

Legacy overhaul playbook

A high-level one-year structure for overhauling a legacy system. Written for day one of the programme, with no prior context assumed. By the end you should know what happens each quarter, who signs off on what and what the words on the timeline mean.

By [Dr. Kavita Jain](https://drkavitajain.ch), Basel · drkavitajain.ch · August 2026

In one sentence: map the real system from its code, agree what it should become, turn that into a tracked plan, then build it with a human checking every meaningful decision, running old and new side by side until it's safe to switch.

1. The four phases

The programme moves through four phases in order. Each one produces something concrete that the next phase depends on. You can't skip ahead: there's nothing to build against until design exists, and nothing to design against until analysis exists. The interdependency acts as your cross-check.

1 Analyse *what the old system actually does*

Automated agents scan the existing codebase and rebuild a map of the real system: use cases, workflows, architecture, integrations, data model and security posture. This runs against the code itself, not the documentation, because documentation drifts from reality and code doesn't.

- Runs alongside interviews with subject matter experts (SMEs), the people who actually use the old system. Some of what looks like messy legacy code is a deliberate fix for a real problem nobody wrote down.
- Includes a check of what the data model says against what's actually sitting in the database. The two are often not the same.

2 Design *what it should become*

From the current-state map, define the target state in the same categories, plus the transition itself: a work breakdown structure (WBS), a phased plan, a risk register with mitigations, a timeline, resourcing and a data migration approach.

- The target state can't come purely from extrapolating the old code forward. It needs input from whoever owns business direction, on what the new system must do that the old one couldn't.

3 Plan *the transition plan goes live*

The transition plan stops being a document and becomes a live execution plan. Agents break it into tasks, allocate the work and track progress automatically as things change.

- Allocation still needs a human check. Agents don't know who's actually available or what they're strongest at unless someone feeds that in.

4 Implement *build, checked, in parallel*

Agents build against the plan. Every meaningful decision routes through a named human checkpoint before it ships. Old and new systems run in parallel until cutover is safe, then the old one is decommissioned.

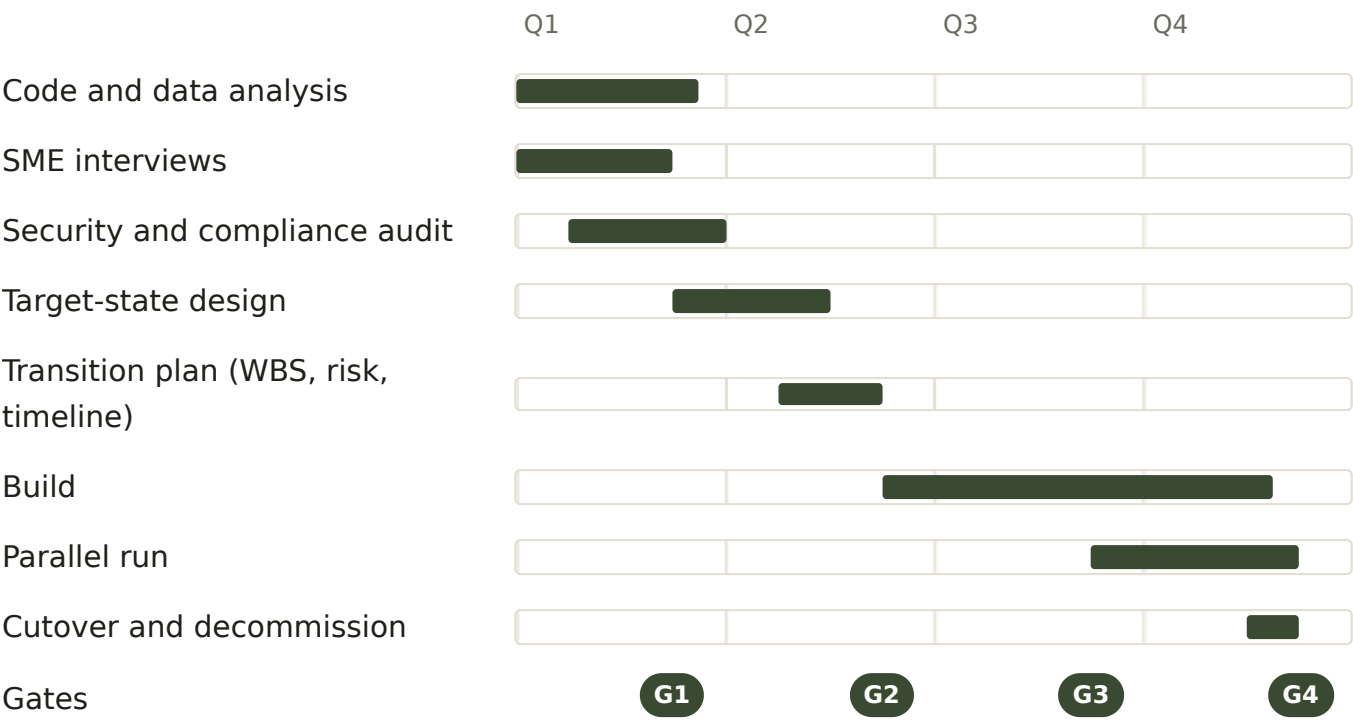
- What counts as a meaningful decision is defined in writing per programme, not left as a feeling. Otherwise the checkpoint either bottlenecks everything or becomes a rubber stamp.

