Technical delivery framework for high-complexity projects

Purpose

This framework brings structure and clarity to high-complexity projects—where multiple teams, systems, and decisions intersect. It's built from years of experience delivering complex technical initiatives with deep dependencies. Without this structure, projects risk misalignment, delays, and rework. With it, we drive alignment, surface risks early, and deliver with confidence.

TL;DR

For projects with high technical and/or business complexity: first, define the full final solution and delivery path — then execute in clear, scoped phases. Don't start building until the end state is defined.

Why it matters: skipping structured planning leads to critical non-reversible architectural decisions, lack of scalability, poor integration choices, compounded rework cost, timeline collapse, stakeholder distrust, and solutions that fail to meet business or user needs.

When this framework is needed

Low complexity projects — not required (e.g. websites/webpages)	Step-by-step changes (e.g., website edits, static content pages). Work can proceed and iterate linearly, **working forward**.
High complexity projects — recommended (e.g. applications, platforms, systems)	Involve systems integration, workflows, user behaviors, and long-term maintenance. Must be approached by **working backwards** from the end state to avoid fragmented ("Frankenstein") outcomes.

Involve the right people

Responsibility Area	Best Practices
Decision Authority	Appoint a lead with authority to say "No" to feature creep and conflicting requests (can be a Product Lead).
Process Owner	Assign a generalist with technical, business, and institutional knowledge. This person identifies red flags, reassesses scope, pulls in SMEs, and ensures cross-functional alignment (can be a Product Lead or Technical lead).
Project Oversight	Where possible, involve a dedicated Project Manager to track timelines, sign-offs, blockers, and deliverables.
Stakeholder Identification	Identify all impacted or involved teams. Secure engagement from both subject matter experts (SMEs) and domain decision makers.

Stage discipline

- Each stage may include feedback loops.
- Red/yellow flags should be identified explicitly.
- Stages may overlap, but should not be skipped.

Cross-stage best practices

- **Feedback Loops:** add review/re-evaluate points at every stage, especially before build and delivery.

- **Documentation:** maintain clear records of goals, decisions, and known limitations.
- **Dependencies:** track external dependencies explicitly (e.g., vendors, systems, approvals).
- **Risk Management:** flag high-risk areas early. Create fallback plans.
- **Decision Logs:** maintain a single source of truth for major decisions.

Stages

Stage	Guiding Questions	Key Actions	Critical Tips
Identify Problem	What are the problems with the old system? What is the **MAIN PROBLEM** we are trying to solve? Who is the **PRIMARY CUSTOMER**?	• Identify systemic issues: outdated logic, slowness, manual work, instability • Document root causes • Define the main problem this project is solving • Define the primary customer who will benefit IMPORTANT: ❗ **Avoid discussing solutions**	• Understand WHY those are important (watch for personal 'wants' from internal teams vs actual needs of users/biz) ✔ Ensure to have: subject matter experts, decision makers, impacted teams on both the **biz & implementation sides**

Identify Goals	What kind of outcome are we trying to achieve?	• Gather desired outcomes from all stakeholders • Separate must-haves from nice-to-haves • Flag conflicting requests • Link goals to solving the main problem for the primary customer	• Understand WHY those are important (watch for personal 'wants' from internal teams vs actual needs of users/biz) • Watch for conflicting requests. • Use authority to say NO as necessary ✔ Ensure to have: subject matter experts, decision makers, impacted teams on both the **biz & implementation sides**
Identify Project Team	Who needs to participate in decision making?	• Include decision makers + specialists from: • Engineering • Data • Product • Design, accessibility • Marketing/sales • Legal, finance, supply chain	✔ Ensure to have: subject matter experts, decision makers, impacted teams on the **biz & implementation sides**

		• Ensure all impacted business + implementation roles are covered	
Identify Requirements	What are the target characteristics?	• Define functional + non-functional requirements (e.g., performance, reporting, security) • Sort into must-have, nice-to-have, out-of-scope • Include edge cases and constraints • Finalize with group approval IMPORTANT: ❗ **Final group approval - a MUST**	• Understand WHY those are important (watch for personal 'wants' from internal teams vs actual needs of users/biz) • Remember about non-functional requirements (tracking/reporting, translations, security etc.) • Watch for conflicting requests. • Use authority to say NO as necessary ✔ Ensure to have: subject matter experts, decision makers, impacted teams on the **biz & implementation sides**

Determine Solution Options & Viability	Is this even possible? How?	• Can something like this be built? • What are the possible options/solutions, and what's the best one? • How'd we approach it? • What systems would be involved? • Is it feasible? Is it usable? • If so - create high-level system design for "state B" • What are the limitations? • What are the one-way door decisions? • What are the dependencies?	• Ensure to have: subject matter experts, decision makers, impacted teams on the implementation side **Important notes:** • This is where the tech team can start working autonomously, and the wider team can step back.

Repair or Replace	Can we adapt the current system or must we rebuild?	• Evaluate incremental vs rebuild • Define phased vs full migration • Confirm what form “State B” will take and how to get there • Document tradeoffs and transition plans	• Ensure to have: subject matter experts, decision makers, and impacted teams on the implementation side **Important notes:** • This is where you can have a preliminary best-guess estimate (based on technical implementation leaders' “gut feeling”). This estimate is contingent on other factors like unknowns, other teams and dependencies, concurrent projects, etc. Such estimate could be used in planning/sequencing, but it should NOT be taken or communicated as an expected deadline.

Assess Work: POC, Design, Timelines	What are the distinct chunks of work?	• Define key deliverables/components to be built • Draft low-level technical design for each chunk • Design prototypes • Migration path (e.g. data migration) • Key non-technical activities identified (testing, trainings, communications) • Timelines & milestones identified & agreed upon IMPORTANT: ✅ **“State B” is defined here** (get the wider team sign-off before building)	• Empower implementation teams to make decisions within their areas of expertise/responsibility and communicate back to the wider team **Important notes:** • This is where you can have reliable LOE & time estimates • This is where you can start on simple projects • This is where we tend to start on all projects, which leads to multiple problems and a waste of time!!!

Build	Main execution phase	• Each specialized team takes on design & decision making for their area with a CLEAR COMMON goal in mind • Coordinate across integration points • Keep blockers surfaced and tracked	• Empower implementation teams to make decisions within their areas of expertise/responsibility and communicate back to wider team • Build in phases if needed
Test	Testing and refinement	• Internal testing • User testing • Pilots • Refinement and iterations • Address blockers before launch approval	✔ Coordinate with wider team (**biz & implementation sides**)

Deliver	New product shipped to users	• Coordinate launch, support readiness, and documentation • Ensure training, comms, and incident plans are active • Monitor adoption and performance	✔ Coordinate with wider team (**biz & implementation sides**)