Whitepaper

Systems thinkers: where did they go?

How complexity outran integration — and why technology programs still slip, overrun, and break at the seams

FTTM methodology series.

Systems engineering was invented to solve a specific organizational problem in the 1960s. The problem still exists. The roles that solved it have quietly disappeared. This paper argues that modern program failure is the predictable consequence — and that the cure is the same as it ever was.

Prepared by
lateralworks
FTTM methodology

Date
May 2026
Methodology paper

Online
lateralworks.com
FTTM best-practices series

Table of contents

Systems thinkers: where did they go?

Core thesis. The "systems problem" in modern technology programs is not a failure of engineering talent or project management software. It is the absence of two specific roles — the systems architect and the systems planner — that used to own the integrated whole. Complexity grew; the integrators disappeared. The schedule slips and integration failures we treat as unavoidable are the predictable consequence.

Overview

Abstract

Systems engineering was invented in the 1960s to solve a specific problem. How do you build a machine too complex for any one person to hold in their head, and still have it work the first time you turn it on? The answer was a discipline — and a pair of dedicated roles, the systems architect and the systems planner — that owned the whole. Those roles put a person on the moon, on schedule, in a decade, using slide rules [10, 11]. They built the first passenger jets, the first commercial semiconductors, the first globally distributed telecommunications networks.

Sixty years later the systems we build are vastly more complex. Software tangles with silicon, silicon with supply chains, supply chains with geopolitics. Meanwhile, the roles that used to hold the whole together have quietly disappeared. Technology programs run on domain specialists who each optimize their component, hand it over the wall, and assume someone else is worrying about how the pieces fit. Usually no one is. The result shows up as schedule slip, integration rework, field defects, and products that arrive late to a market that has already moved on.

The evidence is substantial. McKinsey and Oxford studied 5,400 large IT projects and found they ran 45 percent over budget and delivered 56 percent less value than promised [2]. The Standish Group has reported for two decades that only a third of software projects succeed on time and on scope, and that large projects succeed less than ten percent of the time [15]. Bent Flyvbjerg's database of more than 16,000 megaprojects across industries finds 91.5 percent of them run over budget, over schedule, or both, with a mean overrun of 62 percent and a fat tail of catastrophic failures [7, 8]. The common thread is not technology. It is the absence of an owner of the whole.

The prescription is not novel and not complicated. Restore the two roles that went missing. Give the systems architect authority over the technical whole: end state, interface contracts, cross-discipline tradeoffs. Give the systems planner authority over the schedule whole: integrated master schedule, critical path, interface milestones. Pair them, empower them, hold them accountable for outcomes no single function can deliver alone. This paper lays out what the roles do, why they vanished, and how to put them back.

01

The complexity paradox

When the systems outgrew the integrators

Start with a simple observation. The technology we build today is orders of magnitude more complex than what was built sixty years ago. A modern car carries more lines of code than the F-35 fighter jet. A smartphone integrates radios, sensors, accelerators, and operating-system layers that each would have been a standalone program in 1970. A semiconductor node launch coordinates hundreds of interface contracts across design houses, foundries, tool vendors, and equipment suppliers spread across three continents. Complexity has exploded.

Meanwhile, the organizational capacity to manage that complexity has atrophied. The dedicated systems engineering function — people whose only job is to see across the whole and make the cross-discipline decisions — has been thinned out, absorbed into project management offices, or distributed across domain leaders who already have day jobs. What used to be a named, accountable role is now a part-time responsibility split among people who don't have time for it.

