
Module 4: Problem Framing and Prioritization

Define the right problem, find root causes, size opportunities, and choose what to build first with defensible frameworks.



AUTHOR

Varun Kumar

Product Management Master Handbook

v0.1 · 23 October 2026

Built from the 55-lecture course study guide and extended to the 12-module Product Core curriculum. Hypothetical examples are labelled; no real company results are invented.

Contents

4.1 Problem framing	3
4.2 Root Cause Analysis (RCA)	4
4.3 Hypothesis-driven product development	7
4.4 Opportunity sizing	7
4.5 Prioritisation frameworks	8
4.6 Dependency mapping and prioritisation under constraints	17
4.7 MVP definition	19
4.8 Experiment prioritisation	20
4.9 Roadmap planning	21
4.10 Case-solving structure (product design and improvement questions)	22
4.11 Exercises with model answers	23
4.12 Interview perspective	24
4.13 Key takeaways	24
4.14 Check your understanding	26
4.15 Recommended resources	27

MODULE 4

Problem Framing and Prioritization

Define the right problem, find root causes, size opportunities, and choose what to build first with defensible frameworks.

≈ 12 h study

About the numbers

All backlogs, scores and costs below are **hypothetical** (see `datasets/rice_backlog.csv`). Scores are discussion aids, not facts — the point is the *reasoning*, not the ranking.

4.1 Problem framing

"A problem well stated is half solved." **Problem framing** is turning a vague complaint ("checkout is bad") into a specific, evidence-backed, measurable statement the team can act on.

Problem statement:

For [user/segment], [task] is difficult because [observed barrier].
We know this from [evidence]. We will measure improvement using [metric]
and protect [guardrail].

From your course notes (Extra topic 15; Lecture 38)

The problem statement template above is the course's own. A strong statement identifies *who has difficulty, what they are trying to do, why it matters and the desired outcome*; **it does not begin with a tool name**. Weak: "Add AI to travel." Stronger: "Commuters struggle to find reliable platform information before departure; help them find it faster." Validate the claim with users.

Worked example (hypothetical)

Weak: "Reduce cart abandonment." (No user, no cause, no number.) **Better:** "For first-time buyers on mobile, completing checkout is hard because the delivery fee first appears on the payment step. In March, 30% of checkout starters left at the payment step (analytics); 6 of 8 interviewees said the fee surprised them. We will raise checkout completion for first-time buyers from 70% to 76% while keeping complaint rate $\leq 1.3\%$."

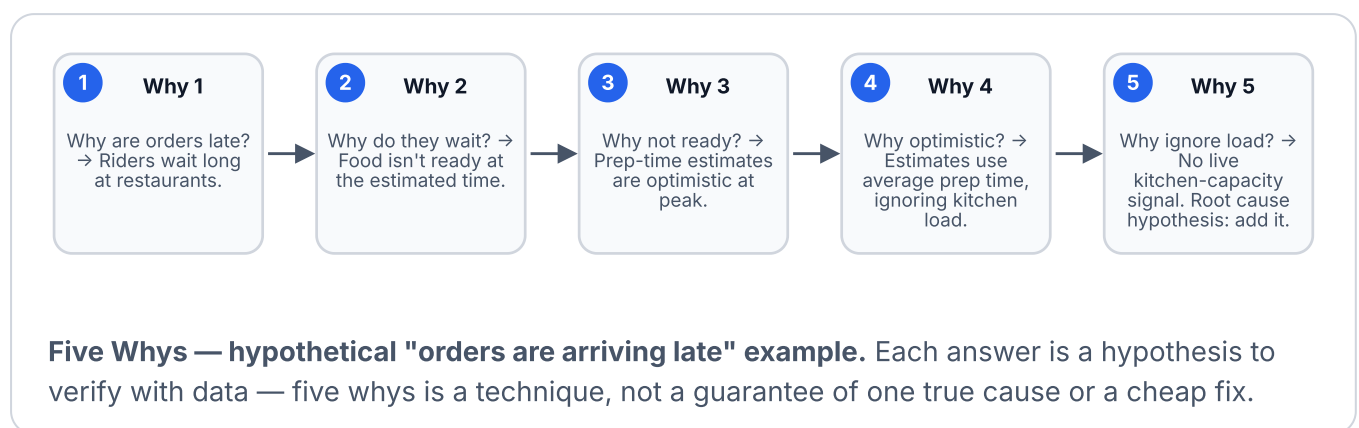
Framing checks: Is the problem **observed** (not assumed)? **Specific** user and moment? **Measurable**? **Solution-free**? Do we know **how big** it is? Is it **worth solving** now versus alternatives?

4.2 Root Cause Analysis (RCA)

RCA looks for the factors that, if fixed, would prevent a problem from recurring — not just the visible symptom. Use several lenses.

4.2.1 The Five Whys

Ask “why?” repeatedly (typically around five times) to move from symptom to a systemic cause.



From your course notes (Lecture 24)

The lecture contrasted immediately proposing a solution with asking successive “why” questions. **Treat both the suspected cause and the proposed solution as hypotheses and test them with evidence.** *The five whys is a technique, not a guarantee of one true cause or a low-cost fix, and it is related to — but not identical with — first-principles thinking.*

Limitations: it can follow a single causal chain and miss parallel causes; answers reflect the asker's knowledge; it needs evidence at each step.

4.2.2 Fishbone (Ishikawa) diagram

A fishbone organises *possible causes* by category (often People, Process, Product, Platform, Policy, Partners) so you don't fixate on the first idea.

