Whitepaper

# Common problems in advanced technology programs

## Four failure patterns we keep seeing — and why traditional project management cannot solve them.

**FTTM methodology series.**

A field-tested look at the four organizational patterns that derail advanced technology programs, and the FTTM practices that resolve them. Drawn from 36 years of lateralworks engagements and grounded in published product-development research.


**Prepared by**
lateralworks
FTTM methodology

**Date**
May 2026
Synthesis paper

**Online**
lateralworks.com
Common problems series

Table of contents

# Common problems in advanced technology programs



**Core thesis.** Advanced technology programs do not fail because the work is too hard. They fail because four organizational patterns — distributed sites, multiple companies, multiple corporate stakeholders, and simultaneous innovation with product development — overwhelm conventional project management. FTTM is the operating system built for these conditions.

Overview

# Abstract

**Most advanced technology programs miss their dates for the same handful of reasons.** After 36 years and more than 200 engagements at lateralworks — from Sony's first PlayStation to GlobalFoundries' $7B Fab8 in Malta, NY — four problem patterns appear again and again. They are structural, not motivational. Smart teams hit them. Disciplined teams hit them. The patterns are baked into how modern programs are organized: across sites, across companies, across stakeholders, and across the boundary between research and product development.

This paper names the four patterns, lays out the characteristics by which we recognize each one in a program, and pairs each pattern with the FTTM practices that resolve it. The framing is deliberately concrete: the figures that follow are the same diagnostic tables we use inside engagements. A program leader should be able to read the left column of any figure and recognize their own team in it.

The second half of the paper makes a sharper claim. FTTM is not a flavor of project management. The two practices look superficially similar — both involve schedules, milestones, and cross-functional teams — but they operate from opposite premises. Project management exists to control variance against a plan. FTTM exists to compress the time between learning and decision. The distinction matters because programs that look like they need better project management almost always need something else: a different operating model that treats time as an economic asset and treats teams as system integrators rather than units of production.

The findings are grounded in lateralworks engagement data and cross-referenced with the published literature on heavyweight teams [1], product-development flow [2], and the communication-distance research of Tom Allen [3]. Where our observations refine or extend the published work, we note it. Where they agree, we cite the source so the reader can verify.

# 01

Section

# The four problem patterns

Four organizational patterns explain most of the schedule slip we see in advanced technology programs. They are not failure modes in the moral sense — no one inside the program causes them. They are environmental.

The patterns compound. A program with one of them is hard. A program with three or four is the kind of program lateralworks tends to get called into. The figures in this section are the diagnostic we use to spot each pattern and the FTTM practices we use to resolve it. They are not exhaustive — the characteristics column is the smell test, not a checklist — but they are specific enough that a program leader reading them should recognize their own situation within the first two or three rows.

A note on the solutions. The right-hand column of each figure names FTTM practices by their canonical labels: **core team**, **refresh planning**, **cost of delay**, **integrated schedule**, **learning milestones**. These are not generic project-management terms. Each one carries a specific meaning inside FTTM and a specific implementation. The point of the figure is to map the problem to the practice — the practice itself is covered in the FTTM methodology documentation.

Pattern one

# Multiple sites

**Distance kills communication faster than anyone expects.** Tom Allen's research at MIT, conducted in the late 1970s, showed that the frequency of technical communication between two engineers drops sharply as the physical distance between their desks increases. The effect levels off around 50 meters — beyond which the communication rate is effectively flat at a low value, no matter how far apart the engineers actually sit [3]. The intuition that "they're only one time zone away" or "we have video conferencing now" does not survive contact with the data. The Allen Curve has been re-validated repeatedly in the four decades since [4].

For programs that span multiple sites, this is the structural fact. Every other characteristic listed in Figure 1 is a downstream consequence. Compartmentalized workflow follows from low cross-site communication. Sub-optimized technical solutions follow from compartmentalized workflow. Isolated remote teams follow from a default of asynchronous messaging rather than synchronous interaction. The FTTM response is to fight the consequences directly with practices that force the kind of communication distance would otherwise prevent.