Two curves diverging

The gap between what we build and what we can integrate

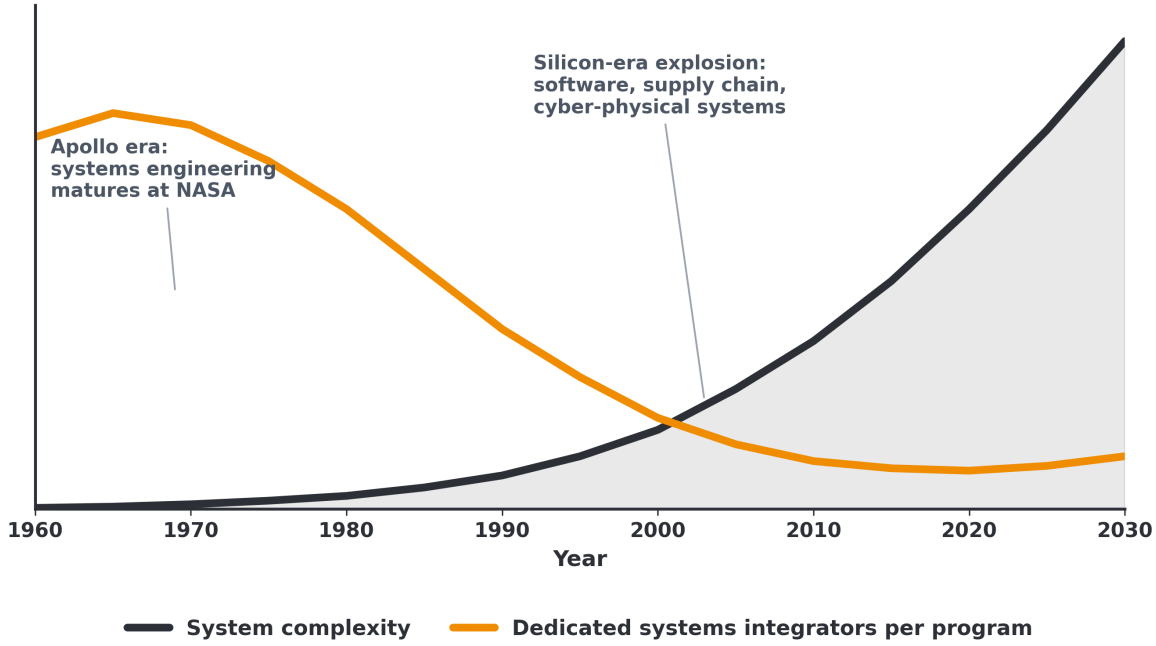

Figure 1. System complexity has grown roughly exponentially since the 1960s while the number of dedicated, named systems integrators per program has declined. The gap between what modern programs demand and what modern organizations supply is the "systems problem." Illustrative; based on INCOSE workforce surveys [9] and industry program-team data.

The gap between the two curves is the systems problem. Programs of breathtaking ambition are now launched with no one in the room whose job is to hold the whole thing together. Leadership assumes the domain leads will coordinate among themselves. Domain leads assume leadership has it covered. Both assume the integrated master schedule in the project management tool means someone is managing integration. It rarely does.

Conway's Law, formulated in 1968, predicted this. Mel Conway observed that the architecture of any system mirrors the communication structure of the organization that built it [5]. When the organization has no dedicated path for information to flow across functional boundaries, the product won't have one either. The seams in the org chart become the seams in the product. That is true even when every individual subsystem is excellent.

02

Four cognitive habits

What a systems thinker actually does

Before diagnosing where the systems thinker went, define what the role does. A systems thinker is not a generalist, not a manager, and not a project planner — though the role touches all three. Four cognitive habits define the work, and most organizations no longer cultivate any of them.

Habit one

Start from the end state, not the problem

Conventional engineering starts with a problem statement and works forward through analysis and design to a solution. Systems thinking reverses the flow. It starts with a clearly defined end state — the product in market, the mission accomplished, the customer using the system successfully — and works backward to determine what must be true, in what order, for the end state to be reached. The difference is not semantic. Starting from the solution forces clarity about success criteria before any technical decision is made.

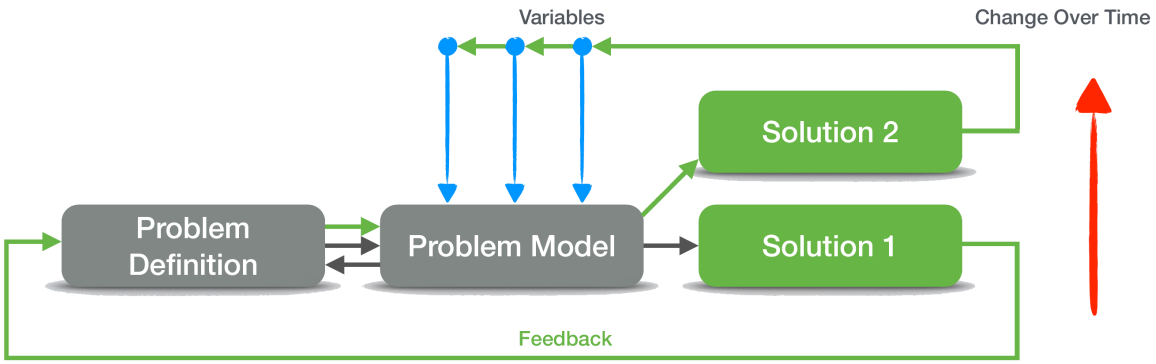

Figure 2. Systems thinkers work backward from a clearly defined end state. Conventional thinking works forward from a problem statement and often discovers too late that the problem was framed incorrectly.