PRODUCT

- Fee shown only at payment
- Form has too many fields
- No guest checkout

PLATFORM / TECH

- Gateway timeouts at peak
- Slow page on low-end devices

PROCESS

- No retry flow after failure
- Promo code validation errors

PEOPLE / USERS

- First-time buyers unsure of fees
- Users comparing with rival apps

POLICY / PRICING

- Minimum basket value rule
- Dynamic fees change between cart and pay

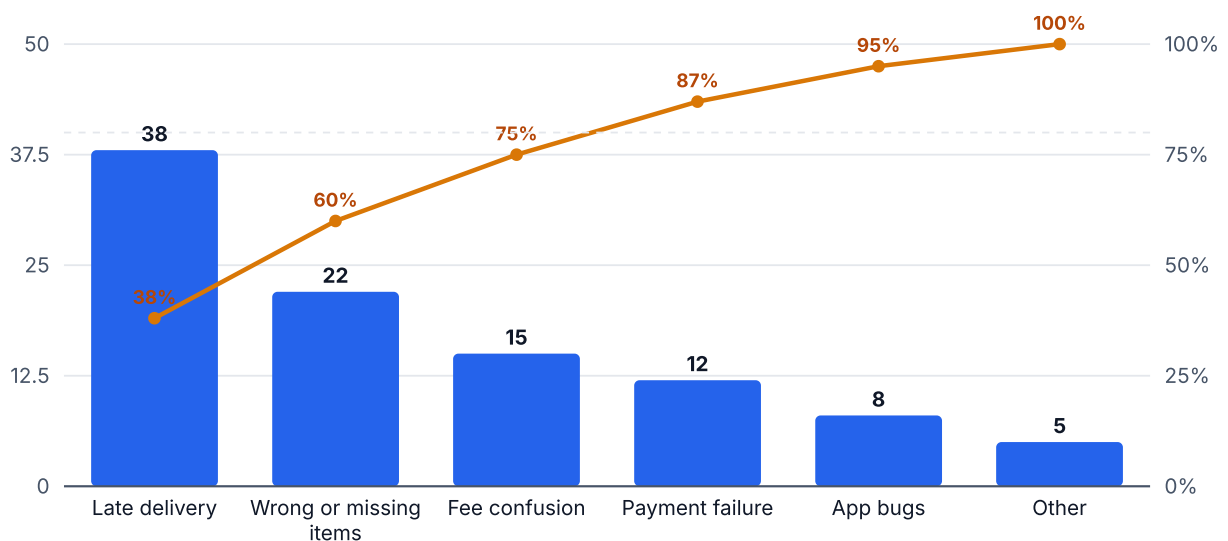
PARTNERS

- Bank 3-D Secure redirects fail
- Wallet provider outages

Fishbone causes for "checkout abandonment at payment step" (hypothetical). Hypothetical. After listing candidate causes, rank them by evidence and impact, then test the top two. Each cell is a hypothesis, not a finding.

4.2.3 Pareto analysis

The **Pareto principle** (often "80/20") observes that a few causes typically produce most of the effect. A **Pareto chart** sorts categories by frequency with a cumulative-percentage line.



Support complaints by category (synthetic: 100 complaints). Synthetic data. The top four categories account for 87% of complaints; the top two for 60%. Pareto is a guide to focus, not a law — fix severity and fixability matter as well as frequency.

4.2.4 First Principles Thinking

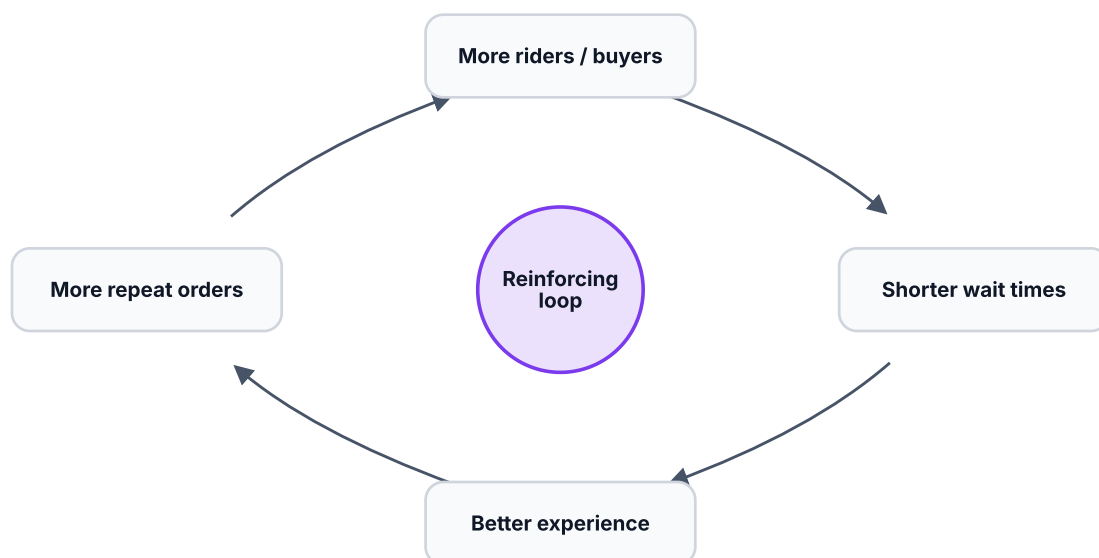
First principles thinking breaks a problem into its fundamental truths and rebuilds the solution from there, instead of reasoning by analogy ("competitors do X"). Steps: (1) state the goal; (2) list current assumptions; (3) challenge each ("is this a law of physics or just how it has been done?"); (4) identify what is actually fixed; (5) design from those truths.

Example (hypothetical)

Assumption: "Our delivery promise is 30 minutes because competitors say 30." First principles: customers choose a *time they can rely on*; the cost drivers are travel time and wait time at the restaurant. Option: promise a *window* based on live load and show an honest ETA — reliability beats a faster-but-missed promise.

