Whitepaper

Team best practices

How fast teams get the right product to market at the right time

FTTM methodology series.

A lateralworks Fast Time To Market (FTTM) methodology paper — the 15 practices that turn an empowered team into a fast team. Companion to *Host best practices*.

Prepared by
lateralworks
FTTM methodology

Date
April 2026
Companion edition

Online
lateralworks.com
FTTM series

Table of contents

Team best practices

Core thesis. A fast team organizes itself around the customer (right-product), assembles the smallest competent group of role-owners (right-team), and uses its schedule as the primary tool for driving decisions and exposing gaps (right-time). Inside a provisioning host, these 15 practices cut time-to-market by 30 to 50 percent. Inside an interrupting host, they recover only a fraction of that, which is why this paper has a companion.

Overview

Abstract

Companies invest heavily in product development teams. They hire experienced engineers, recruit program managers, buy collaboration tools, and run training programs. Most of those investments produce little change in time-to-market. The team itself is only half the picture. The other half is the host: the executive staff and functional organization that surrounds the team and decides what it can do. The companion paper, *Host best practices*, describes 13 host practices. This paper describes the 15 team practices that operate inside a healthy host.

The 15 practices fall into three groups from the lateralworks Fast Time To Market (FTTM) framework. **Right-product** covers six practices for getting the team to a product customers actually want and will pay for. **Right-team** covers two practices for turning a roster of names into a team that can deliver. **Right-time** covers seven practices for turning a schedule from a reporting artifact into a tool the team uses every day.

The practices come from three decades of direct observation of fast-to-market teams, starting with a 15-company study of more than 500 practitioners in the early 1990s. They connect to a broader body of research: Wheelwright and Clark on heavyweight team structure [11], Reinertsen on product development flow [2], Smith and Reinertsen on developing products in half the time [4], Khurana and Rosenthal on the fuzzy front end [7], and the lean and agile traditions descended from Toyota [1] and elaborated by Ries [8] and others.

The paper is written for program managers, chief engineers, and team leads who want a concrete checklist for how a fast team operates day to day. Executives who have read the host paper can use it to see what team-side practices their hosts will need to enable. Each practice has a short summary, a Best vs Normal contrast, and a practical example drawn from a lateralworks engagement or the published literature.

01

Foundation

The team: companion to the host

A product development team is the visible engine of new product creation. It writes the spec, builds the prototype, runs the qualifications, and answers for the outcome at the executive review. When a product ships late, leadership looks at the team first, and often nothing else. That is a mistake. The team's performance is bounded by the host that surrounds it: the executive staff, the functions, the development framework, the portfolio, the decision-making system. A world-class team inside an interrupting host runs only marginally faster than a mediocre one. A merely-good team inside a provisioning host can outperform an industry leader [1, 11].

The companion paper, *Host best practices*, describes the 13 practices that make a host fast. This paper describes the 15 practices that make a team fast inside that host. The two are designed to be read together. Read the host paper first if you are an executive deciding what the company will permit. Read this paper first if you are a program manager, chief engineer, or team lead deciding how the team will operate. Either order works. Both papers are required to accelerate time-to-market.

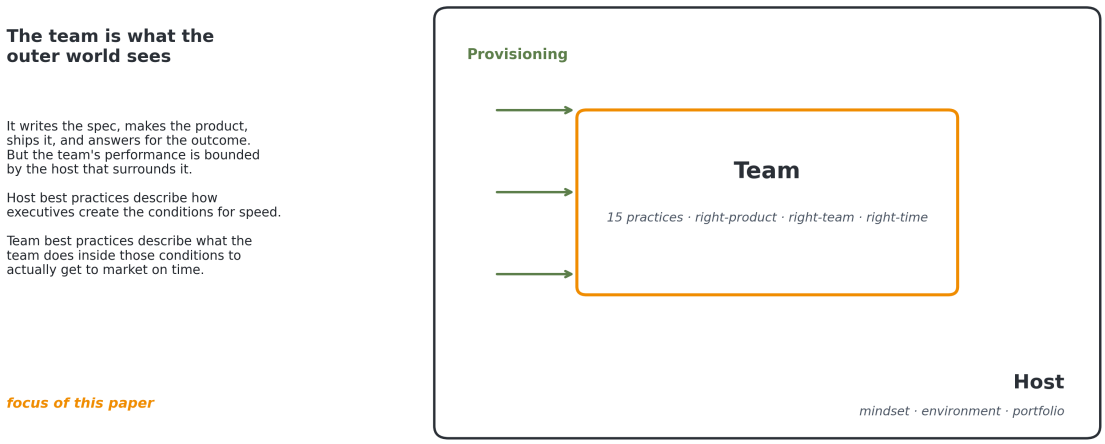

Figure 1. The team is the visible engine. The host is the structural envelope. Team practices operate inside the conditions the host creates.

What this paper does not cover

This paper does not cover the executive mindset around speed, the design of the development framework, the portfolio prioritization system, the interrupt-and-provisioning matrix, the empowerment matrix, or the break-even time scorecard. Those are host concerns, addressed in the companion paper. What this paper does cover is what the team itself does: the practices that distinguish a 5-person team that ships in 9 months from a 25-person team that ships in 24.

Right-product, right-team, right-time

The 15 practices group naturally into three diagnostic categories. A team that consistently ships products customers do not want has a Right-Product problem. A team with the right idea but broken execution has a Right-Team problem. A team that executes well but always misses the market window has a Right-Time problem. Most teams that come to lateralworks have problems in two or three categories at once, but the dominant pattern is usually clear within the first two days of assessment. The framework on the next page makes the grouping concrete; the rest of the paper walks each practice in turn.

The framework

15 practices, three groups

The 15 team practices fall into three groups, summarized at a glance below. Right-product covers what the team builds; right-team covers who builds it; right-time covers when it gets built and how the team manages the schedule that drives all three.

Figure 2. The 15 team practices at a glance, organized by Right-product, Right-team, and Right-time. Each group answers a different question; together they describe how a fast team behaves day to day.

02

Right-product
Build the product the customer wants

Six practices that get the team to a product the customer actually wants. The common thread is the customer's voice. In normal organizations, the customer is heard twice: at the start and at beta. In fast organizations, the customer is in the room throughout, in different ways at different phases. The product the team ships is the one the customer asked for, in their own words, validated against their actual workflow.