Habit two

Optimize the system, not the components

A component-optimizing organization produces parts that are individually excellent and collectively incompatible. Every function tunes its deliverable against its own local metric — head performance, media yield, firmware code size, supplier cost — and no one asks whether the sum of those local optima adds up to a working product. The systems thinker trades local optimization for system optimization, and accepts that some components will be less than their best so the whole can be better than a patchwork of best-in-class parts. That was the core of George Mueller's "all-up" testing decision on Apollo: integrate the whole stack and test it as a system rather than perfect each stage in isolation [11].

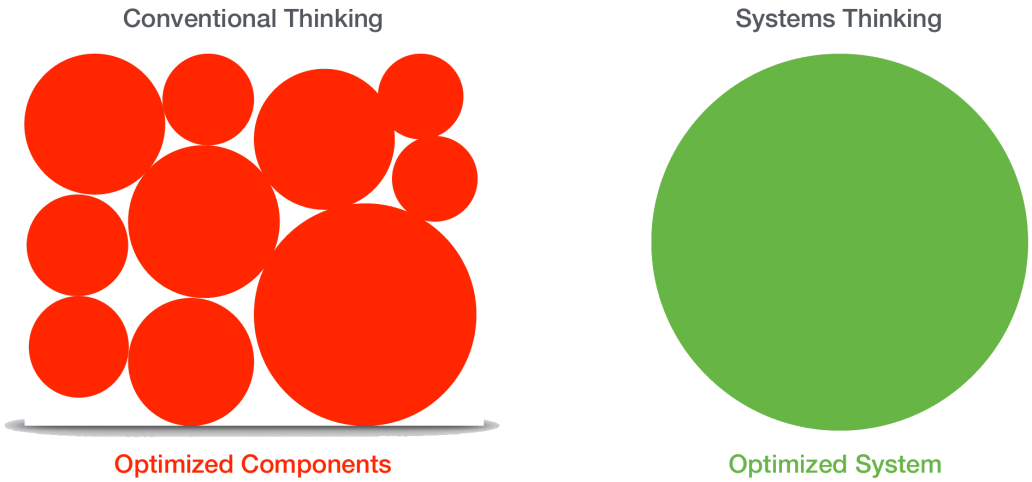

Figure 3. Component optimization (left) yields a collection of individually excellent parts that fit together badly. System optimization (right) accepts local compromise in service of a coherent whole. The two philosophies look similar on paper and produce very different products.

Habit three

Think laterally, not vertically

Most organizational structures push information vertically: engineers report to leads, leads report to directors, directors report to VPs. Information moves up and down within a functional silo. Systems thinking moves information laterally — across silos, between disciplines, around the organization chart. A decision made in the head design group affects media, which affects servo, which affects firmware, which affects test. The systems thinker is the connective tissue that carries those signals across the vertical organization.

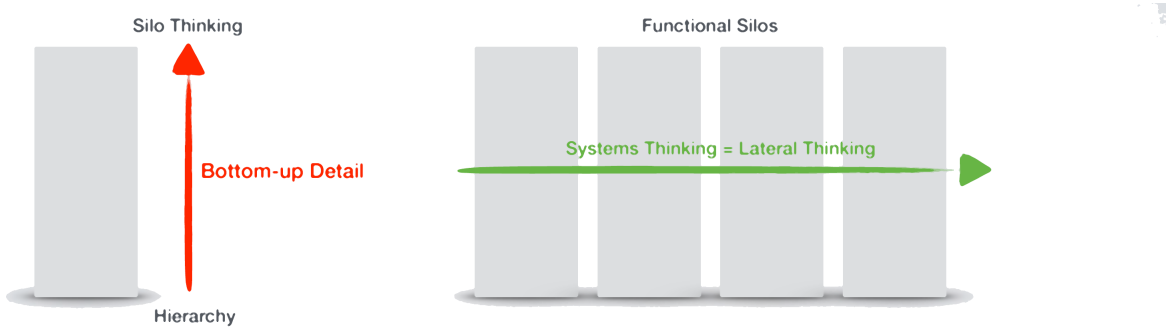

Figure 4. Silo thinking moves information up and down functional hierarchies. Systems thinking moves it laterally, across disciplines. When lateral flow is absent, decisions made in one silo impose hidden costs on every other silo — and no one sees the total until integration.