The single most consequential FTTM practice for distributed programs is the **core team** — Wheelwright and Clark called the same idea the *heavyweight team* in their 1992 California Management Review paper [1]. A core team is small, empowered, and accountable for the integrated outcome from concept through volume production. Members are drawn from every site and every function involved in the program. They rotate physically through each site on a regular cadence — which is what the Allen Curve says has to happen, because synchronous in-person communication cannot be replaced by tools, only supplemented by them.

## Multiple Sites

| Characteristics | Solutions |
|---|---|
| Team members distributed across many locations/time zones | Core Team; empowered, heavyweight, owns integrated "product" outcome, i.e. from concept to volume production |
| Lack convenient regular meeting times | Core Team rotation; all core team members travel to each location at least once a month |
| Disconnected and compartmentalized work flow | The integrated schedule; score card for each team/location with integration points clearly defined and managed (virtual project) |
| Lack continuity and ability of sub-teams to visualize and own the overall development project | Refresh Planning; regular and consistent core team meeting time and accountability - use of automation to collect/process progress updates |
| Lack of overall ownership of the integrated product outcomes | Include suppliers and key development partners; they are integrated into a single lateral schedule, blurs lines between internal and external resources |
| Lack of a systems vantage point, causes sub-optimized technical solutions (at each site), making downstream integration problematic | Key technical resources visit each site on a scheduled rotation |
| Poor, inconsistent communications, causing remote teams to feel isolated and disconnected | Use electronic communication tools for asynchronous communications, not as the primary communications mode |
| Over-reliance on electronic communication tools (tends to drive isolation and lack of confrontation) | Clearly defined self-contained work packages for each team/site with clear interfaces between teams/sites defined |

***Figure 1.*** *Characteristic problems on multi-site programs and the FTTM practices that resolve them. lateralworks engagement data, 2026.*

Two further practices anchor the figure. The **integrated schedule** is a single program-wide schedule rather than a collection of site schedules stitched at the seams. Each site sees its own work, but every dependency between sites is named, dated, and owned by a core-team member who is responsible for the handoff. **Refresh planning** is the cadence at which the integrated schedule is updated — typically weekly — with a structured ritual that surfaces gaps early rather than at quarterly reviews. Refresh planning is also where the program acknowledges what it has learned in the last seven days, which is the unit of time over which distributed teams forget what they agreed to last.

A common counterargument runs: "we have invested in modern collaboration tools, so the Allen Curve does not apply to us." The data does not support this. Allen returned to the question with Gunter Henn in 2006 [4] and concluded that physical proximity remained the dominant predictor of technical communication, even in organizations with mature email and video-conferencing practice. Tools supplement co-location. They do not replace it.

Pattern two

# Multiple companies

**When two or more companies share a program, their incentives almost never line up cleanly.** Each company answers to its own executives, its own shareholders, its own quarter. The program is one thing among many on each company's agenda. The team members assigned to it may or may not have the authority to commit their company's resources, may or may not have the same motivation as their counterparts across the table, and almost certainly do not share a definition of success.

The classic failure mode is asymmetric resourcing: Company A under-commits, Company B is forced to absorb the gap, and by the time anyone admits this out loud the schedule has already slipped a quarter. Reinertsen's work on product-development flow names a parallel pattern — economic factors invisible to one party in a partnership become invisible to the program as a whole [2]. The FTTM antidote is to make those economics legible.

## Multiple Companies

| Characteristics | Solutions |
|---|---|
| Each company has an independent agenda and priorities pulling them in different directions | Empowered heavyweight core team with clear roles and responsibilities defined; where all the functions are represented from each company/site |
| Team members assigned to project are not empowered to make decisions/commit resources | Upfront commitment/contract for each member's participation and contribution |
| Team members lack the same motivation and incentives | A single overall program owner with authority to make decisions and commit resources |
| Team members focused on research outcomes rather than product commercialization | A program budget with accountability metrics defined |
| Company A does not commit sufficient resources, forcing Company B to make up the difference | Cost-of-delay modeled and used in trade-off decision-making, and to express the value of time |
| Slow and unclear decision-making; decisions forced through each company's hierarchy which is slow and produces variable results | Decision-making system; roles in each decision are defined and a structured process for decision-making is followed |
| Lack of a clear understanding of the economics of delay across member companies | Decision-making cycle time is tracked and reflected in the schedule, so each member can see the impact their decision making has on the program |
| Unbalanced or unequal division of responsibility/labor, or unclear success criteria | Success criteria for the program defined that includes the contribution of each team member |