1. Co-develop with tier-1 customers

Summary. Fast teams treat one or two leading customers as development partners. Engineers talk to engineers. Schedules and unfinished designs move in both directions. The customer's roadmap and the team's schedule pull on each other. The result is a product that fits the customer's system before it ships, and a relationship that converts directly to first-design-win revenue.

Normal organization	Best organization
The customer is treated as a vendor-supplier relationship. Contact runs through the sales account team or marketing, who summarize, filter, and occasionally reinterpret what the customer actually said. Engineers are kept away from customers because someone might say the wrong thing. The team builds against a documented MRD and discovers at beta that the real customer wanted something different.	One or two tier-1 customers are engaged as co-developers. Engineer-to-engineer channels are open and used. The team understands the customer's customer and the system the product slots into, not just the spec. Schedules are shared in both directions. Joint problem-solving builds trust that survives the inevitable bumps and converts the customer into a first-design-win at launch.

From a lateralworks engagement

A networking-equipment client was losing socket decisions to a faster competitor. By the time the client's product shipped, the customer had already qualified the competitor. The team set up partner-lab arrangements with two key carriers, stationed two engineers at customer sites for a week per quarter, and shared its working schedule weekly. Within 12 months the team had locked in first-design-win on the next two carrier deployments. The product was not technically superior. It had been co-developed against the carrier's actual rack, software stack, and operational procedures while the competitor was still translating an MRD.

Connection to the literature.

Toyota's keiretsu supplier system, studied extensively by Liker [12], formalizes co-development between OEMs and tier-1 suppliers in the same pattern lateralworks observes among fast technology teams: engineers embedded across organizational lines, shared roadmaps, joint problem-solving as the default rather than the escalation path. Toyota consistently develops vehicles in roughly half the calendar time of comparable Western OEMs [13], and tier-1 co-development is one of the structural reasons.

2. Translate requirements; reconfirm throughout

Summary. Voice of the customer (VOC) is a thread through the whole TTM cycle, from gestation to field support, not a front-end phase. Fast teams build a VOC schedule alongside the development schedule and design specific customer involvement into every phase. Assumptions are reconfirmed continuously instead of frozen at the front end and re-discovered at beta.

Normal organization	Best organization
Customer input is gathered at the front end and then again at beta, with very little in between. The quality of what reaches the team depends on the relationship between marketing and engineering: if it is good, specs are reasonable; if it is bad, specs are wrong and only get discovered in late testing. Beta produces a list of surprises that force a re-design under deadline pressure.	VOC is engineered into every TTM phase: planning (counsel on strategy), gestation (skunk-works with leading customers), requirements (user groups and QFD), design (partner labs and direct contact with designers), development (use-ability testing), beta (early production install), field (root-cause analysis with customer), overrun (mutual problem-solving). Beta produces almost no surprises because the surprises were caught at the phase that could absorb them.

From a lateralworks engagement

A storage company was averaging 30 to 40 "beta surprises" per major release: customer-reported issues engineering had no idea existed until beta hardware reached the site. The team installed a VOC schedule with seven phase-specific touchpoints, ran every touchpoint with the same three lead customers, and converted them into engineering-owned commitments rather than marketing-owned briefings. The next major release went to beta with three reported issues, two of which were already in the team's known-issue tracker. The rework savings alone covered three years of the customer-engagement program.

3. Know what customers value, when they want it, and what they will pay for

Summary. Fast teams separate customer requirements (the problem) from product specifications (the solution). They run a Teach-Negotiate-Tell (TNT) discipline: the customer teaches the team how the product fits into a business workflow; the team negotiates with the customer on "how much is enough"; the team tells the customer what it will build and when. Skipping the teach phase produces a spec the team can build and the customer will not pay for.

Normal organization	Best organization
The team goes straight to specs without ever sorting out what the customer actually values. "Customers don't know what they want" is the operating belief. When customer feedback does arrive, it is usually a complaint from a large account about a feature that the team had no idea was important. Customer satisfaction data exists somewhere but is not connected to the requirements process.	The team uses TNT (Teach, Negotiate, Tell) to pull requirements from the customer in their own language and then convert them to specs in engineering language. The team can describe, in customer terms, what the product does and why the customer would pay for it. Pricing assumptions are tested early. Satisfaction data feeds back into the next round of requirements.

> **From a lateralworks engagement**
>
> A semiconductor client was preparing a new low-power processor variant for an embedded market. Marketing had set lower power consumption as the headline requirement, based on competitive benchmarks. A TNT round with three lead customers revealed that two of them would pay a 40 percent ASP premium for half the wakeup latency, and that power consumption was already "good enough." The team re-targeted the design around latency. The product shipped on schedule and captured premium pricing the original spec would have left on the table.

4. Understand customer wants vs hows; prioritize the drivers

> **Summary.** Customers describe "hows": specific features they think they want. Fast teams translate those hows back into the underlying wants they represent, then prioritize the wants. Quality Function Deployment (QFD) or a lighter-weight equivalent does this systematically. The result is a spec that maximizes satisfaction per engineering hour, instead of a spec that adds every feature mentioned in every interview.

Normal organization	Best organization
Customer requirements are skipped in favor of product specifications. The team responds to the loudest customer complaint of the week. QFD is rejected as too complex; lighter VOC tools are rejected as too soft. The spec accumulates features. Engineering builds them all. The product ships late, with everything the customer asked for and many things they do not value.	The team uses QFD or a lighter-weight equivalent to map the customer's wants to the team's hows, then weights the wants by importance. The top three or four drivers get serious engineering investment; the rest get parity-or-less treatment. The team validates the mapping by asking six lead customers to sign off that the chosen specs would be competitively superior at launch.

From a lateralworks engagement

A medical-device team had a 60-item feature backlog from 18 months of customer interviews. A QFD-style exercise compressed the backlog to 12 ranked customer wants. The top two ("easy to clean" and "works with existing chart software") accounted for two-thirds of the weighted importance. Both were near the bottom of the marketing priority list, which had been organized around novel features marketing thought it could promote. Re-cutting the spec around the top two wants reduced release scope by roughly 18 months of engineering work and left the team with a product that changed clinical workflows on the customer side.