Habit four

Hold macro and micro in the same field of view

The systems thinker works at 32,000 feet and at ground level in the same hour. The same person discusses the product's market position with the CEO and the read-channel noise floor with a firmware engineer, without losing either thread. This is not generalism. It is the ability to zoom between scales without losing the connections. Charles and Ray Eames made the idea visual in their 1977 film *Powers of Ten* [6]: every scale connects to every other scale, and understanding requires movement between them.

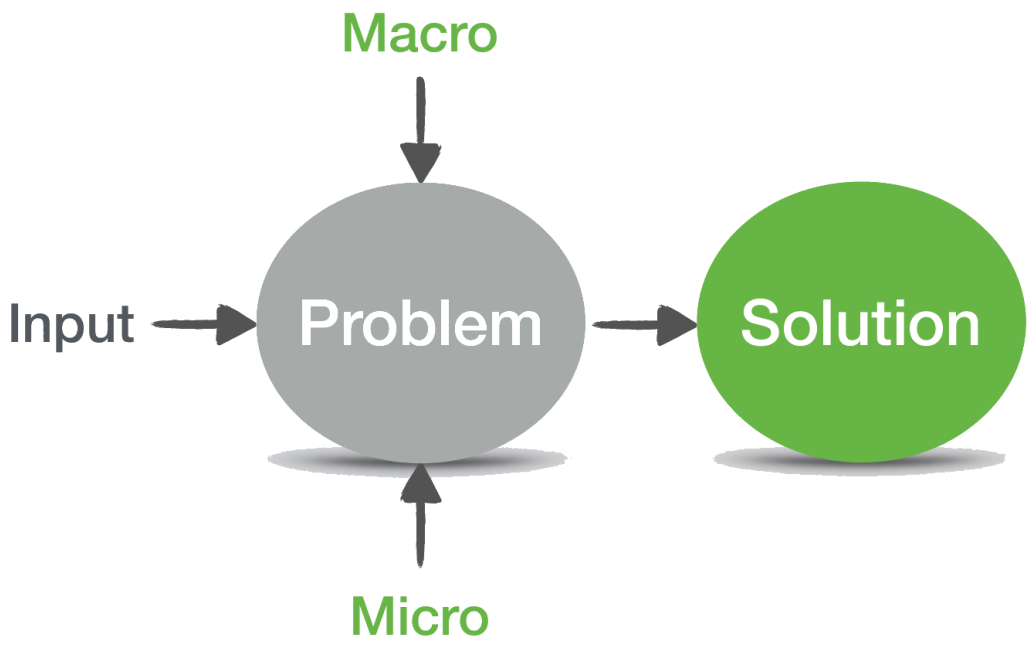

Figure 5. Macro and micro are the same system viewed at different scales. The systems thinker moves between them; the specialist gets stuck at one.

> *"It is better to be roughly right than exactly wrong."*
>
> *— John Maynard Keynes — a maxim every systems thinker eventually internalizes.*

03

The historical proof

When we had them, it worked

The cleanest proof that integrated systems thinking works is Apollo. In 1961 the United States committed to landing a person on the moon and returning them safely to Earth within a decade. Almost nothing required to do it existed: not the launch vehicle, not the spacecraft, not the guidance computer, not the tracking network, not the flight software, not the spacesuit. Most of the underlying technology had to be invented during the program itself.

Apollo

The integrator in chief

Apollo had one thing modern programs no longer have: a systems engineering and integration function with executive authority. George Mueller, running the Office of Manned Space Flight from 1963, imposed "all-up" testing and drove the organization to treat the Saturn V, the Command Module, the Lunar Module, and the ground systems as one integrated system rather than four cooperating programs [10, 11]. The decision was unpopular — Wernher von Braun argued for the safer incremental-test approach — but Mueller outranked him and the program shipped. After Apollo 11, von Braun wrote that "without all-up testing the first manned lunar landing could not have taken place as early as 1969" [11]. A dedicated Apollo-wide systems engineering team held the interface contracts, adjudicated cross-module tradeoffs, and maintained an integrated master schedule that cascaded from the lunar landing date back to molecular-level decisions about epoxy formulations for heat shields.

The 747

Twenty-eight months, four thousand engineers

The same pattern produced the Boeing 747. In 1966 Boeing agreed to deliver the first 747 to Pan Am by the end of 1969, leaving 28 months to design what would become the largest aircraft ever built — about two-thirds of the normal commercial program cycle. Joe Sutter led a 4,500-engineer team called "the Incredibles," and the airplane rolled out on schedule [17]. The integration function was Sutter himself: one chief engineer with authority to make cross-discipline tradeoffs in the room rather than referring them up a hierarchy. The pattern is repeatable. So is its absence.