4.2.5 Systems Thinking

Systems thinking looks at how parts interact through **feedback loops** and delays. In **reinforcing loops** change feeds on itself (growth or collapse); in **balancing loops** change triggers a correcting response. Product problems often involve both — fixing one metric can trigger a hidden loop elsewhere.



A marketplace loop (reinforcing, hypothetical). More demand makes supply more earning-efficient, which improves experience and brings more demand — until supply is exhausted (a balancing limit appears). Systems thinking asks "where does this loop saturate or reverse?"

Level 3 · Senior PM

The classic failure is a *local optimisation with a hidden balancing loop*: raising surge prices increases driver supply (good) but also riders' cancellations and mistrust (bad). Mapping the loops before intervening reveals which guardrails to watch.

4.3 Hypothesis-driven product development

Turn every bet into a falsifiable statement with a pre-agreed decision rule.

We believe that [change] for [segment] will [outcome/metric movement] because [reason/evidence].

We will know we are right when [measurable signal] by [date/sample].

We will stop or pivot if [disconfirming signal].

From your course notes (Lecture 27; Extra topic 15)

Experiment template: "We believe [change] will improve [metric] for [eligible users] because [reason]. Compare [control] with [variant] for [planned duration], with [guardrails]. Decide using [pre-agreed criteria]." **A hypothesis is a testable prediction.** Review sampling and analysis with an analyst. Module 7 covers the statistics.

4.4 Opportunity sizing

Before building, estimate how big the prize could be. Use **reach × frequency × value per instance × share you can influence**, with explicit assumptions and a range.

Worked example — opportunity size for early fee display

- Checkout starters per month: **120,000**
- Abandonment at payment step: **30%** → 36,000 users
- Share of those caused by fee surprise (from interviews + analytics; *assumption*): **20%** → 7,200
- Share we can recover with early display (*assumption*): **30%** → **2,160 additional orders/month**
- AOV ₹400 → extra GMV ≈ **₹864,000/month**
- Contribution per order ₹16 (from Module 7 example) → ≈ **₹34,560/month** Low case (10% fee-driven, 20% recovery): 720 orders → ₹11,520 contribution/month. High case (30%, 40%): 4,320 orders → ₹69,120. Even the high case is modest: contribution is thin here, so the *strategic* value (trust, repeat orders) may matter more than the immediate margin — which is exactly what a PM should flag.

4.5 Prioritisation frameworks

From your course notes (Lecture 17)

Prioritisation means deciding what to do first when you cannot do everything at once. Compare expected value, effort, risk and urgent needs. **Safety or legal work can be mandatory regardless of score.** A RICE score is a **discussion aid**: $\text{Reach} \times \text{Impact} \times \text{Confidence} \div \text{Effort}$ — use consistent units, check assumptions and revisit the decision. Research informs impact; engineers and other specialists help estimate effort, complexity and dependencies.

4.5.1 Impact vs effort (value vs complexity)

Plot candidates on a 2×2 of expected value vs estimated effort: **quick wins** (high value, low effort), **big bets** (high/high), **fill-ins** (low/low), **time sinks** (low value, high effort).



Value vs effort (hypothetical backlog). Hypothetical placement. Note the plot ignores dependencies, risk and confidence — which is why teams add RICE or WSJF.

Use when: early triage, workshops, explaining trade-offs visually. **Don't use when:** you need a defensible numeric ranking across many items. **Limits:** subjective, ignores reach/confidence/risk/dependencies.

4.5.2 RICE

$\text{RICE score} = (\text{Reach} \times \text{Impact} \times \text{Confidence}) \div \text{Effort}$

Reach = people (or events) affected in a defined period, e.g. users per quarter

Impact = effect per person on the goal: 3 = massive, 2 = high, 1 = medium, 0.5 = low, 0.25 = minimal

Confidence = how well supported the estimates are: 100% high, 80% medium, 50% low

Effort = total work, in person-months (must be > 0)

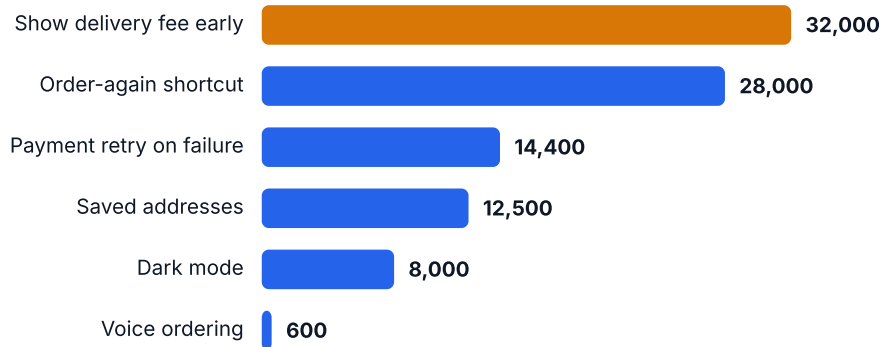
From your course notes (Lecture 17) — corrections

The recording said *reach minus impact equals churn* and that impact must always be numerically lower than reach. **That is wrong:** reach and impact measure different things, and churn is a proportion of users lost over a defined period. The instructor also initially inverted the effort scale then corrected it: **more effort belongs in the denominator and lowers the score.** Use consistent units and a nonzero effort. Hypothetical Amazon features scored in class (search, registration, cart, payment) were *assumptions, a discussion aid, not research*.