***Figure 2.*** *Multi-company program characteristics and FTTM resolutions. Drawn from lateralworks consortium and joint-development engagements.*

Three FTTM practices carry the weight here. First, an **empowered core team** with each company represented and with explicit pre-engagement commitment from each member's home executive on the scope and duration of their participation. Without that commitment, the core team is a standing meeting, not an operating body. Second, a **single program owner** — one person, not a steering committee — with the authority to commit resources and break ties. Steering committees are valuable for oversight and intractable

disputes; they are slow at everything else.

Third, and most under-used in our experience, is a **cost-of-delay model**. Reinertsen's claim — "if you only quantify one thing, quantify the cost of delay" [2] — applies with double force in multi-company programs. When each company can see, in dollars per week, what its own decision-making latency is costing the joint program, the conversations about resourcing get short and serious. House and Price made an early version of the same argument in their 1991 HBR paper on the return map [5], which tied product-team performance to break-even time. The economics of delay sit at the center of both frameworks.

Pattern three

# Multiple corporate stakeholders

**The "multiple stakeholders" pattern is the corporate-internal cousin of the multi-company pattern.** The stakeholders all work for the same parent company, but they answer to different business units, different P&Ls;, and different annual plans. They each believe they have a legitimate say in product requirements. They are often right. The result is a program that has to be everything for everyone — which means it is nothing for anyone.

This pattern is particularly common in semiconductor and platform programs, where a single chip or platform feeds multiple downstream product lines. GlobalFoundries Fab8 in Malta, NY, where lateralworks joined the initial five-person planning team, was a textbook example. First silicon shipped two weeks ahead of the target date because the core team had explicit authority to make trade-offs that no individual business unit could have made unilaterally.

**Multiple Corporate Stakeholders**

| Characteristics | Solutions |
|---|---|
| Unclear motivations; from very motivated to distracted and disinterested | A "Contract" between the stakeholders, like a joint venture project in the construction industry where each stakeholder's role is clear |
| Lots of "cooks in the kitchen" and unclear who is in charge | A single empowered core team with each stakeholder represented; the core team is empowered to make decisions, spend money, chart their own course |
| A lack of clear product requirements | A VOC process to determine market and customer requirements; continuously refined through customer council's providing iterative feedback |
| Conflicting product requirements; customer use-cases, market segments, etc. | VOC process owned by the core team |
| Lacking a clear product definition; product must be everything for everyone | Executive Steering Committee made up of most senior executive from each stakeholder; tie breaker when core team can't decide |
| Strategic shifts are common during the project; causing confusion and delay | Clear target windows set and trends tracked with early warning given through Core Team to Steering Committee |
| Unclear targets; market windows, success criteria, budget and investment (each stakeholder has their own version) | |
| | |

***Figure 3.*** *Multi-stakeholder program characteristics and FTTM resolutions. Drawn from semiconductor and platform engagements where one program feeds several business units.*

The decisive FTTM practice for this pattern is a clear "contract" between stakeholders — modeled, deliberately, on the construction-industry joint venture, where the role and contribution of each party is written down before the foundation is poured. Once the contract is in place, a **voice-of-customer** process owned by the core team produces the requirements set, and an **executive steering committee** made up of the most senior executive from each stakeholder serves as the tie-breaker for the small number of decisions the core team genuinely cannot make.

The point of the steering committee is to be used rarely. A steering committee that meets every week to settle disputes is a sign that the core team is not empowered. A steering committee that meets quarterly and mostly approves what the core team has already decided is the working configuration. The 1992 Wheelwright and Clark framework for heavyweight teams is explicit on this point [1]: the core team's authority is real, or the team itself is theater.