> **Historical pattern.** Every large-scale technology program that finished on time and on scope had a dedicated, empowered systems integration function. Every one that didn't slipped, overran, or broke at the seams. The pattern is remarkably consistent across industries and eras.

A maxim worth keeping

On managing the whole

"Systems fail to perform properly when their managers believe some aspect of the world is outside the system, and not subject to any control."

C. West Churchman

The Systems Approach (1968)

04

Decline and consequence

Where the systems thinkers went

If the pattern works, why did it disappear? Several structural forces, operating over decades, thinned the systems engineering function down to a ceremonial role in most modern technology companies.

Decline

Four structural forces that drained the role

The rise of specialized expertise. As technology domains deepened — VLSI design, machine learning, RF engineering, cloud infrastructure — career reward structures in every major technology company shifted to reward depth over breadth. Engineers who became world-class in a single narrow domain were promoted; engineers who tried to know a little about everything were seen as having no specialty. The person whose job is to connect across domains has no natural home on a modern engineering career ladder.

The outsourcing of integration. Conway's Law operates at the corporate scale just as it does inside a single building [5]. When a company decomposes a product into subsystems assigned to separate teams, separate vendors, and separate geographies, the product itself will reflect those seams. The rise of global supply chains, contract manufacturing, and fabless semiconductor business models pushed integration across organizational boundaries. The person who used to walk across the hall to settle a cross-discipline tradeoff now negotiates it across a statement of work with a vendor in another country.

The triumph of project management over systems engineering. Project management — the discipline of tracking tasks, managing deliverables, and reporting status — is not the same thing as systems planning, but in most organizations the two have merged under a single project management office. Project management answers the question, "is everyone doing their work on time?" Systems planning answers a different question: "if everyone does their work on time, will the integrated product work?" The first question is tractable and measurable. The second requires someone who understands the technical content deeply enough to judge whether the pieces will actually fit. Most PMOs cannot answer it and most organizations no longer have anyone whose job it is to try.

The belief that tools will do the work. Over the last twenty years, enormous sums have been spent on enterprise tools — requirements management systems, model-based systems engineering platforms, integrated development environments, digital thread infrastructure — on the theory that if the right software is deployed, integration will happen automatically. It does not. Tools record decisions; they do not make them. A requirements database that no one curates becomes a museum of obsolete requirements. A model-based platform that no one uses to reason across disciplines becomes expensive drawing software. The tools are neutral. The missing piece is the person.

The cost

What integration failure actually buys you

The literature on complex program outcomes agrees on one point: failure rates are high, and the failures cluster around integration. The McKinsey-Oxford study of 5,400 large IT projects (initial budgets above $15 million) found average overruns of 45 percent on cost and 7 percent on schedule, with 56 percent less value delivered than projected; 17 percent of the projects went so badly wrong they threatened the existence of the company [2]. The Standish Group's CHAOS Report has reported for two decades that only about a third of large software projects succeed on time, on scope, and on budget; another third are challenged; the rest fail outright [15]. Flyvbjerg's research across more than 16,000 megaprojects in twenty-plus industries finds 91.5 percent run over budget, over schedule, or both, with a mean overrun of 62 percent and a fat tail of catastrophic failures — including IT projects whose tail-end overruns average 447 percent [7, 8].

These are not engineering failures in the narrow sense. The components usually work. The failure is in integration: the parts do not fit, the interfaces are wrong, the timing does not line up, the assumptions each team made were mutually inconsistent, and no one noticed until the pieces were put together. Boehm quantified the economics of that failure in 1981: a defect costs roughly 1× to fix in architecture, 10× in design, 100× in integration, and up to 1,000× once the product is in the field [1]. NIST put a national number on it in 2002 — $59.5 billion per year in U.S. software defect costs alone, more than half of it borne by users in prevention and mitigation [12].

