**1️⃣ Cost vs Performance vs Maintainability**

Think of this as the **three‑way balancing act** for any tech decision.  
As a **TPM (Technical Program Manager)**, you are not coding the solution yourself — but you **need to guide teams** so that **business value** and **technical health** stay aligned.

**a) Cost**

* **What it means:**  
  How much money the company will spend to build, run, and maintain the solution.
* **Types of cost to consider:**
  1. **One‑time cost** – Hardware, licenses, initial development effort.
  2. **Ongoing cost** – Cloud bills, maintenance, updates, support contracts.
  3. **Opportunity cost** – If we spend here, what do we NOT spend on?
* **Example (Retail):**  
  If you deploy a high‑end analytics platform that costs ₹25L/year in licensing, is it really giving enough sales insight to justify it? Or could a cheaper cloud‑native service work?

**b) Performance**

* **What it means:**  
  How fast, reliable, and scalable the system is for users.
* **Key points for TPM:**
  + Does it meet user expectations? (Speed, availability, responsiveness)
  + Can it scale for growth without expensive rework?
* **Example (E‑commerce):**  
  If your checkout API responds in 500ms normally but spikes to 5 seconds during a flash sale, **performance** is the bottleneck.

**c) Maintainability**

* **What it means:**  
  How easy it is to keep the system running, fix bugs, and update features over time.
* **Why it matters:**
  + High maintenance = higher cost in long run.
  + Low maintainability = team frustration + delays in delivering features.
* **Example:**  
  A legacy monolith system might work fine now, but every change requires testing the whole app → slows down innovation.

**⚖️ TPM Role — Balancing the Three**

* Increasing **performance** might increase **cost** (e.g., bigger servers, more caching layers).
* Reducing **cost** might hurt **performance** or **maintainability**.
* Making things super **maintainable** might require upfront **investment**.

**TPM Tip:** Always frame trade‑offs in **business impact language**.  
Example:

“If we choose Option A, it’s ₹10L cheaper but will slow down page loads during Diwali sale, which could lose ₹50L in revenue. Option B costs more now but avoids that risk.”

**2️⃣ Build vs Buy Decisions**

When faced with a problem, teams can:

1. **Build** the solution internally.
2. **Buy** or **adopt** an existing product/service.

**a) Build (In‑house Development)**

* **Pros:**
  + Fully customizable to your needs.
  + You own the IP (Intellectual Property).
  + Can integrate deeply with your internal systems.
* **Cons:**
  + High upfront development time and cost.
  + Requires skilled engineers to maintain.
  + Slower time‑to‑market.

**Example:**  
Building your own data warehouse from scratch vs using AWS Redshift. Building might give you full control but will take months and more people.

**b) Buy (Third‑Party Solution)**

* **Pros:**
  + Faster to implement.
  + Proven, battle‑tested solution.
  + Comes with vendor support.
* **Cons:**
  + Licensing/subscription cost.
  + Limited customization.
  + Vendor lock‑in risk (switching later can be expensive).

**Example:**  
Buying Salesforce CRM instead of building your own CRM tool.

**TPM Role in Build vs Buy**

* Facilitate **objective discussions**:
  + **Time to market:** How urgently do we need it?
  + **Long‑term control:** Do we need deep customization?
  + **Budget:** Can we afford the licensing or dev cost?
  + **Risk:** What if the vendor goes out of business?
* Document trade‑offs so leadership understands **why** you picked one path.

**3️⃣ Dependency & Integration Risks**

In modern systems, no product is built in isolation.  
We depend on:

* **Internal teams** (e.g., API from Payments team).
* **External services** (e.g., Stripe for payments, Google Maps API).

**Types of Risks**

1. **Availability risk** – If a dependent system is down, your feature breaks.
2. **Version change risk** – Updates in a dependency may break your integration.
3. **Performance risk** – Slow dependency slows your whole system.
4. **Lock‑in risk** – Hard to replace dependency without big rework.

**Example (Retail Checkout Flow)**

* You depend on:
  + Payment Gateway API (external).
  + Inventory Service (internal).
* If **Payment Gateway** goes down → Checkout fails → Lost sales.
* If **Inventory API** is slow → Users abandon cart.

**TPM’s Mitigation Role**

* Identify critical dependencies **early**.
* Ask:
  + Do we have **fallbacks**? (Backup payment gateway)
  + Do we have **SLAs** with vendors?
  + Can we **gracefully degrade**? (Let customers place orders and confirm payment later)
* Keep **integration testing** part of the delivery plan.