Why this order matters. Most failures in this kind of programme come from skipping straight to building. Analysis without design gives you a faithful copy of the old mess. Design without a business voice gives you a technically clean system nobody asked for.

2. The year, mapped

The whole year at the level of detail that matters at the start: which workstreams run when, and where they overlap. Day-by-day task detail comes later, inside each

quarter.



3. The four gates

A gate is a point where the programme cannot move forward until a named person signs off on a specific artefact. No artefact, no signature, no progress. This is what stops the programme turning into a rolling status update.

Gate	What must exist	Who signs
G1: Architecture sign-off	A written current-state map covering use cases, workflows, architecture, integrations, data model and security posture, built from the code itself.	Technical lead
G2: Design and plan approved	Target-state design at the same depth as the current-state map, plus a complete transition plan: WBS, risk register with mitigations, timeline, confirmed resourcing and migration approach. Budget locked, not estimated.	Technical lead and business stakeholder, jointly

Gate	What must exist	Who signs
G3: Build readiness	Enough of the build complete and tested that a parallel run against the live old system won't create risk. Rollback plan written down.	Technical lead
G4: Cutover approval	Explicit cutover criteria met, not a feeling that it's probably fine. Compliance sign-off recorded with an audit trail, not a chat thread.	Technical lead, business stakeholder and compliance

4. Rules of the road

1. **Docs don't get trusted, code does.** The current-state map is built by scanning the codebase directly. Documentation is a cross-check, never the source.
2. **Nothing ships without a human checkpoint.** Every meaningful build decision is reviewed before it goes live. What counts as meaningful is defined in writing before build starts.
3. **Old and new run side by side.** The old system stays live until the new one has proven itself under real conditions. Cutover only happens against stated criteria.
4. **Every gate needs a name attached, not a team.** If a sign-off can't be traced to one person, it isn't a gate.
5. **Compliance runs through the whole thing, not at the end.** In a regulated environment the audit trail is built as you go, so there's a record of who approved what and why.

5. Glossary

WBS, work breakdown structure

The transition plan broken into discrete tasks, small enough to assign and track individually.

SME, subject matter expert

Someone who uses or built the old system and knows why it works the way it does, beyond what the code shows.

Current state and target state

Current state is what the old system does today, mapped from its code. Target state is what the new system is meant to do, defined during design.

Parallel run

Running the old and new systems at the same time so the new one can be checked against real activity before anyone depends on it alone.

Cutover

The point where the new system takes over completely and the old one is switched off.

Gate

A checkpoint between phases that requires a named person's sign-off on a specific artefact before work can continue.

Legacy overhaul playbook by Dr. Kavita Jain, drkavitajain.ch. Free to use and share with attribution. Questions or want help running one? [Get in touch.](#)