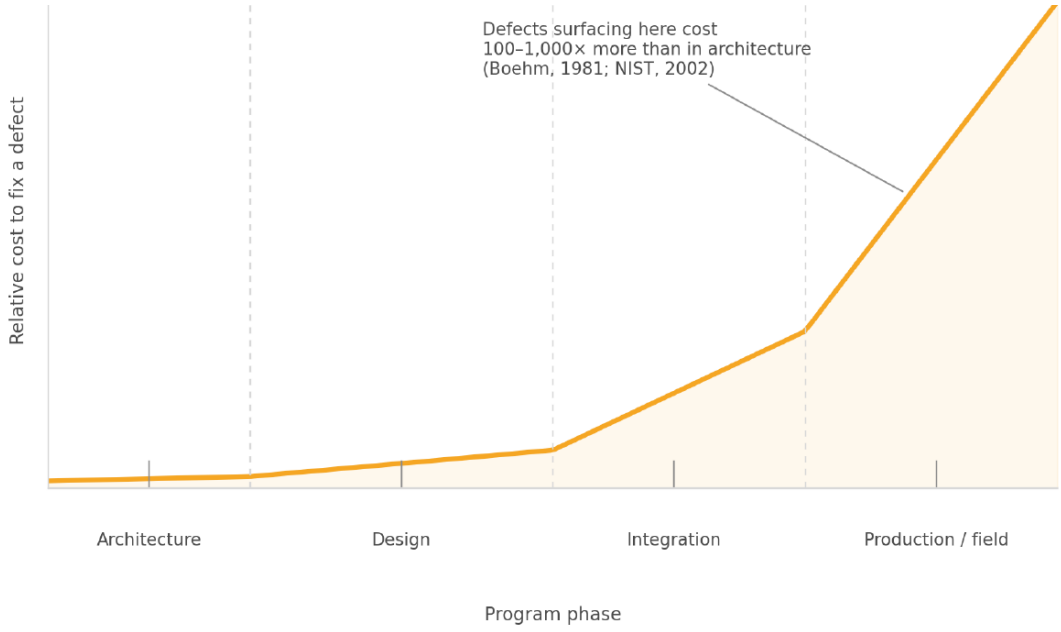

Figure 6. The cost to fix a defect rises sharply with program phase. Defects caught in architecture cost roughly 1×. By integration they cost 10–50×. In production they cost 100–1,000×. The systems architect's highest-value work is catching the defects that would otherwise surface at the right end of this curve. Data: Boehm (1981) [1]; NIST (2002) [12]; IBM Systems Sciences Institute.

Recent high-profile failures show the pattern. The Boeing 737 MAX MCAS accidents trace directly to a flight-control function architected as an isolated subsystem reliant on a single angle-of-attack sensor, with

neither cross-system failure analysis nor pilot training reflecting its authority over the airframe [18]. Major automotive OEMs have delayed electric-vehicle launches by years because battery, drive motor, thermal, and software teams each optimized their own subsystem without a single owner of the integrated vehicle. Enterprise software rollouts routinely blow past budget and schedule because the implementation, data migration, business process, and change management teams each ran their own project with no one owning the integrated cutover. The technologies differ. The mechanism does not.

> **The integration tax.** When no one owns the whole, the cost of integration is paid in schedule slip, rework, field defects, and missed market windows. The tax is always paid — either up front by investing in systems thinking, or at the back end in recovery programs, warranty claims, and lost revenue. The back-end payment is always larger.

05

Diagnosis and prescription

Restoring the two roles

Organizations rarely notice they have a systems problem until the schedule is already slipping. By then the damage is done and recovery is expensive. Leaders who want to catch the problem earlier can watch for a characteristic set of symptoms.

Symptoms

Six warning signs that the integrator is missing

- **The integrated master schedule is unsigned.** Every functional team has its own schedule. No one can point to a single document that shows how the schedules roll up to the committed customer date, or who owns it.
- **Interface defects dominate the bug list.** The defects that slow the program are not in any single subsystem; they are at the interfaces between subsystems, and each team blames the other.
- **Cross-functional decisions take weeks.** A decision that affects three or four disciplines gets scheduled as a series of one-on-one meetings because no single person has the authority to resolve it in a single room.
- **Requirements churn never stops.** Requirements are modified late in the program because no one checked them against the other disciplines' constraints until integration forced the check.
- **The schedule recovers through scope reduction.** When the program misses a gate, the recovery plan is to descope features rather than to restructure the integrated plan. Scope reduction is a sign that no one can find schedule through technical tradeoffs.
- **Executive reviews feel like status theater.** Program reviews consist of each function reporting its own status in sequence. No one presents the integrated health of the program, and when a challenge surfaces, leadership asks a question no one in the room can answer.

Any one of these symptoms, in isolation, may have another explanation. All of them together, persistently, are diagnostic. They indicate that the program has no dedicated owner of the integrated whole and is relying on hope and committee to do the job instead.