Fully worked example (dataset: rice_backlog.csv)

Feature	Reach / quarter	Imp act	Confid ence	Effort (person- months)	RICE = $R \times I \times C \div E$	Ra nk
Show delivery fee early in cart	40,000	2	80%	2	$40,000 \times 2 \times 0.8 \div 2 = \mathbf{32,000}$	1
Order-again shortcut	35,000	1	80%	1	$35,000 \times 1 \times 0.8 \div 1 = \mathbf{28,000}$	2
Payment retry on failure	18,000	3	80%	3	$18,000 \times 3 \times 0.8 \div 3 = \mathbf{14,400}$	3
Saved addresses	25,000	1	100%	2	$25,000 \times 1 \times 1.0 \div 2 = \mathbf{12,500}$	4
Dark mode	30,000	0.5	80%	1.5	$30,000 \times 0.5 \times 0.8 \div 1.5 = \mathbf{8,000}$	5
Voice ordering	8,000	2	30%*	8	$8,000 \times 2 \times 0.3 \div 8 = \mathbf{600}$	6

*Confidence below 50% is allowed here to show the effect of low evidence.



RICE scores (hypothetical backlog). RICE = Reach × Impact × Confidence ÷ Effort. The low confidence and high effort of voice ordering drive its score to near zero — a signal to run discovery before committing, not necessarily a permanent "no".

Sensitivity check (always do this). If confidence in "fee early" falls from 80% to 50%, its score becomes $40,000 \times 2 \times 0.5 \div 2 = 20,000$, and *order-again* (28,000) moves to #1. If effort for "payment retry" is actually 2 (not 3), its score becomes 21,600 (#3 → still #3 but closer). A ranking that flips with small input changes should be treated as a *tie* and decided on other grounds (strategy, risk, dependencies).

Beyond the score — decision notes:

- **Dependencies:** order-again may require saved payment tokens (security review) → check sequence.
- **Mandatory work:** a regulatory or security fix comes off the top regardless of score.
- **Strategy fit:** if the quarter's goal is checkout completion, "fee early" and "payment retry" map directly; "order-again" serves repeat orders.
- **Risk:** payment retry touches money flows — higher QA and rollback needs.

Use RICE when: comparing many comparable initiatives with some data. **Don't use when:** items are not comparable (a legal fix vs a feature), reach/impact are unmeasurable, or the exercise is political theatre. **Limits:** false precision; garbage-in/garbage-out; biased toward high-reach features over deep strategic bets; confidence is subjective.

Defending the decision in a stakeholder meeting (script)

PM: "For Q2 I'm recommending we do three things: fee early, order-again, and payment retry. Fee early scored highest and maps directly to our checkout goal; order-again is cheap and supports repeat orders; payment retry has the biggest per-user impact but costs three person-months and touches payments, so I'm scheduling it after fee early." **Sales:** "But our biggest client asked for voice ordering." **PM:** "I hear it. It's expensive (8 person-months) and our confidence is low — we've had one request. I'll interview five of their users this month and propose a prototype test; if we see broader demand, it enters the next planning cycle with a better score. Meanwhile, what problem does the client need solved — is it hands-free ordering, or speed?" **Why this works:** it shows the data, names trade-offs and risks, offers a path for the stakeholder's idea, and invites correction.

4.5.3 ICE

ICE score = Impact × Confidence × Ease (each scored 1–10; some teams average instead of multiplying)

Worked example (same backlog, 1–10 scales)

Feature	Impact	Confidence	Ease	ICE (I×C×E)
Show fee early	8	8	7	448
Order-again shortcut	6	8	9	432
Payment retry	9	8	5	360
Saved addresses	5	9	7	315
Dark mode	3	8	8	192
Voice ordering	7	3	2	42

ICE vs RICE: ICE omits *Reach* and uses a single "Ease" in place of person-months — faster but less discriminating. **Use ICE** for quick growth-experiment ranking; **use RICE** when audience size varies a lot. Both inherit subjectivity.

4.5.4 MoSCoW

Must have, Should have, Could have, Won't have (for now) — classify scope for a *specific release* or *time-box*.

From your course notes (Lecture 17)

Make the scope and release period clear, and agree with stakeholders on **why** each item belongs in its category.

MUST HAVE

- Show item total + estimated fee before payment
- Label estimates as estimates
- Fallback text if the fee service fails

SHOULD HAVE

- Refresh estimate when address changes
- Event tracking for fee_displayed

COULD HAVE

- Animated fee breakdown
- "Add ₹X to avoid fee" nudge

WON'T HAVE (THIS RELEASE)

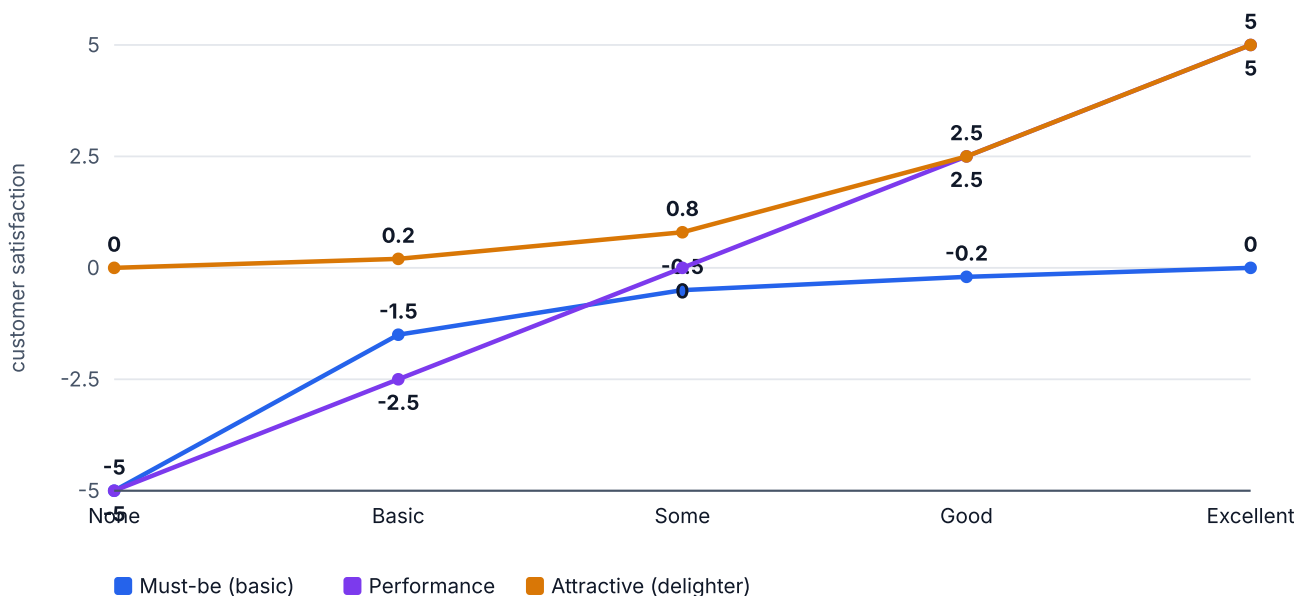
- Changing fee policy
- Personalised fee offers