Connection to the literature.

QFD originated at Mitsubishi's Kobe shipyard in 1972 and was popularized in the West by Hauser and Clausing in their 1988 HBR article "The House of Quality" [14]. Their data showed that companies using QFD reported 30 to 50 percent reductions in engineering changes, design cycle time, and start-up costs. The lateralworks observation across hundreds of engagements is consistent: the QFD discipline matters more than any specific QFD tool.

5. Start design while refining specs; never freeze

Summary. Specifications and design are interdependent. Freezing specs at the front end and forbidding change is a large source of TTM overrun, because the world changes during development and the eventual re-plan costs more than the discipline saved. Fast teams use flexible containment: short-form 80-percent specs early, an active Change Control Board (CCB), and a deliberate strategy of front-loading changes when the rework penalty is lowest.

Normal organization	Best organization
Either design does not start until specs are fully defined (slow), or design starts with no specs at all (chaotic). Change control is ad hoc. Specs are eventually "frozen" by mandate, the world changes, the freeze breaks, and a major re-plan resets a new "really frozen" set of specs. The cycle repeats, accounting for a substantial fraction of total TTM overrun.	Specs are short-form (one page) and cover the 80 percent that engineering can act on; design starts immediately. A standing CCB governs incoming changes. Early changes are encouraged (low rework penalty); late changes are stopped or absorbed via contingency. Total change volume is minimized. Adaptation speed is maximized: the team responds quickly to customer or competitive moves.

> **From a lateralworks engagement**
>
> A mobile-software team had a 14-week requirements lockdown phase before engineering could start. The team replaced it with a 3-week 80-percent spec and a standing CCB that met for 30 minutes every Tuesday. The first prototype was in user hands at week 4 instead of week 17. The CCB absorbed 41 change requests over the next 20 weeks; under the old process those would have either forced a re-plan or been deferred. The product shipped 11 weeks ahead of the original schedule with the same scope.

6. Fail fast; integrate early; many do-it-try-it-fix-it cycles

Summary. Fast teams integrate early and often, even when individual components are not yet finished. The integration surfaces system-level problems that unit testing cannot find. Integration failures are treated as learning events, with many short cycles budgeted into the schedule instead of one big cycle at the end. Roughly right beats exactly wrong.

Normal organization	Best organization
The phase-gate waterfall (EVT → DVT → PVT) drives behavior. Each function makes its part "perfect" and hands off to the next. Integration happens at the end. System problems surface there, with no schedule buffer left to absorb them. The team falls back through the gates, redoing earlier work. Failure is treated as blame; people protect their pieces and avoid early integration.	Concurrent engineering. Unit work, integration, and system test run in parallel and overlap. The team integrates as early as possible, often with simulated parts where physical parts are not yet ready, and integrates frequently. Each integration cycle is short. Failures are treated as information about the system. Learning cycles are explicitly budgeted into the schedule.

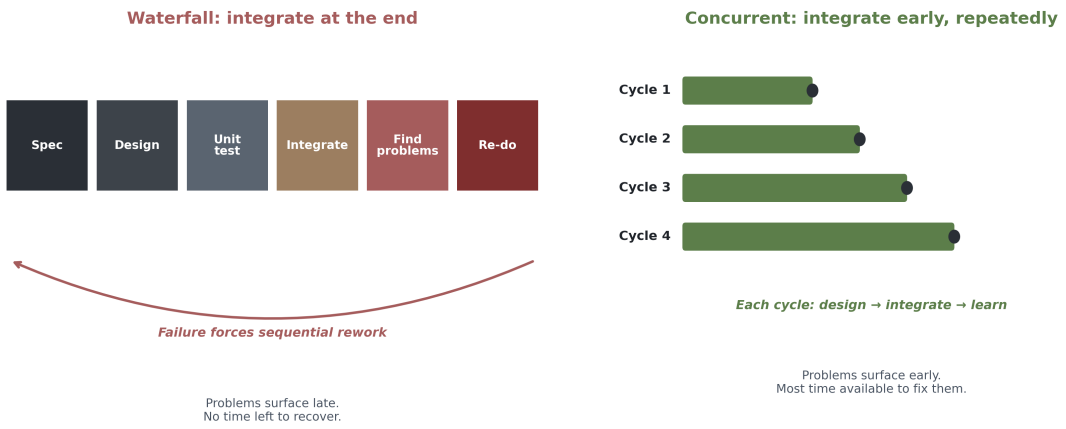

Figure 3. Waterfall integration loads all the system risk onto the end of the project. Concurrent integration spreads it across many short cycles, each of which surfaces problems while there is still time to respond.

From a lateralworks engagement

An industrial-automation client ran a strict EVT → DVT → PVT process for a new motion-controller board. Each phase took 8 to 14 weeks, and roughly one in three boards looped back to EVT after PVT discovered an integration problem. The team moved to concurrent design-of-experiments, running four parallel DOEs that each tested a specific integration risk in the first 6 weeks of the project. Two of the four DOEs surfaced problems that would previously have been caught at PVT and triggered a loop-back. Catching them at week 6 avoided roughly 22 weeks of rework and shipped the controller 42 percent faster than the previous generation.

Connection to the literature.

Reinertsen [2] formalizes this as the economics of fast feedback loops: the cost of finding a problem rises by roughly an order of magnitude for each phase the discovery is delayed. Ries [8] and the lean-startup tradition extend the same logic to whole product concepts. The deeper root is Toyota's *jidoka* principle [1]: make problems visible immediately, stop the line until they are understood, and resume only when the system has learned. Fail fast is jidoka applied to engineering, rebranded for the software era.

03

Right-team

Build the team that can deliver

Two practices that turn a roster of names into a team that can deliver. Right-team is the smallest of the three groups, but the highest-leverage: a team organized incorrectly will under-perform every other practice in this paper. Both practices are about composition and structure. A motivated team in the wrong structure will burn out compensating for its own design.

7. Core team organized by ROLES, not functions; lateralized

Summary. A fast core team is 5 to 7 people, each carrying a program-level role (chief engineer, system architect, customer integrator, manufacturing lead) in addition to their functional title. The role bias is horizontal: each person owns a slice of the whole product. The functional bias is vertical: each person owns a piece of their discipline. Right-team structure makes the horizontal bias dominate. Without it, the team becomes a coordination meeting between functional silos.