The prescription

Restore the two roles that went missing

Fixing the systems problem does not require a new methodology, a new tool, or a cultural transformation. It requires two specific roles in the program structure, with explicit authority and named accountability. The roles are paired: one owns the technical whole, the other owns the schedule whole. Neither role is new. Both have clear historical analogs in Apollo, in the 747 program, and in the INCOSE handbook of modern practice [9, 10, 11]. In most modern programs, neither is present.

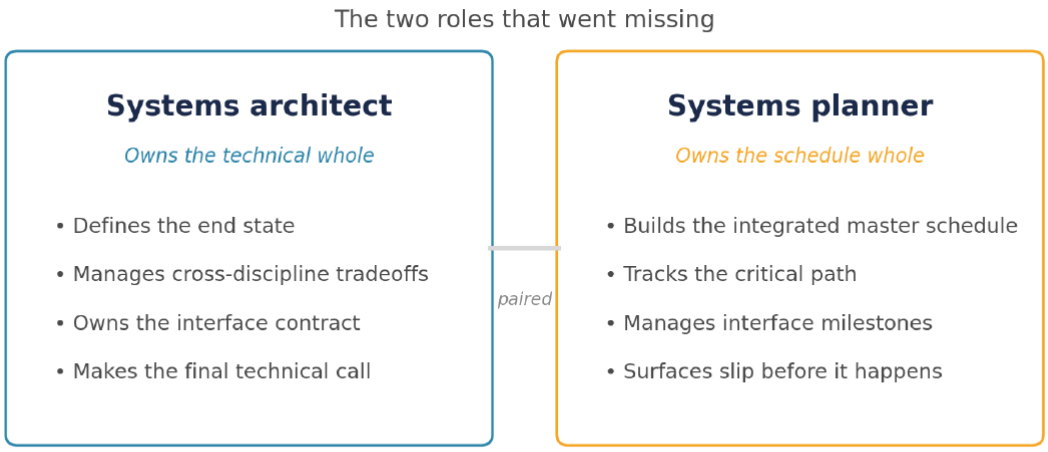

Figure 7. The two roles that went missing. The systems architect owns the technical whole — end state, interfaces, cross-discipline tradeoffs. The systems planner owns the schedule whole — integrated master schedule, critical path, interface milestones. They are distinct roles, but they must be paired. Neither works alone.

The systems architect owns the technical definition of the integrated product. That includes the end-state specification, the interface contracts between subsystems, the cross-discipline tradeoffs, and the final technical decision when disciplines disagree. The architect is not the chief engineer of a particular subsystem; the architect is chief engineer of the integration. The role requires deep enough technical credibility to earn the respect of the domain leads, broad enough vision to see across them, and explicit organizational authority to make decisions that bind the domain leads to an integrated answer.

The systems planner owns the integrated master schedule. That is a specific deliverable: a single, technically grounded schedule that connects the committed customer date back through every major program deliverable, identifies the critical path, and exposes the interface milestones between subsystems as named, tracked events. The planner surfaces schedule slip before it happens, not after. This role is frequently confused with project management; the two are not the same. A project manager tracks that people are doing their assigned work. The systems planner asks whether the integrated schedule is technically achievable and what must change if it is not.

The technical whole and the schedule whole are two views of the same underlying integrated system. A technical tradeoff has schedule implications; a schedule slip forces a technical tradeoff. Separating the two roles across organizations, as many companies do — a chief architect in engineering and a PMO lead in operations — guarantees that technical and schedule decisions are made in isolation from each other. Pairing them produces a single integrated voice that can answer both "will this work?" and "will this work on time?" in one conversation.

Pattern to restore. Name a systems architect with authority over the technical whole. Name a systems planner with authority over the schedule whole. Pair them, empower them, and hold the program accountable to their integrated view. Do this at program kickoff, not in recovery. The earlier the roles are in place, the lower the integration tax.

Normal organization	Best organization
Architecture lives inside engineering. Schedule lives inside the PMO. Neither role has executive authority. Cross-discipline tradeoffs happen in committee, by escalation, or by default when the deadline arrives. The integrated master schedule is a status-reporting artifact, not a technical instrument.	A named systems architect and a named systems planner are paired at program kickoff. Their authority over the integrated whole is written down and respected. Tradeoffs and slip are surfaced and resolved in single conversations, technically grounded, and cascaded into the schedule before they become recovery problems.