MoSCoW for a hypothetical "early fee display" release. Hypothetical. A common rule of thumb is that Musts should be a minority of effort (roughly 60% or less) so there is room to absorb surprises — treat it as guidance, not law.

Use when: negotiating release scope against a fixed date. **Don't use when:** you need to rank within a category (it doesn't). **Limit:** everything tends to become a "Must".

4.5.5 Kano Model

The **Kano model** classifies features by how they affect satisfaction: **Must-be** (expected; absence dissatisfies, presence is neutral), **Performance / one-dimensional** (more is better), **Attractive / delighter** (unexpected; presence delights, absence is fine), **Indifferent**, **Reverse**. Features drift over time: today's delighter becomes tomorrow's must-be.



Kano curves (conceptual). Conceptual illustration, not measured data. X = how well the feature is implemented; Y = satisfaction. Must-be features cap satisfaction at neutral; delighters raise it without being expected.

How to run it: For each feature ask two questions — *functional* ("How would you feel if the product had X?") and *dysfunctional* ("...if it did not have X?") — with answers: Like / Expect / Neutral / Live with / Dislike. Map each pair to a category:

Functional ↓ / Dysfunctional →	Like	Expect	Neutral	Live with	Dislike
Like	Q	A	A	A	O
Expect	R	I	I	I	M
Neutral	R	I	I	I	M
Live with	R	I	I	I	M
Dislike	R	R	R	R	Q

Kano evaluation table (A = Attractive, O = Performance, M = Must-be, I = Indifferent, R = Reverse, Q = Questionable). Tally the categories across respondents; the modal category classifies the feature. Use a reasonable sample (dozens) and segment — Kano results differ by user group.

Example (hypothetical)

Payment success confirmation → Must-be (expected). *Faster checkout* → Performance.
Surprise free dessert → Attractive. *Dark mode* for most users → Indifferent; for night-shift users → Performance. **Implication:** never ship a product missing must-bes to chase delighters.

Limits: survey-based; wording sensitive; categories shift over time; doesn't capture cost or feasibility.

4.5.6 Weighted scoring models

Score options against weighted criteria that reflect strategy, then multiply and sum.

Worked example

Criteria and weights: **Customer impact 35%, Revenue 25%, Strategic fit 20%, Ease 20%**

(scores 1–5, higher is better; ease is inverse of effort). | Initiative | Impact | Revenue | Fit | Ease |
Weighted score | |---|---:|---:|---:|---:| | Payment retry | 5 | 4 | 4 | 3 | $5 \times .35 + 4 \times .25 + 4 \times .20 + 3 \times .20 = 1.75 + 1.00 + 0.80 + 0.60 = \mathbf{4.15}$ | | Show fee early | 4 | 3 | 5 | 4 | $1.40 + 0.75 + 1.00 + 0.80 = \mathbf{3.95}$ | | Saved addresses | 3 | 2 | 3 | 4 | $1.05 + 0.50 + 0.60 + 0.80 = \mathbf{2.95}$ | | Voice ordering | 2 | 2 | 2 | 1 | $0.70 + 0.50 + 0.40 + 0.20 = \mathbf{1.80}$ |

Note that the ranking differs from RICE: weighted scoring rewards strategic fit and revenue explicitly, while RICE weights reach. *Frameworks encode values* — choose the one that reflects the decision you are really making, and say why.

Rules: agree weights *before* scoring (to avoid reverse-engineering); keep ≤ 5 criteria; use anchored scales ("5 = measurable impact on the quarterly KR"); test sensitivity by changing weights ± 10 points.

4.5.7 Cost of Delay and WSJF

Cost of Delay (CoD) is the value lost per unit time by *not* having something. **WSJF (Weighted Shortest Job First)** — from the Scaled Agile Framework — prioritises by $\text{CoD} \div \text{job size}$, so small, valuable, time-critical items go first.

$\text{WSJF} = \text{Cost of Delay} \div \text{Job size}$

Cost of Delay = User/business value + Time criticality + Risk reduction or opportunity enablement

(each on a relative scale, e.g. Fibonacci 1, 2, 3, 5, 8, 13, 20)

Worked example (relative scores)

Job	Value	Time criticality	Risk/opportunity	CoD	Size	WSJF	Order
C · Fix gateway timeout before sale season	5	13	2	20	3	6.67	1
A · Fee shown early	8	5	3	16	5	3.20	2
B · Redesign loyalty programme	13	8	5	26	13	2.00	3
D · Platform refactor to reduce incident risk	3	2	8	13	8	1.63	4

Job C is small and time-critical, so it jumps to the front even though B has the highest absolute value. **CoD** is a related variant ("Cost of Delay Divided by Duration"). If a feature earns ₹50,000/week and a delay of 6 weeks is likely, the delay costs ₹300,000 — a concrete way to argue for staffing.