Pattern four

# Simultaneous innovation and product development

**The fourth pattern is the one teams resist most, because admitting it means admitting that the program is doing two incompatible jobs at once.** Innovation is the discovery of something new. Product development is the commercialization of something known. The two activities have opposite economics. Innovation rewards rapid cycles of learning and tolerates failure as the cost of the next insight. Product development rewards steady maturation toward a known target and treats failure as schedule risk.

When a program tries to do both at once — and most advanced technology programs do — the two activities interfere with each other. Engineers who should be running learning cycles are pulled into commercialization reviews. Engineers who should be hardening a known design are asked to explore three alternative architectures because someone in research wanted to know what they would look like. A related tension at the firm level — between exploiting existing capabilities and exploring new ones — has been documented in the broader innovation literature [6]. Inside a single program, the same conflict shows up as scope creep, missed dates, and a demoralized team.

## Simultaneous technology innovation and product development

| Characteristics | Solutions |
|---|---|
| Mindset that innovation can't be managed, it just happens when it happens | Isolate the "innovation" that is needed; then dedicate colocated team of experts to work on it, insulated from daily interrupts |
| Technology and product development are not synchronized, i.e. technology lags behind when it is needed in the product (delay) | Define the project such that less than 10% requires innovation/breakthrough thinking (best practice projects have 5-10% new things, rest is known) |
| Assumption that most of the work requires "innovation" | Technology roadmap planning, separated from development projects with clear points and times for intersection, i.e. asynchronous streams |
| Resources doing development are also required to innovate | Liberal use external subject matter experts, because time to market is more valuable than IP leakage & outside ideas can open minds to new thinking |
| Lack of dedicated resources to the program; not enough subject matter expertise internally, unwilling to seek them outside company | Cannibalize your own products; the hedge against IP leakage (your competition is always 1-2 steps behind) |
| Fear of IP leakage drives inward thinking, lack of solution finding because of people are thinking the same way (group think) | Clearly define what is required to commercialize a product quickly and unburden development teams with research requirements |
| Poor strategic planning of the technology roadmap; forces product development teams to do their own Research while developing | Manage innovation using Learning Milestones (vs product performance milestone); plan learning cycles and use to improve predictability |
| Development projects over burdened with requirement to commercialize while also innovating | Innovation = faster cycles-of-learning-increase frequency of learn/fail<br>Development = faster performance maturity-increase rate of product maturity |

***Figure 4.*** *Characteristics of programs that conflate innovation with product development, and the FTTM practices that separate the two streams without disconnecting them.*

The FTTM response is to isolate the innovation cleanly. Best-practice programs in our engagement data have five to ten percent of total scope allocated to genuine innovation; the rest is the execution of known engineering. When the innovation portion is much larger than that, the program is either mis-staged (research should run ahead of development on a roadmap, not in parallel inside the same project) or mis-scoped. The remedy is a **technology roadmap** that runs as an asynchronous stream with explicit handoff points into the product-development stream, plus a dedicated, co-located innovation cell that is insulated from the daily interrupts of product execution.

A second remedy is the use of **learning milestones** rather than product-performance milestones for the innovation work. A product milestone asks: did the design hit the spec? A learning milestone asks: what did we discover, and what does it imply for the next cycle? The two metrics are not interchangeable. Reinertsen makes a similar argument for queueing and feedback in product-development flow [2]: optimizing for throughput is the wrong target when the activity is fundamentally one of discovery.

# 02

Section

## FTTM is not project management

The four patterns share one thing: traditional project management is not built to handle them. Project management optimizes for control. Control assumes the work is known and that variance is the enemy. In advanced technology programs, the work is partly unknown and variance is the source of every insight worth having.

This section makes the distinction precise. It is not a takedown of project management — project management works very well for the kind of work it was designed for. It is a positioning argument: FTTM and project management are different operating systems built for different operating conditions, and using one when the conditions call for the other is the most common root cause of slow programs.

The figures in this section come from how we explain FTTM to executives in the first hour of a new engagement. They are the operating principles a program needs to internalize before any of the practices in Section 01 will stick.