Normal organization	Best organization
The "core team" is whoever needed to be invited so they would not complain about being left out. It has 20 to 30 part-time members, each loyal to a function. Meetings degrade into rat-hole technical debates won by the loudest SME. There is no system architect, no customer integrator, no chief engineer; technical trade-offs default to the most senior person in whichever function is currently complaining. Nobody owns the product as a whole. Failures are blamed on the next function downstream.	The core team is 5 to 7 dedicated people, each carrying a clearly named program role: chief engineer, system architect, system integrator, customer integrator, manufacturing lead, test lead, quality lead. Functional sub-teams report into the relevant role. The role bias subordinates the function bias. The system architect, in particular, owns the trade-offs nobody else has authority to make, and is accountable for product-level coherence.

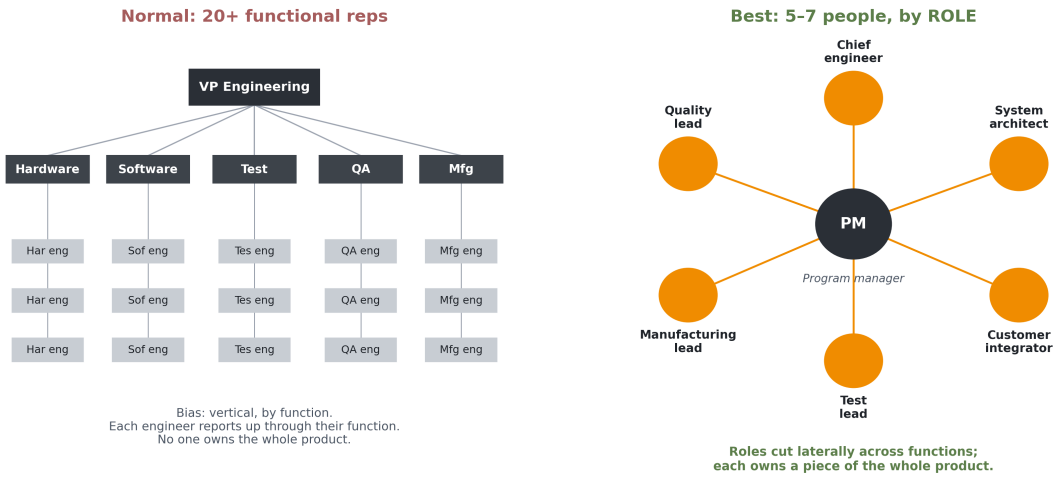

Figure 4. The shape of a function-organized team versus a role-organized team. The role-organized team has a center of gravity (the program manager) and a small ring of role-owners; sub-teams hang off the role-owners, not the core team.

From a lateralworks engagement

A datacenter-hardware client ran its flagship platform through a 26-person "core team" that had grown over four releases. Meetings ran four hours. Decisions deferred to follow-up meetings. Two prior systems-level integration failures had each been blamed-and-forgotten with nobody holding the architectural baton. The team reorganized into a 7-person core (PM, chief engineer, system architect, customer integrator, hardware lead, software lead, manufacturing lead) with the rest moved into role-aligned sub-teams. The next platform shipped 5 months earlier. In the team's words: "finally felt like one product instead of seven."

Connection to the literature.

Wheelwright and Clark [11] introduced the four-quadrant team taxonomy (functional, lightweight, heavyweight, autonomous) and documented a factor-of-three time-to-market advantage for heavyweight teams over lightweight ones in their study of automotive and electronics OEMs. Toyota's *shusa* or chief engineer system, described by Morgan and Liker [12], is the canonical heavyweight role: a single individual owns vehicle-program integration end-to-end with authority that cuts across the functional hierarchy. The lateralworks role-not-function discipline generalizes the chief engineer model into a small ring of named program roles.

8. Get the best people; mix of inside and external SMEs

Summary. Fast teams secure the best technical talent for the project, whether it sits inside the company or outside. Internal employees, contractors, consultants, and supplier engineers all work as one virtual team on one mission. Resource constraints inside the company are real, but they are not a reason to staff the project with the wrong people.

Normal organization	Best organization
The team is staffed with whoever was free that month. Key skills are missing or borrowed at 8 hours per week from someone with another priority. External SMEs are rejected on principle ("IP leakage," "only we know our domain") even when the internal expertise does not exist. The best people are split across 5 to 10 projects and are most absent at the start of the new one, when their input matters most. "Lack of the right people" becomes the all-purpose explanation for slip.	The team is composed deliberately, with the right skills on day one. External SMEs are recruited when needed; NDAs handle the IP concern. The boundary between employees and external resources is invisible from the team's perspective. Everyone is on the same virtual team, accountable to the same outcomes. The portfolio runs with enough discipline that the right internal people are actually available when the project needs them.

From a lateralworks engagement

A semiconductor client preparing a high-speed memory interface needed deep signal-integrity expertise it did not have internally. The functional manager's instinct was to assign whoever was available and have them ramp. The PM instead brought in two specialist consultants for a 6-month engagement, gave them the same access, schedule visibility, and decision authority as internal team members, and put their reviews on the chief engineer's desk rather than the consultants' agency. The two consultants caught two eye-diagram problems in the first month that, by the team's own estimate, would have surfaced in qualification 6 months later and forced a respin. The fee for the engagement was less than 5 percent of one respin.

Connection to the literature.

Parker [31] documented the same pattern in cross-functional team research: external members, treated as full members rather than visiting consultants, consistently outperform pure-internal teams on novel-domain projects. Reinertsen [2] frames the economics: the cost of an external specialist is small relative to the cost of delay caused by absence of expertise. Organizations that optimize headcount budget over delivered product almost always pay the difference, with interest, in slip.

04

Right-time
Plan and drive the schedule

Seven practices that turn a schedule from a reporting artifact into the primary tool the team uses every day. Fast teams treat their schedule the way a startup treats its cash runway: a real number that drives real decisions, refreshed continuously, owned by everyone, honest about the gap between what is wanted and what is achievable. Normal teams treat their schedule as a presentation deliverable for the next executive review.