Use when: many items compete for scarce capacity and timing matters (seasonal launches, compliance dates, competitor moves). **Don't use when:** you can't estimate relative sizes honestly. **Limits:** relative scales are subjective; favours small jobs and can starve large strategic ones.

4.5.8 Risk-adjusted prioritisation

Discount value by the probability of success and subtract cost:

$$\text{Expected value} = P(\text{success}) \times \text{value if successful} - \text{cost}$$

Example

Project X — value if successful ₹60 lakh, $P = 50\%$, cost ₹10 lakh $\rightarrow EV = 30 - 10 = \text{₹20 lakh}$.

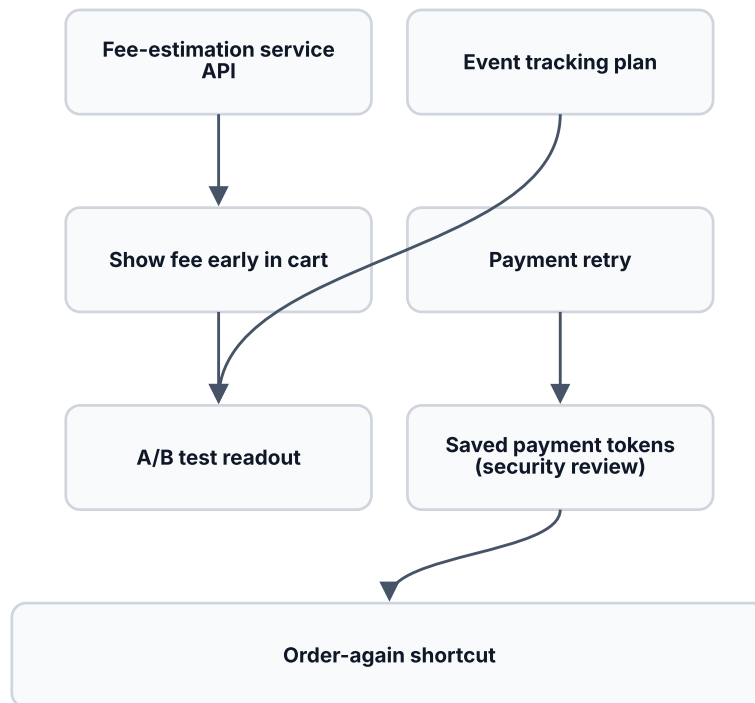
Project Y — value ₹25 lakh, $P = 90\%$, cost ₹5 lakh $\rightarrow EV = 22.5 - 5 = \text{₹17.5 lakh}$. X has a higher EV but is riskier; a PM might still choose Y first (less variance, funds learning) or split X into a cheaper test that raises its probability estimate. EV is only as good as the probabilities — treat them as ranges.

4.5.9 Choosing a framework

Situation	Reach for
Quick triage in a workshop	Impact–effort 2×2
Many comparable features, some data	RICE (or ICE if reach is similar)
Fixed-date release scope	MoSCoW
Understanding customer expectations	Kano
Strategy-weighted choice among few big initiatives	Weighted scoring
Time-sensitive work with scarce capacity	WSJF / Cost of Delay
Choosing between experiments	ICE/PIE on the <i>learning value</i> + cost to run

4.6 Dependency mapping and prioritisation under constraints

A **dependency** is work or a decision that something else relies on. Map them so the ranking reflects *feasible sequence*.



Dependency map for the checkout quarter (hypothetical). Hypothetical. The critical path runs through the fee service and the tracking plan; order-again waits on the token security review. A high-scoring item behind a long dependency may start later than a lower-scoring item.

Prioritising under constraints

1. **Take mandatory work off the top** (legal, security, contractual deadlines, incidents).
2. **Reserve capacity** for reliability and technical debt (e.g., 15–25% — a guideline, justify with data).
3. **Score the rest**, then **sequence** by dependencies, team skills and risk.
4. **Limit work in progress**; finishing beats starting.
5. **Record what you are *not* doing** and why.
6. **Re-plan** on a cadence; treat the plan as a hypothesis.

From your course notes (Extra topic 12)

A dependency is work or a decision that another piece of work relies on. **Technical debt** is the future cost of a technical shortcut or outdated implementation; *not every bug or maintenance item is technical debt*. Showing an estimated fee depends on a delivery-price service and an agreed fallback — if the service is late, a staged release may show “estimate unavailable” rather than invent a number. Ask engineering to identify system owners, risks and realistic alternatives.

4.7 MVP definition

An **MVP (Minimum Viable Product)** is the *smallest useful version or experiment that lets a team learn about an important assumption with real users*. It is defined by the **learning goal**, not by feature count.

From your course notes (Lecture 20; Lecture 43)

An MVP must still be **appropriate and safe** for its intended test. *Prototype vs MVP*: a prototype can be interactive and helps test a concept or flow; an MVP may be a usable early offering or another focused experiment — **the boundary depends on the learning goal, and real users can test prototypes too**. *The recording sometimes equated an MVP with a prototype and implied it must always be a public launch — neither is a universal rule*. A **pivot** is a substantive change in direction, not every small iteration. A payment screen that shows "success" without a real payment provider is a **simulation** — label it. Record what works, what is mocked and what needs security, accessibility and reliability testing.