## Operating principles

# Conditions required to win

**Programs at the bleeding edge of a technology are not "normal" programs running at higher speed.** They are a different kind of work, with different rules. The list in Figure 5 is what we tell program leaders on day one. It is intentionally blunt. Each line is a constraint that the rest of the engagement is built around.

A few of the conditions warrant particular emphasis. "Be willing to lose in order to win" is not a slogan — it is the operational truth that shipping a product fast almost always means shipping it with known shortcomings, and improving it over subsequent releases. The original iPod is the canonical case. Tony Fadell, who had led the Philips Velo project that was built using FTTM, applied the same practice at Apple when he created the iPod, iPhone, and iPad. The first iPod was not the perfect iPod. It was the iPod that shipped, and the perfect iPod arrived in subsequent generations.

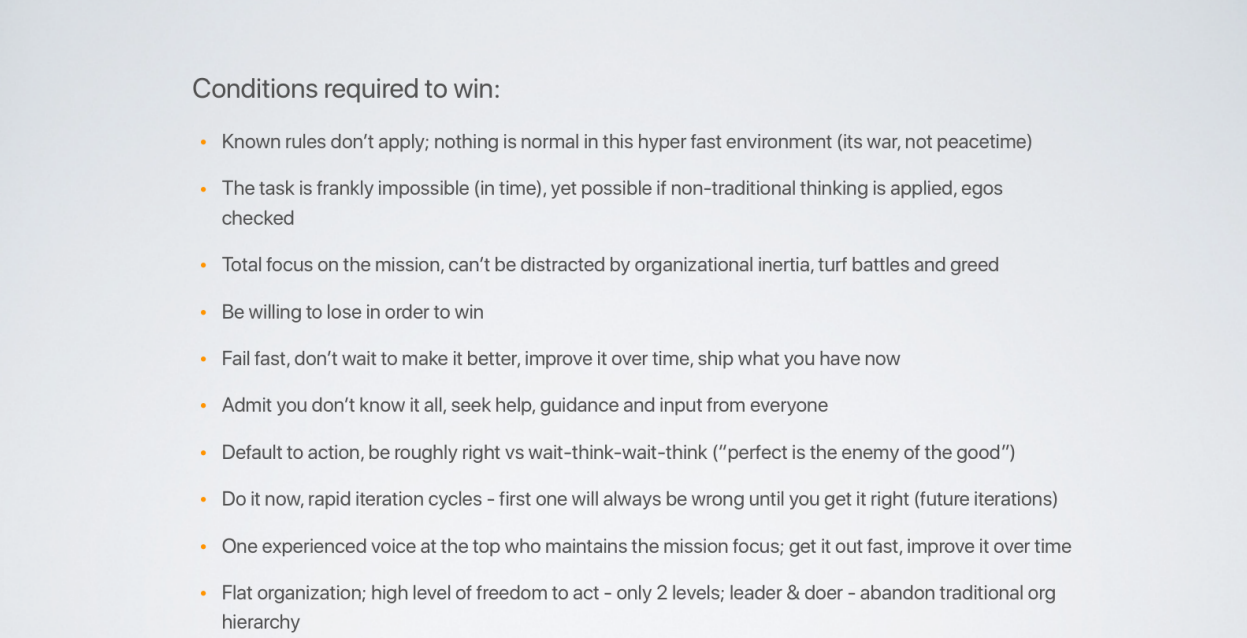


***Figure 5.*** *The conditions a program has to operate under to win at speed. These are constraints, not aspirations — programs that violate them slow down regardless of how hard their teams work.*

"Flat organization; only two levels — leader and doer" is the most organizationally radical condition on the list. It is also the one that most directly maps to the Wheelwright and Clark heavyweight-team model [1]. A heavyweight team has a single leader with real authority and a small number of empowered doers. Layers of management between the leader and the doers slow decisions by definition, and on a fast program the cost of those slowed decisions compounds. The cost-of-delay framework [2] makes that compounding visible in dollars; the practice of flat organization removes the compounding at its source.

Positioning

# How FTTM differs from project management

**The temptation, on first encounter with FTTM, is to file it as a variant of project management.** It looks like project management. It uses schedules, milestones, status reviews, and cross-functional teams. The first hour of an FTTM engagement is often spent dismantling this assumption, because the two practices operate from opposite premises and the misunderstanding tends to ruin the engagement if it is left in place.