9. Schedule as driver, not reporter

Summary. Fast teams use the schedule to drive day-to-day decisions: what to work on, what to defer, what to escalate, what to pull in. Normal teams use the schedule to report what already happened. A driving schedule prevents slip; a reporting schedule documents it. Making the schedule king does not require sacrificing quality or cost. It requires using the schedule to find and remove the wasted time that exists in every project.

Normal organization	Best organization
Schedules are produced for management consumption. Inside the team, they are out of date within weeks and are used neither to set daily priorities nor to decide trade-offs. Schedule slip is treated as inevitable; "we can't sacrifice quality for time" is the universal rationalization. Re-plans happen at major reviews, after the fact, and slide the date out by months at a time.	The schedule is the team's primary planning tool. It is up to date, honest, and used every day. Top management can articulate the cost of delay in dollars, which makes "pay to save a day" the default trade-off. Slip is treated as a signal of waste to hunt down and remove, not as inevitability to accommodate. The team challenges itself to find and eliminate slack.

From a lateralworks engagement

A networking-equipment client's schedule was traditionally regenerated the Friday before each monthly executive review. Between reviews, engineers worked from email threads, ad-hoc spreadsheets, and the chief engineer's memory. The team adopted a weekly schedule scrub on Wednesday mornings (60 minutes, full core team) and made the resulting schedule the only authoritative source for what was on the critical path. Within two months the team was identifying and recovering one to two weeks of slip per month before it accumulated. The product shipped 9 weeks ahead of target, the first on-schedule release the division had produced in 4 years.

10. Aggressive planning

Summary. Fast teams spend roughly 3x the planning time of normal teams at the start of a project, then refresh the plan every week throughout. The investment looks expensive in calendar days, but it pays back many times over in execution. Aggressive planning surfaces the schedule gap early, when it can be closed. Passive planning surfaces it late, when it can only be reported.

Normal organization	Best organization
Schedules are top-down, manufactured to fit the date the team was given, and never seriously challenged. Individuals who push back are labeled non-team-players. The schedule is high-level, mirrors the functional structure, and is rarely used to manage daily work. Heroics and eleventh-hour firefighting are rewarded; the planning that would have prevented the fire is not.	Plans are built bottom-up by the team using critical-path methods, structured around customer-need dates rather than internal capacity, and reconciled to the target date with explicit gap analysis. If the gap is too large to close, the project is killed before it starts. If it goes ahead, the plan is refreshed weekly with actuals, and the team looks continuously for ways to pull in the critical path before it slips.

> **From a lateralworks engagement**
>
> An enterprise-software client adopted aggressive planning across three concurrent projects. Two passed gap analysis and went forward; one was killed at week 2 because the gap exceeded what any reasonable acceleration could close. Killing it freed the people for the two that survived. Both shipped on time, in roughly half the calendar duration of the previous comparable project. The killed project, in retrospect, would have consumed 9 to 12 months of effort and shipped 6 months past its market window. Aggressive planning saved the cost of building it.

11. Macro-micro planning; rolling window

Summary. Fast teams plan the whole project at the macro level (around 50 tasks, end-to-end, with 4 to 6 cross-functional integration points) and break down only the next 6 weeks into micro detail (1- to 15-day tasks). The micro window rolls forward weekly. Detail is added where there is clarity; macro placeholders carry the rest. This avoids two opposite failure modes: too little detail to manage near-term work, and too much detail in the far term where it will all change anyway.

Normal organization	Best organization
Either the schedule has no detail (just milestones), or the project manager spends weeks producing a fully decomposed plan that is out of date the day it is published. The detailed plan is too complex for anyone but the PM to read, so it functions as a paper deliverable rather than a tool for managing work. When the project changes scope, the plan either does not change or has to be rebuilt from scratch.	A 1-to-2-day offsite produces a 50-task macro plan covering the entire project, with explicit cross-functional integration points. The next 6 weeks are broken down into 1-to-15-day micro tasks. The window rolls forward weekly: as time advances, additional micro tasks are pulled out of the macro placeholders. The team has detail where it has clarity, and a coherent end-to-end view in the abstract.

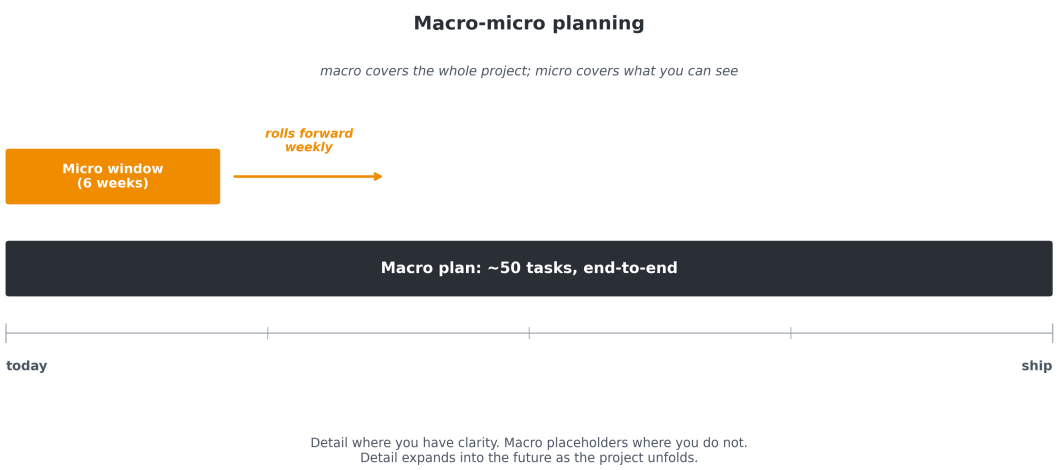

Figure 5. Macro-micro planning. The macro plan covers the full project; the micro window covers what the team can see clearly, and rolls forward as the project advances.

> **From a lateralworks engagement**
>
> A memory-controller qualification program had repeatedly tried to build a fully decomposed 18-month detailed schedule and abandoned it within a quarter each time, because the schedule was unmaintainable. The team adopted macro-micro: a 50-task macro plan covering the full 18 months, with 6-week rolling micro detail. The macro plan caught two front-loaded integration points the previous detailed plan had buried. The rolling micro window kept the team focused on what it could act on. The program completed in 13 months, and the planning effort fell from roughly 0.4 of a PM FTE to 0.15.