Type	What it is	Best for testing	Example (hypothetical)
Fake door / smoke test	Landing page or button for a feature that doesn't exist yet	Demand	"Weekly meal plan — join waitlist" page; measure sign-ups
Concierge	Deliver the service manually for a few customers	Value and willingness to pay	A founder hand-builds weekly meal plans for 10 families
Wizard of Oz	Looks automated to the user; humans work behind the scenes	Usability and value before building tech	Chat 'AI planner' answered by a nutritionist
Clickable prototype	Designed screens linked together	Usability, comprehension	Figma prototype of the planning flow
Single-feature product	One core job built for real	Retention of the core loop	App that only generates a weekly grocery list
Pilot / limited release	Real product to a small consented group	Real-world performance and support load	Launch in one neighbourhood with a feature flag

MVP types — pick the cheapest that answers your question. Define the assumption, audience and success measure before choosing. A polished screen does not prove a service works.

MVP checklist: (1) Riskiest assumption named? (2) Success threshold set in advance? (3) Who will use it and are they informed/consenting? (4) Safe — privacy, security, financial, legal? (5) What is simulated vs real, and is that disclosed internally? (6) What decision will the result drive?

4.8 Experiment prioritisation

Rank experiments by **expected learning per unit cost**: how important is the assumption (importance), how uncertain (evidence gap), how cheap to test, and how long to read out. A simple score:

Importance × Uncertainty ÷ Cost-to-test, or use ICE with "Impact" defined as *decision value*.

Example (hypothetical)

Experiment	Assumption	Importance (1–5)	Uncertainty (1–5)	Cost (days)	Score
Fake-door for “weekly plan”	Parents want weekly plans	5	4	2	$5 \times 4 \div 2 = 10$
Price test ₹99 vs ₹149	Willingness to pay	4	4	10	$4 \times 4 \div 10 = 1.6$
Prototype test of onboarding	Users understand setup	3	3	5	$3 \times 3 \div 5 = 1.8$
Run the fake-door first: the answer decides whether the other tests matter.					

4.9 Roadmap planning

A roadmap communicates intended outcomes and likely sequence. Formats: **Now / Next / Later** (outcome-based, no dates), **quarterly themes**, **timeline** (for committed dates). Keep **committed** items separate from **exploratory** bets and show dependencies and confidence.

NOW (IN DELIVERY)

- Outcome: reduce checkout surprise → Fee shown early (A/B)
- Outcome: recover failed payments → Payment retry

NEXT (VALIDATED, BEING SCOPED)

- Outcome: faster repeat orders → Order-again shortcut
- Outcome: reliability → Gateway timeout fix

LATER (IDEAS, NOT COMMITMENTS)

- Voice ordering (needs discovery)
- Loyalty redesign (pending strategy)

Now / Next / Later roadmap (hypothetical). Hypothetical. Each item states the problem/outcome, not just a feature, and "Later" carries no date promise. Review at least monthly and after every major learning.

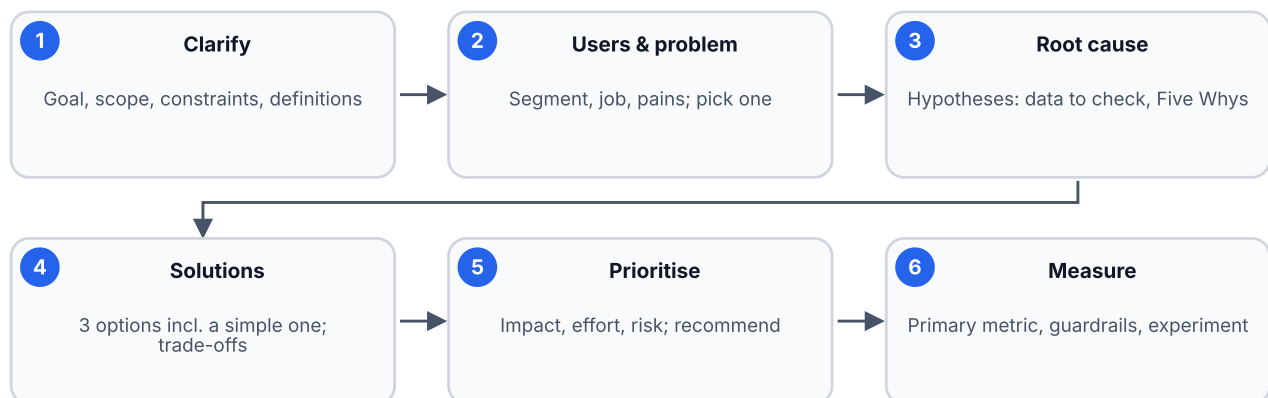
From your course notes (Lecture 13)

Ask engineers about complexity, testing, risks and dependencies, then explain trade-offs to business stakeholders. Collaboratively explore scope or sequencing rather than assuming a visible change is technically simple. A roadmap is an evolving plan, not a guarantee.

4.10 Case-solving structure (product design and improvement questions)

From your course notes (Lectures 8, 11, 36)

Clarify the problem before proposing features. In an interview ask about scope, users and constraints; in written work, state assumptions. Then describe the user, their need, the proposed solution and trade-offs. Pick *one* user problem and investigate it before suggesting features. For community products, define the desired behaviour (useful conversations), not just clicks; measure meaningful participation, retention and safety together; and **filtering is imperfect — avoid promising parents a completely safe experience**. Treat each solution as a hypothesis.



A reliable structure for improvement cases. Use as a skeleton, not a script — adapt to the question and spend most of your time on the user problem and the trade-offs.

4.11 Exercises with model answers

Exercise 1 · Compute and challenge RICE

Score: A) Reach 10,000, Impact 2, Confidence 80%, Effort 4. B) Reach 3,000, Impact 3, Confidence 100%, Effort 1. C) Reach 50,000, Impact 0.5, Confidence 50%, Effort 5. Rank them and say what would change your mind.