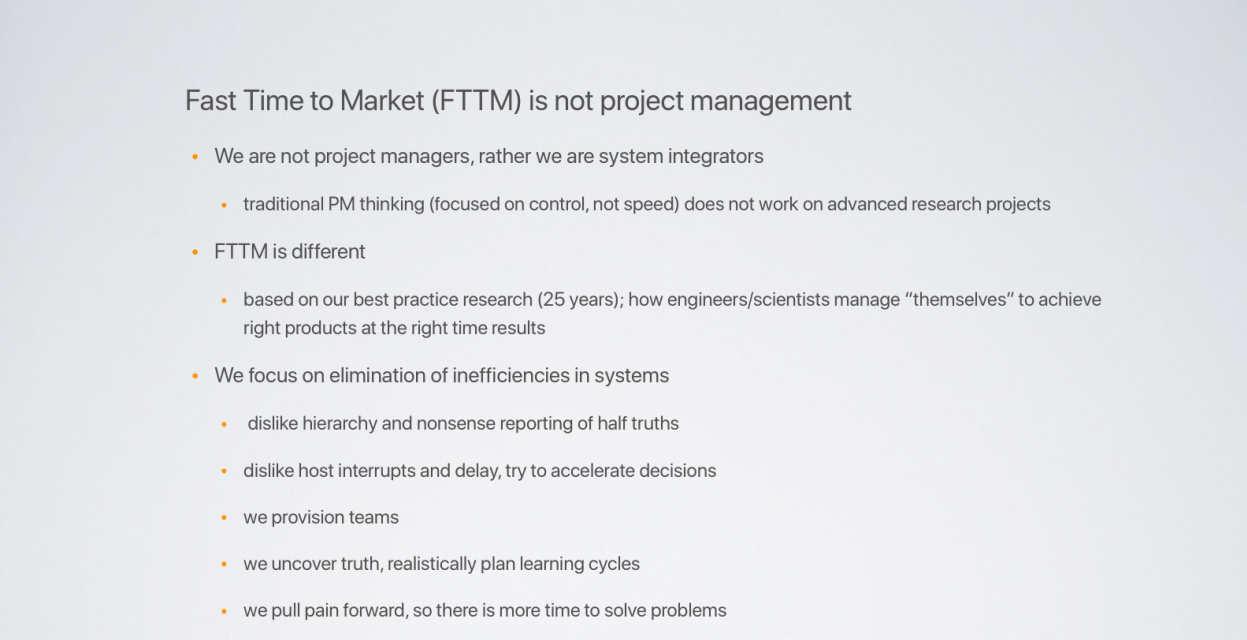


***Figure 6.*** *The core distinction. lateralworks engages as system integrators rather than as project managers. The practice is built on 36 years of research into how teams that ship on time actually work — not on a theory of how they should.*

A few points from Figure 6 are worth amplifying. **"Eliminate inefficiencies in systems."** Traditional project management focuses on controlling the project; FTTM focuses on the system the project sits inside — the meetings, the approvals, the cross-site handoffs, the reporting chains that consume calendar time without producing learning. Most of the time we recover on a program is recovered by removing system-level friction, not by working faster.

**"Pull pain forward."** A traditional schedule is optimistic by construction: it shows the work succeeding the first time, with risks noted as caveats. An FTTM schedule is the opposite. We model the real schedule by including the rework, the dependency latency, the second and third design spins. The gap between the optimistic plan and the realistic model is the conversation we want to have with leadership in month one, not

in month nine. The integrated schedule turns this gap into a quantifiable economic question that the cost-of-delay framework can answer [2].