12. Team planning, simulation, scrubbing, ownership

Summary. Fast teams build and maintain their own plan, with the project manager as facilitator. Continuous scrubbing (small groups reviewing detailed sections of the plan against actuals) keeps the schedule honest and generates ownership. The team can simulate alternative paths together and choose with full information. The PM never edits the schedule alone.

Normal organization	Best organization
The PM creates the plan in isolation, then "sells" it to the team in a review meeting. Members are disengaged because they did not create it. The schedule slips out of sync with reality within weeks. The PM spends increasing time on cosmetic updates for management while the team tracks its real work in spreadsheets, email, and meetings. Team members do not own the dates because they were never asked to commit to them.	The full team builds the plan together, scrubs it in small groups, and treats it as their own document. The PM facilitates. The team uses the schedule daily to set priorities, predict events, and identify cross-team dependencies. Refresh meetings take 30 to 60 minutes weekly. Ownership is high. The schedule is honest because the people doing the work are the people maintaining it.

> **From a lateralworks engagement**
>
> An industrial-controls client had a heroic PM who personally maintained the full project schedule across three releases. When she went on leave, the schedule deteriorated within four weeks. The team adopted shared ownership: a 60-minute Tuesday refresh with the full core team, plus 30-minute scrubbing meetings with sub-teams as needed. The PM stopped editing the schedule alone. The schedule stayed accurate through the next two releases regardless of who was on vacation. Team confidence in the predicted dates rose noticeably.

13. Market-driven, not resource-constrained

> **Summary.** Fast teams plan first without resource constraints. They establish what the market or the lead customer needs (the target date), what work is required to get there, and how long that work takes. Then they evaluate the resource and budget gap and decide explicitly how to close it: add people, outsource, rescope, or kill. Normal teams plan from the resources they happen to have and force-fit the schedule, hiding the gap and pushing it into the future.

Normal organization	Best organization
Planning starts inside the constraint: "You have 12 months, $5M, and 15 people; show me a plan that fits." The team produces a plan that appears to fit, knowing privately that it will not. The project finishes at $7M, 18 months, and 30 people. The gap was real on day one but was not surfaced because the planning method hid it. The schedule, the budget, and the resource plan are all systematically optimistic.	Planning ignores constraints initially, except the target date set by the market or lead customer. The team determines the work needed, estimates duration honestly, and then computes the resource and cost implications. The gap (if any) is explicit and becomes the focus of an executive trade-off conversation: invest more, outsource, rescope, or kill. The decision is made with information, not against the wall.

From a lateralworks engagement

A storage client planned its next-generation array assuming the existing engineering headcount. The unconstrained plan revealed a 40 percent gap on a 14-month target. The executive conversation that followed went somewhere the constrained plan would have prevented: the team outsourced two non-differentiating subsystems to a long-time supplier, freed 6 senior engineers to work the differentiated path, and shipped the array on the original 14-month target. The constrained plan would have showed apparent fit at 14 months, then missed by 8.

Connection to the literature.

The Lockheed Skunk Works under Kelly Johnson is the canonical example of market-driven planning: the SR-71 Blackbird program defined the performance and schedule first, then assembled the resources required, rather than asking what could be built with available staffing. Reinertsen [2] reaches the same conclusion analytically: starting from the resource constraint optimizes utilization at the cost of throughput, and throughput determines time-to-market.

14. Know the gap; drive before-the-fact urgency

Summary. The schedule gap is the difference between when a project is wanted (target) and when it will actually finish (honest schedule). Fast teams compute this gap on day one and use it to drive urgency while there is still time to respond. Normal teams hide the gap with happy schedules and discover it late, when the only response is a major re-plan and a slip announcement.

Normal organization	Best organization
Happy schedules are the cultural norm. Showing a schedule that finishes later than the target is career-limiting. The team produces a plan that fits the target, says so on every status update, and only acknowledges slip when a major deliverable visibly fails. The slip is rationalized as "unforeseeable." The cycle repeats project after project. Senior managers treat the chronic slip as a team performance problem rather than as evidence that the schedules they accept are fictional.	Target and schedule are separated explicitly. The gap (if any) is calculated and shown to leadership on day one. Senior management treats exposure of a gap as good behavior. The team uses the gap to drive specific actions: reduce scope, run things in parallel, outsource, ship a minimum viable version first. The earlier the gap is known, the cheaper it is to close.

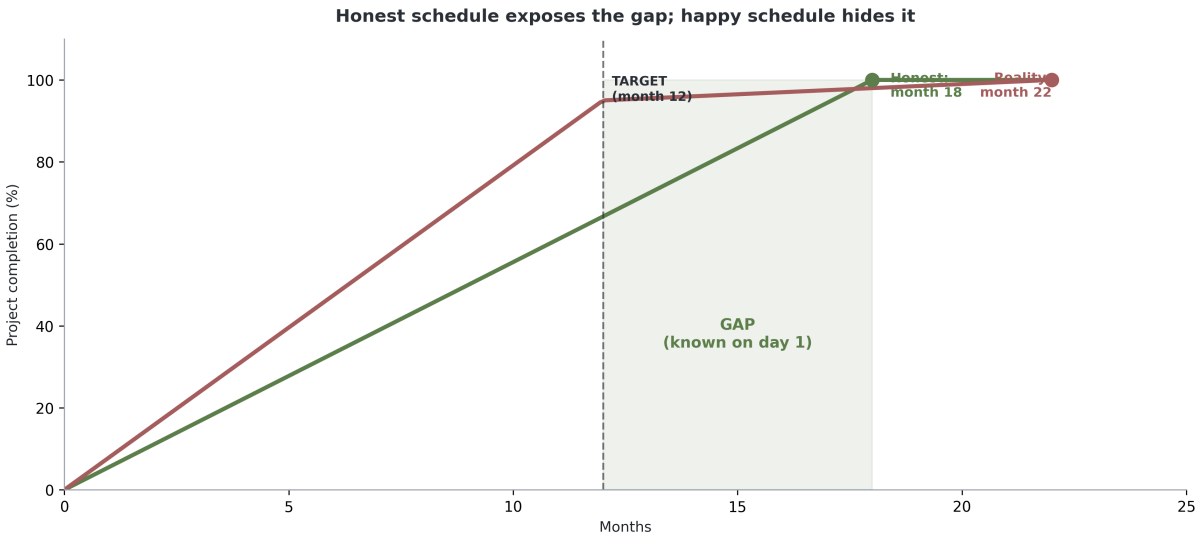

Figure 6. The honest schedule shows the gap on day one and gives the team time to respond. The happy schedule hides it until late, when the only available response is to slip further.