Model answer

$A = 10,000 \times 2 \times 0.8 \div 4 = \mathbf{4,000}$. $B = 3,000 \times 3 \times 1.0 \div 1 = \mathbf{9,000}$. $C = 50,000 \times 0.5 \times 0.5 \div 5 = \mathbf{2,500}$.
Ranking: $B > A > C$. B wins on low effort and high confidence despite small reach. Challenge: is B's reach limited to a niche that doesn't serve this quarter's goal? Would C's reach justify a cheap test to raise confidence? If B's effort is really 3, its score drops to 3,000 and A wins — so I'd verify the effort estimate with engineering.

Exercise 2 · Pick a framework

A regulator requires a new consent screen by 30 June. Marketing wants a referral feature; support wants better refund tracking. How do you decide the order?

Model answer

The consent screen is **mandatory** — take it off the top and schedule it first (legal deadline). Then compare referral vs refund tracking with RICE or WSJF: referral's value depends on growth goals and CAC; refund tracking reduces support cost and complaints (risk reduction). Check dependencies (consent flow may touch the same onboarding screens). Present the trade-off with numbers and agree on a review date.

Exercise 3 · Five Whys + test

"Our app's rating fell from 4.4 to 4.1 last month." Do a five-whys chain and name the data that would verify each "why".

Model answer (example)

1. Why did the rating fall? Many 1-star reviews mention payment errors → verify with review tagging.
 2. Why payment errors? A new gateway SDK was released on the 12th → check release dates and error logs.
 3. Why were they not caught? No test for bank redirect flow on Android 14 → check QA coverage.
 4. Why no test? Test matrix excludes new OS versions → review the matrix.
 5. Why? No owner for compatibility testing → process gap.
- Fix:** hotfix SDK; add the OS to the matrix; assign an owner. Validate each step with evidence, not assumption.

4.12 Interview perspective

"How do you prioritise a roadmap?" — structure and model answer

What it tests: judgement, communication, humility. *Structure:* (1) goals and constraints; (2) gather and size opportunities; (3) mandatory work; (4) choose a framework fit for the decision (e.g., RICE) and show a few scores; (5) sensitivity and dependencies; (6) stakeholder alignment and a review cadence. *Model answer:* "I start with the quarter's goal — say, checkout completion. I list candidate problems with evidence and size them. I take mandatory items first, then score the rest with RICE using agreed definitions, and check how sensitive the ranking is to confidence and effort. I map dependencies, then share a one-page recommendation with trade-offs and what we're *not* doing, and revisit it each month as we learn." *Common mistakes:* naming a framework without applying it; ignoring dependencies; treating scores as truth.

"A key stakeholder insists their feature goes first. What do you do?" — 60-second answer

"I'd first understand the problem behind the request and its evidence and urgency. Then I'd show where it sits against our goals using the same framework we use for everything — with numbers — and the cost of delaying it versus the alternatives. If it still doesn't win, I'd offer options: a smaller version, a time-boxed discovery, or a workaround, and agree a date to revisit. I'd document the decision and who made it."

4.13 Key takeaways

- Frame the problem with **user, task, barrier, evidence, metric, guardrail**.
- **RCA tools** (Five Whys, fishbone, Pareto, first principles, systems thinking) generate *hypotheses* to test — not answers.
- Frameworks encode values: **RICE** (reach-weighted), **ICE** (speed), **MoSCoW** (scope), **Kano** (expectations), **weighted scoring** (strategy), **WSJF** (timing).

- Always do **sensitivity checks**, map **dependencies**, take **mandatory** work first and **record what you're not doing**.
- An **MVP** is a learning vehicle; choose the cheapest type that tests the riskiest assumption.
- Roadmaps are **outcome-oriented, evolving plans**.

4.14 Check your understanding

CHECK YOUR UNDERSTANDING

RICE for Reach 20,000, Impact 1, Confidence 80%, Effort 2 equals...

- A. 8,000
- B. 16,000
- C. 32,000
- D. 40,000

Why: $20,000 \times 1 \times 0.8 \div 2 = 8,000$.

In RICE, increasing Effort...

- A. Increases the score
- B. Decreases the score
- C. Has no effect
- D. Only affects Reach

Why: Effort is the denominator, so more effort lowers the score.

A regulatory deadline item should usually be...

- A. Scored with RICE like everything else
- B. Taken off the top as mandatory work
- C. Deferred until after features
- D. Dropped if its score is low

Why: Mandatory legal, security or contractual work is scheduled regardless of score.

Which describes a Kano 'must-be' feature?

- A. Its absence dissatisfies but presence is just expected
- B. More of it always raises satisfaction
- C. It delights when unexpected
- D. Customers don't care

Why: Must-be features are expected basics: absent = dissatisfied; present = neutral.

WSJF is computed as...

- A. Value ÷ effort
- B. Cost of Delay ÷ job size
- C. Reach × Impact
- D. Time ÷ cost

Why: Weighted Shortest Job First divides Cost of Delay by job size.

An MVP is best described as...

- A. The cheapest version of the full product
- B. The smallest useful version or experiment to learn about a key assumption
- C. A prototype only
- D. A public launch

Why: MVPs are defined by what they let you learn; they may be non-public and need not be a prototype.

4.15 Recommended resources

- [Intercom: RICE prioritization](#) — the original RICE write-up.
- [Scaled Agile Framework: WSJF](#).
- [Atlassian: Prioritization frameworks](#).
- [Product Talk: Opportunity Solution Trees](#).
- [Kano model — overview and references \(Wikipedia\)](#).
- [The Scrum Guide](#).