define	Problem: orders delivered after the promised time; goal: reduce late deliveries; customer requirement: delivery within promised time.
Measure	Collect data for every order — preparation time, rider assignment time, travel time, distance; compute current percentage of late orders.
Analyze	Find root causes, e.g. slow restaurant preparation at peak hours, delayed rider assignment, wrong address pins.
Improve	Show realistic preparation times, auto-assign nearest rider earlier, address verification with map pin.
Control	Daily dashboard of late-order percentage; alerts when it crosses the limit; standard procedures for restaurants and riders.

(Six Sigma 1 M, DMAIC 2 M, DMADV 2 M, application 1 M)

b) Test cases — fund transfer in mobile banking.

6 M

TC ID	Test scenario	Test steps / test data	Expected result
TC_01	Transfer with valid beneficiary, amount within balance and limit	Beneficiary = saved account, Amount = ₹ 2,000, correct MPIN	Transfer successful; balance reduced by ₹ 2,000; reference number shown; SMS sent
TC_02	Amount greater than available balance	Balance ₹ 1,500; Amount = ₹ 5,000	Message “Insufficient balance”; no debit
TC_03	Amount zero / negative / non-numeric	Amount = 0, -100, “abc”	Validation message “Enter a valid amount”; Proceed button disabled
TC_04	Amount above daily transfer limit	Daily limit ₹ 1,00,000; Amount = ₹ 1,20,000	Message “Daily limit exceeded”
TC_05	Wrong MPIN entered	Correct details, MPIN = wrong 4 digits	Message “Incorrect MPIN”; after 3 wrong attempts transfers are blocked for the day
TC_06	Invalid / unregistered beneficiary account or IFSC	Account no. with wrong IFSC	Message “Invalid beneficiary details”; transfer not initiated
TC_07	Network lost after confirming transfer	Switch off data after tapping Pay	Transaction status shown as pending/failed; no double debit; status updated on reconnect

Pre-condition for all cases: user is logged in to the mobile banking app. Actual result and status are filled during execution.

(Six valid test cases, 1 M each)

c) Test plan — Online Library Portal.

6 M

A **test plan** describes the scope, approach, resources and schedule of the intended testing activities (IEEE 829). Test plan for the Online Library Portal:

Section	Content
1. Test plan identifier	TP-OLP-01, version 1.0
2. Introduction	Testing of the Online Library Portal (search, issue / return, reservation, fines) as per SRS v1.2.
3. Test items	Web application build 1.0 (member module, librarian module, admin module), database.
4. Features to be tested	Member login, book search, reserve book, issue / return, fine calculation, e-book download, reports.
5. Features not to be tested	Third-party SMS gateway internals; hardware barcode scanner firmware.
6. Approach	Black-box functional testing, integration and system testing, regression with automation (Selenium), basic load test for 200 users.
7. Pass / fail criteria	100 % of critical test cases pass; no open critical / high defects; at least 95 % overall pass rate.
8. Suspension / resumption	Suspend if build fails smoke test or login is broken; resume after a fixed build passes smoke test.

Section	Content
9. Test deliverables	Test plan, test cases, test data, defect reports, test summary report.
10. Testing tasks	Prepare test cases, set up environment, execute, log defects, retest, regression, report.
11. Environmental needs	Windows / Linux clients, Chrome and Firefox, test server with MySQL, test data of 5,000 books.
12. Responsibilities	Test lead — planning and reporting; 2 testers — execution; developers — fixing defects.
13. Staffing and training	3 testers; training on library rules and the Selenium tool.
14. Schedule	Test design: week 1; execution: weeks 2–3; regression and report: week 4.
15. Risks and contingencies	Late builds — extend testing by overtime; tester unavailability — backup tester assigned.
16. Approvals	Project manager and test lead sign-off.

(Format with sections 3 M, suitable contents 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