> **From a lateralworks engagement**
>
> A semiconductor client's team built its first macro plan on the second day of the project and found a 6-month gap on a 12-month target. Under the old culture this would have been suppressed. Under the new gap-aware culture it triggered an immediate conversation with the lead customer. The outcome was a two-step release: a minimum-viable variant on the original 12-month target (covering the customer's mandatory feature set) and a full-feature variant 4 months later. Both shipped on time. The customer preferred the staggered release. The team preferred the honest plan. The executive sponsor started asking other teams why they had not surfaced gaps as early.

15. Refresh planning; continuous schedule pull-in

Summary. Fast teams refresh their schedule weekly, sometimes daily. They update with actual progress and actively look for ways to pull in the critical path. When the schedule is on time, they accelerate to bank time against future unknowns. The discipline tracks the longest path and the cluster of paths behind it, because tomorrow's critical path is hiding in today's second-longest.

Normal organization	Best organization
The schedule is built once, presented to the team as theirs, and drifts out of sync within weeks. The PM updates it cosmetically before executive reviews. The team uses spreadsheets and email to track real work. There is no systematic pull-in attempt, only damage control after slip. Secondary critical paths are ignored on the assumption they will stay shorter than the primary.	The team refreshes the schedule weekly in a 30-to-60-minute core-team meeting, updating actuals and re-estimating remaining work. It models pull-in scenarios and acts on them. It tracks 10 to 20 paths behind the critical path on the assumption that any one of them can become the critical path tomorrow.

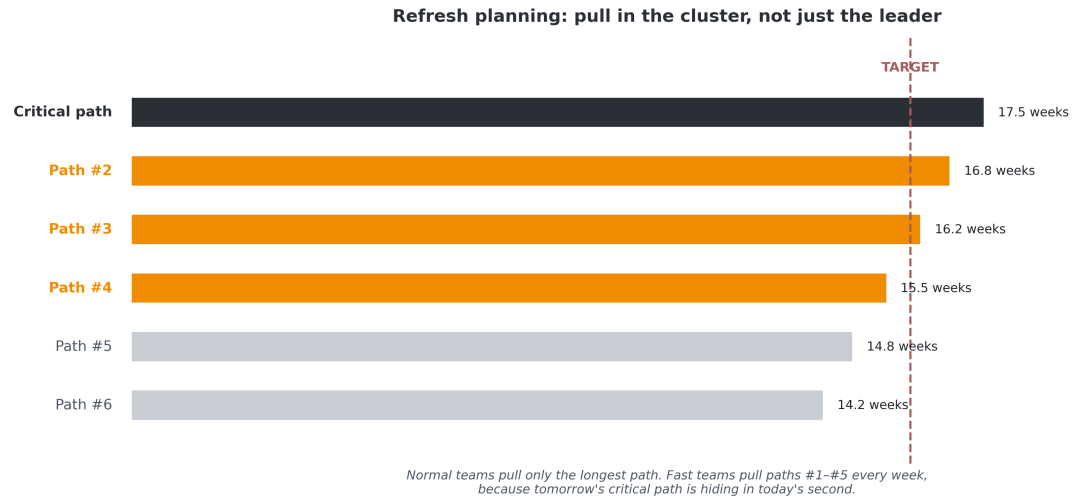

Figure 7. The critical path is not alone. Fast teams pull in the cluster of paths behind it, because the next critical path is one of those paths.

From a lateralworks engagement

A memory-products client had a strong project manager who pulled the critical path aggressively but ignored the rest of the schedule. Six weeks before tape-out, a path that had been the eighth-longest at project start surged forward as a verification subsystem ran into a corner-case issue and became the new critical path. No acceleration plan was ready. The team slipped 4 weeks. Subsequent projects adopted the discipline of pulling the top 15 paths weekly. Two later projects survived comparable late-emerging paths without slip, because each had been actively scrubbed and accelerated all along.

Connection to the literature.

Goldratt's Critical Chain Project Management [27] argues the same principle from a different angle: managing only the critical path leaves the rest of the network exposed, and a slip on any feeding path can instantly become the new critical path. The lateralworks pull-in discipline operationalizes this in a weekly cadence rather than as a separate methodology, which is why teams adopt it more readily.

05

Implementation

Putting it to work

The 15 team practices are a system. Adopting one in isolation produces a temporary improvement at best; the surrounding practices pull the new behavior back to the prior norm within a quarter or two. The teams that lateralworks has seen sustain 30 to 50 percent reductions in time-to-market adopt the practices in a deliberate sequence, with executive sponsorship from a host willing to provision and protect them through the early months when the new behaviors are still fragile.

What follows is the typical adoption sequence on a team transformation engagement. The order is not rigid. A team with strong customer engagement but a broken schedule would reorder steps 1 and 4. The sequence as written works for most teams that come to lateralworks with a "we ship late" diagnosis.

1. Reorganize the core team by role

Get the team structure right before anything else. Reduce the core team to 5 to 7 people. Assign program-level role names: chief engineer, system architect, customer integrator, manufacturing lead, and so on. Move performance review of core team members from the functional managers to the program manager. Without this step, the rest of the practices have no team to land on.

2. Build the macro plan as a team

Run a 1-to-2-day off-site to produce the macro plan. Surface the schedule gap on day one. If the gap exceeds what reasonable acceleration can close, kill the project before it consumes resources. If the gap is closeable, agree on the acceleration strategy (concurrency, outsourcing, scope reduction) before any committed dates leave the room.

3. Install weekly refresh planning

Schedule a 60-minute weekly refresh meeting with the full core team. Make it the only forum where the schedule is edited. Track the top 15 paths. Within four weeks the team will be identifying and recovering one to two weeks of slip per month before it accumulates.