**📌 Summary Table for TPM View**

| **Decision Factor** | **Key Questions for TPM** | **Example** |
| --- | --- | --- |
| **Cost vs Performance vs Maintainability** | Are we spending in the right place? Are we okay with the trade‑off? | Bigger servers for faster flash sale checkout vs cheaper but slower option |
| **Build vs Buy** | Do we need speed or control? Long‑term cost vs quick launch? | Build own CRM vs buy Salesforce |
| **Dependency Risks** | What can break us? Do we have backup plans? | Payment API outage during sale |

**1️⃣ Writing Technical Specs & PRDs with Architectural Alignment**

A **TPM (Technical Program Manager)** isn’t usually writing deep code — but **you must write documents** that engineers, product managers, and leadership all understand.  
The **two core doc types** you’ll often create or review are:

**A) Technical Specification (Tech Spec)**

* **Purpose:** Detailed explanation of *how* a technical solution will work.
* **Audience:** Primarily engineers & architects, sometimes product managers.
* **When used:** Before development starts, to confirm *approach* and *design*.

**Key Sections (TPM‑friendly version):**

1. **Background / Context**  
   Why we are doing this? (Business & technical reason)
2. **Goals & Non‑Goals**
   * Goals = What we must achieve.
   * Non‑Goals = What’s *not* in scope (to avoid confusion).
3. **Proposed Solution**
   * High‑level architecture diagram.
   * Key components and how they talk to each other.
4. **Detailed Design**
   * APIs, data flows, integration points.
   * Sequence diagrams if needed.
5. **Trade‑offs Considered**  
   Why we picked this approach instead of alternatives.
6. **Risks & Mitigation**  
   Dependencies, performance risks, fallbacks.
7. **Testing & Rollout Plan**  
   How we’ll validate and release.
8. **Appendix**  
   Links to related docs, glossary.

**TPM Tip:**  
Even if engineers write most of the tech spec, you as a TPM **ensure it connects to the bigger architecture** (no one building in isolation).

**B) Product Requirements Document (PRD)**

* **Purpose:** Explain *what* the product or feature should do from a **business & user perspective**.
* **Audience:** Product managers, engineers, QA, designers, leadership.
* **When used:** Before solution design starts, to agree on *what needs to be built*.

**Key Sections:**

1. **Problem Statement**  
   The business/user problem we’re solving.
2. **User Stories / Use Cases**  
   Example: “As a customer, I want to save my cart so I can purchase later.”
3. **Functional Requirements**
   * Exact features & behaviors.
   * Acceptance criteria.
4. **Non‑Functional Requirements**
   * Performance (e.g., page load < 2s).
   * Availability (e.g., 99.9% uptime).
5. **Constraints**
   * Legal, compliance, budget, time limits.
6. **Metrics for Success**
   * How will we measure success? (conversion rate, uptime, etc.)

**TPM Tip:**  
Your job is to **bridge PRD → Tech Spec → Architecture**, making sure:

* PRD goals map to actual technical designs.
* Tech spec doesn’t break existing architectural patterns.

**2️⃣ Effective Diagramming for TPMs**

As a TPM, **diagrams are your superpower**.  
They make complex systems understandable for **non‑engineers** while still being accurate for **engineers**.

**A) Sequence Diagram**

* **Purpose:** Show *order of interactions* between systems/components over time.
* **Best for:** Explaining workflows like:
  + Checkout process.
  + API request/response flow.
* **Example: E‑commerce Checkout**

yaml

CopyEdit

Customer → Website: Add item to cart

Website → Inventory Service: Check stock

Inventory Service → Website: Stock confirmed

Website → Payment Gateway: Request payment

Payment Gateway → Website: Payment success

Website → Customer: Order confirmed

* **TPM Value:** Lets you explain *where delays or failures might happen*.

**B) Component Diagram**

* **Purpose:** Show the **building blocks** (services, databases, APIs) and how they connect.
* **Best for:** High‑level architecture overviews.
* **Example:**
  + Web App → Talks to → Order Service → Talks to → Database
  + Payment Service connects to → External Payment Gateway
* **TPM Value:** Helps you check that the design aligns with existing architecture patterns.

**C) Deployment Diagram**

* **Purpose:** Show **where** each component runs (servers, cloud services, regions).
* **Best for:** Explaining physical/cloud deployment.
* **Example:**
  + AWS Region: ap‑south‑1
    - EC2 Instance → Runs Web App
    - RDS → Runs Database
    - S3 → Stores images
* **TPM Value:** Helps you discuss **scaling, cost, and failover** strategies with clarity.

**📌 TPM‑Focused Communication Workflow**

Here’s how these fit together in your role:

1. **Start with PRD** – Capture business need in simple, measurable terms.
2. **Work with engineers to draft Tech Spec** – Ensure it covers dependencies, risks, trade‑offs.
3. **Use diagrams** – Sequence for workflows, Component for structure, Deployment for environment.
4. **Validate against architecture** – No rogue designs; match to org’s patterns & principles.
5. **Communicate trade‑offs to leadership** – In business terms, not deep code details.

**🎯 Real‑World TPM Example**

Imagine you’re launching **“Buy Now, Pay Later”** for your e‑commerce site:

* **PRD** – Says:
  + Business goal: Increase conversion rate by 10%.
  + User story: “As a customer, I want to split my payment into EMIs.”
  + Non‑functional: Checkout latency < 3s.
* **Tech Spec** – Says:
  + We’ll integrate with Razorpay’s BNPL API.
  + Sequence diagram of checkout with BNPL call.
  + Component diagram showing payment service integration.
  + Deployment diagram showing services in AWS Mumbai region.
* **Your TPM Job**:
  + Ensure API dependency risks are logged.
  + Confirm performance requirements are met.
  + Keep all docs aligned with the **existing architecture** (not a one‑off hack).