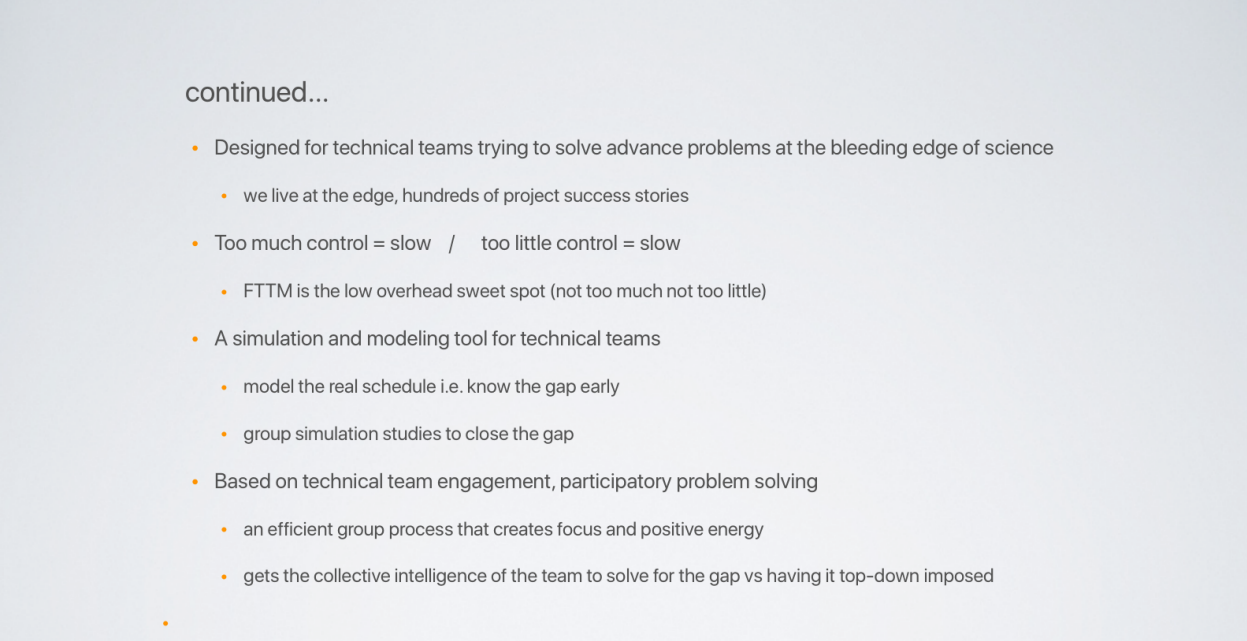


***Figure 7.*** *Continued. FTTM as a simulation-and-modeling discipline for technical teams, and as a participatory problem-solving practice that engages the collective intelligence of the program rather than imposing solutions top-down.*

**"Too much control is slow; too little control is slow."** This is the most counterintuitive line in the figure, and the one that takes longest for executives to internalize. Highly controlled programs slow down because every decision has to climb a chain of approvals. Lightly controlled programs slow down because there is no shared schedule and no shared sense of what done looks like. FTTM is the low-overhead sweet spot — enough structure to align the team, not enough to crush its judgment. Calibrating that level is most of what an engagement is for.

**"Group simulation studies."** Decisions in an FTTM program are made by the people who will execute them, working through the implications collectively, in a structured session that produces a decision the team owns. This is the practical embodiment of the heavyweight-team principle [1]: a core team that has been empowered and that has shared visibility into the integrated schedule arrives at better decisions, faster, than a hierarchical organization can. The data on this in our engagement record is unambiguous, and it is consistent with the broader literature on participatory decision-making in product development.

Sources

# References

[1] Clark, K. B., and Wheelwright, S. C. "Organizing and Leading ‘Heavyweight’ Development Teams." *California Management Review*, Vol. 34, No. 3, Spring 1992, pp. 9-28.

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

[3] Allen, T. J. *Managing the Flow of Technology: Technology Transfer and the Dissemination of Technological Information within the R&D Organization.* MIT Press, 1977.

[4] Allen, T. J., and Henn, G. *The Organization and Architecture of Innovation: Managing the Flow of Technology.* Butterworth-Heinemann, 2006.

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

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

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

[8] lateralworks. "Internal assessment database." Engagement data across client programs, 1988-2026. lateralworks.com.

[9] lateralworks. "Methodology: the FTTM framework." lateralworks.com/methodology, accessed May 2026.