4. Open direct customer channels

Identify one or two tier-1 customers and establish engineer-to-engineer channels. Run a TNT-style requirements pull. Start a VOC schedule alongside the development schedule, with named touchpoints in every TTM phase. Within 90 days the team will know what the customer actually values, and the spec will reflect it.

5. Move to concurrent integration

Replace the EVT → DVT → PVT waterfall with overlapped phases and early system integration. Budget explicit learning cycles into the schedule. Treat each integration failure as information about the system, not as a defect to assign blame for. Update the development framework with the new model so the next project inherits the practice instead of reinventing it.

6. Hand the schedule to the team

Move the schedule from a PM-owned artifact to a team-owned tool. The PM stops editing it alone. Sub-teams scrub their own sections. The schedule becomes the team's primary planning instrument, not a management presentation. This is the longest-lead change because it is cultural; allow two to three quarters for it to settle.

Key finding. Across lateralworks engagements, teams that complete this six-step sequence inside a provisioning host achieve 30 to 50 percent reductions in time-to-market on the projects run under the new system, within 12 to 18 months. Teams that attempt the sequence inside an interrupting host see at best a 10 to 15 percent improvement that fades within a year. Fast team practices and fast host practices reinforce each other. Neither survives long without the other.

Sources

References

[1] Ohno, T. *Toyota Production System: Beyond Large-Scale Production.* Productivity Press, 1988.

[2] Reinertsen, D. G. *The Principles of Product Development Flow: Second Generation Lean Product Development.* Celeritas Publishing, 2009.

[3] Christensen, C. M. *The Innovator's Dilemma: When New Technologies Cause Great Firms to Fail.* Harvard Business Review Press, 1997.

[4] Smith, P. G., and Reinertsen, D. G. *Developing Products in Half the Time: New Rules, New Tools.* 2nd ed. Wiley, 1998.

[5] House, C. H., and Price, R. L. "The Return Map: Tracking Product Teams." *Harvard Business Review,* January–February 1991, pp. 92–101.

[6] Cooper, R. G. *Winning at New Products: Creating Value Through Innovation.* 5th ed. Basic Books, 2017.

[7] Khurana, A., and Rosenthal, S. R. "Integrating the Fuzzy Front End of New Product Development." *MIT Sloan Management Review,* Winter 1997.

[8] Ries, E. *The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses.* Crown Business, 2011.

[9] Edmondson, A. C. *The Fearless Organization: Creating Psychological Safety in the Workplace for Learning, Innovation, and Growth.* Wiley, 2018.

[10] Womack, J. P., and Jones, D. T. *Lean Thinking: Banish Waste and Create Wealth in Your Corporation.* 2nd ed. Free Press, 2003.

[11] Wheelwright, S. C., and Clark, K. B. *Revolutionizing Product Development: Quantum Leaps in Speed, Efficiency, and Quality.* Free Press, 1992.

[12] Morgan, J. M., and Liker, J. K. *The Toyota Product Development System: Integrating People, Process, and Technology.* Productivity Press, 2006.

[13] Liker, J. K. *The Toyota Way: 14 Management Principles from the World's Greatest Manufacturer.* McGraw-Hill, 2004.

[14] Hauser, J. R., and Clausing, D. "The House of Quality." *Harvard Business Review,* May–June 1988, pp. 63–73.

[15] Akao, Y. *Quality Function Deployment: Integrating Customer Requirements into Product Design.* Productivity Press, 1990.

[16] Cooper, R. G., and Edgett, S. J. *Lean, Rapid, and Profitable New Product Development.* Product Development Institute, 2005.

[17] Hamel, G., and Zanini, M. *Humanocracy: Creating Organizations as Amazing as the People Inside Them.* Harvard Business Review Press, 2020.

[18] Spear, S., and Bowen, H. K. "Decoding the DNA of the Toyota Production System." *Harvard Business Review,* September–October 1999.

[19] De Smet, A., Jost, G., and Weiss, L. "Three Keys to Faster, Better Decisions." *McKinsey Quarterly,* May 2019.

[20] Bezos, J. "1997 Letter to Shareholders" and subsequent annual letters. Amazon Investor Relations.

[21] Koen, P. et al. "Providing Clarity and a Common Language to the Fuzzy Front End." *Research-Technology Management,* March–April 2001.

[22] Cooper, R. G., Edgett, S. J., and Kleinschmidt, E. J. "Portfolio Management for New Product Development: Results of an Industry Practices Study." *R&D Management,* 31(4), 2001.

[23] Rubinstein, J. S., Meyer, D. E., and Evans, J. E. "Executive Control of Cognitive Processes in Task Switching." *Journal of Experimental Psychology: Human Perception and Performance,* Vol. 27, No. 4, 2001, pp. 763–797.

[24] DeMarco, T. *Slack: Getting Past Burnout, Busywork, and the Myth of Total Efficiency.* Broadway Books, 2002.

[25] Little, J. D. C. "A Proof for the Queuing Formula: L = λW." *Operations Research,* 9(3), 1961, pp. 383–387.

[26] Kaplan, R. S., and Norton, D. P. *The Balanced Scorecard: Translating Strategy into Action.* Harvard Business School Press, 1996.

[27] Goldratt, E. M. *Critical Chain.* North River Press, 1997.

[28] Beck, K. et al. "Manifesto for Agile Software Development." 2001. https://agilemanifesto.org

[29] Sutherland, J., and Schwaber, K. *The Scrum Guide.* 2020. https://scrumguides.org

[30] Oncken, W., Jr., and Wass, D. L. "Management Time: Who's Got the Monkey?" *Harvard Business Review,* November–December 1974 (reprinted 1999).

[31] Parker, G. M. *Cross-Functional Teams: Working with Allies, Enemies, and Other Strangers.* 2nd ed. Jossey-Bass, 2002.

[32] Adler, P. S., Mandelbaum, A., Nguyen, V., and Schwerer, E. "Getting the Most Out of Your Product Development Process." *Harvard Business Review,* March–April 1996.

[33] Clark, K. B., and Fujimoto, T. *Product Development Performance: Strategy, Organization, and Management in the World Auto Industry.* Harvard Business School Press, 1991.

[34] lateralworks. "FTTM New Product Development Best Practices." Internal methodology document, Revision 2018.002.

[35] lateralworks. "Internal assessment database." Engagement data across client programs, 1992–2026.