**1️⃣ Writing Technical Specs & PRDs with Architectural Alignment**

A **TPM** is often the bridge between **business goals** and **technical execution**.  
Your **specs** (Technical Specs, PRDs — Product Requirements Documents) are the **source of truth** for what needs to be built and **how** it should align with architecture.

**A) PRD / Technical Spec Basics**

Think of a **PRD** like a **map** for engineers:

* **What** to build → Features, requirements, acceptance criteria.
* **Why** it matters → Business value, user pain points.
* **Constraints** → Budget, timelines, compliance.
* **Alignment** → Must fit within existing **system architecture**.

**B) Structure of a TPM‑friendly PRD**

1. **Title & Summary** – Clear, concise.
   * *Example:* "Flash Sale Checkout Optimization – Reduce Latency from 5s to 1s"
2. **Background** – Why we’re doing this.
   * *Example:* Sales drop 15% during high traffic; customers abandon carts.
3. **Goals & Success Metrics** – Measurable outcomes.
   * *Example:* 90% of checkout requests <1s response time.
4. **Requirements** – Functional & non-functional.
   * *Functional:* Handle 5x normal order volume.
   * *Non-functional:* 99.9% uptime during sale.
5. **Dependencies** – Other teams/services needed.
6. **Constraints & Risks** – Compliance, budget, vendor limits.
7. **Architectural Alignment** –
   * Use **existing platforms/services** where possible.
   * Avoid creating **technical debt** by bypassing architectural guidelines.
   * Work with **architects** to confirm your design is consistent with enterprise standards.

**C) TPM Tips**

* Don’t over-specify **how** to code — leave engineering freedom.
* Ensure **requirements** are testable and measurable.
* Link to architecture diagrams so engineers see **the bigger picture**.

**2️⃣ Effective Diagramming**

Diagrams are **visual contracts** — they help align engineers, architects, and non-technical stakeholders.

**A) Sequence Diagrams**

* **Purpose:** Show how systems/actors interact **over time**.
* **When to use:** Explaining workflows, API calls, user actions.
* **Example:**  
  Flash Sale Checkout → User clicks *Buy* → Payment API → Inventory API → Confirmation Email.

yaml

CopyEdit

User → Checkout Service: Place Order

Checkout Service → Payment Gateway: Authorize Payment

Payment Gateway → Checkout Service: Payment Approved

Checkout Service → Inventory Service: Reserve Item

Inventory Service → Checkout Service: Confirm Reservation

Checkout Service → User: Order Confirmation

**B) Component Diagrams**

* **Purpose:** Show **logical components** of a system and their relationships.
* **When to use:** Explaining **what systems exist** and **how they connect**.
* **Example:**
  + Components: Checkout Service, Payment Service, Inventory Service.
  + Show interfaces between them.

**C) Deployment Diagrams**

* **Purpose:** Show **where** components run (servers, regions, cloud services).
* **When to use:**
  + Discussing **scaling** and **resilience**.
  + Planning **multi-region deployments**.
* **Example:**
  + AWS Region A: Web Tier, API Tier, Database.
  + AWS Region B: Failover Database.

**TPM Diagramming Tips**

* Keep diagrams **simple enough** for non-engineers but **detailed enough** for engineers.
* Use **consistent icons** (AWS, Azure, GCP icons if cloud).
* Always include a **legend** if using special symbols.
* Avoid clutter — **one diagram, one main message**.

**3️⃣ Communicating with Engineers, Architects & Leadership**

A TPM must **tailor** the same message for **different audiences**.

**A) With Engineers**

* Focus on **technical clarity**.
* Speak in terms of APIs, latency, throughput, dependencies.
* Example:

“Payment API SLA is 200ms. Current latency is 350ms. This risks hitting our checkout SLA of 1s.”

**B) With Architects**

* Focus on **architectural consistency** & **scalability**.
* Example:

“We’ll use our existing event bus for order events instead of building a custom queue to keep architecture consistent and reduce maintenance.”

**C) With Leadership**

* Focus on **business impact** & **risk**.
* Avoid deep tech — give impact, cost, timeline.
* Example:

“Without optimization, we could lose ₹50L in potential revenue during the sale due to slow checkout.”

**TPM Communication Checklist**

✅ Understand **who** you’re talking to.  
✅ Translate between **business language** ↔ **technical language**.  
✅ Use **data + visuals** to back your points.  
✅ Summarize decisions & action items after meetings.

**📌 How All This Fits Together for TPMs**

1. **Write clear PRDs** → Everyone knows *what* to build and *why*.
2. **Use diagrams** → Makes architecture & flows crystal clear.
3. **Adapt communication style** → Ensures alignment from engineers to executives.