A

Closing observation

The pattern is sixty years old

There is an irony in the current state of technology programs. The discipline of systems engineering was invented, in the 1960s, specifically to solve the problem that modern programs now routinely fail to solve. The industries that pioneered the practice — aerospace, defense, large-scale civil works — institutionalized the roles and produced generations of integrated systems that worked. Much of that capability has been lost, not because it stopped working, but because the organizational memory of why it mattered faded as the people who learned it during Apollo and its successors retired [13].

A pattern sixty years old

And one we keep rediscovering the hard way

The technology industry rediscovers systems engineering one painful program at a time, usually after a failure expensive enough to make the executive team take notice. The pattern is predictable: a program slips by a year, leadership commissions a review, the review finds no one was owning the integrated whole, and the organization promises to do better next time. The discipline that would have prevented the failure already exists, is well documented, and is taught in every major engineering school [9, 16]. It is just not practiced.

The systems thinker is not a mystical figure. The role is definable, teachable, and hireable. What is required is a decision by technology leaders to restore the role, give it authority, and hold it accountable for an outcome — the integrated whole — that no domain lead can deliver alone. The programs that make that decision early will ship on time. The programs that don't will slip, overrun, and break at the seams, in the way that programs run without systems thinkers have always slipped, overrun, and broken at the seams. The pattern is sixty years old. It has never changed. It will not change now.

> **The through-line.** Complexity is not the problem. Complexity is the condition. The problem is that organizations stopped naming someone whose job is to make the complexity cohere. The host that restores the role will ship faster than the host that does not — not because its engineers are better, but because someone is actually in the room whose job is the whole.

Sources

References

[1] Boehm, B. W. *Software Engineering Economics.* Prentice-Hall, 1981. Foundational data on the cost of fixing defects across program phases.

[2] Bloch, M., Blumberg, S., and Laartz, J. "Delivering large-scale IT projects on time, on budget, and on value." *McKinsey Digital,* October 2012. Study of 5,400 large IT projects with initial budgets above $15 million showing 45 percent average cost overrun, 7 percent average schedule overrun, and 56 percent average value shortfall. https://www.mckinsey.com/capabilities/tech-and-ai/our-insights/delivering-large-scale-it-projects-on-time-on-budget-and-on-value

[3] Brooks, F. P. *The Mythical Man-Month: Essays on Software Engineering.* Addison-Wesley, 1975 (anniversary edition 1995). Classic treatment of integration complexity and the limits of adding people to late projects. Brooks also coined the name "Conway's Law."

[4] Churchman, C. W. *The Systems Approach.* Delta, 1968. Foundational text articulating systems thinking as a discipline distinct from component-level engineering.

[5] Conway, M. E. "How Do Committees Invent?" *Datamation,* April 1968. The original 1968 statement of Conway's Law: "organizations which design systems are constrained to produce designs which are copies of the communication structures of these organizations." http://www.melconway.com/Home/Committees_Paper.html

[6] Eames, C. and R. *Powers of Ten.* IBM / Eames Office, 1977. Nine-minute film essay on scale and connection, frequently cited in systems thinking pedagogy.

[7] Flyvbjerg, B. "What You Should Know About Megaprojects and Why: An Overview." *Project Management Journal,* April 2014, pp. 6-19. Cross-industry data on cost and schedule overruns in large complex programs across 16,000+ projects.

[8] Flyvbjerg, B., and Gardner, D. *How Big Things Get Done.* Currency, 2023. Modern treatment of why large projects fail and what the successful ones have in common; reports 91.5 percent of megaprojects go over budget, over schedule, or both, with mean cost overrun of 62 percent.

[9] INCOSE. *Systems Engineering Handbook: A Guide for System Life Cycle Processes and Activities.* 5th edition, edited by D. D. Walden, Wiley, 2023. ISBN 978-1-119-81429-0. Reference text for modern systems engineering practice, including the integrated systems architect role.

[10] Johnson, S. B. *The Secret of Apollo: Systems Management in American and European Space Programs.* Johns Hopkins University Press, 2002. Historical account of how NASA's systems engineering practices were developed and institutionalized under George Mueller.

[11] Slotkin, A. L. *Doing the Impossible: George E. Mueller and the Management of NASA's Human Spaceflight Program.* Springer Praxis, 2012. Detailed history of the all-up testing decision and the organizational design that delivered Apollo on schedule.

[12] NIST. *The Economic Impacts of Inadequate Infrastructure for Software Testing.* NIST Planning Report 02-3, prepared by Research Triangle Institute, May 2002. Source for the $59.5 billion annual U.S. cost of software defects and the multipliers across software lifecycle phases. https://www.nist.gov/document/report02-3pdf

[13] NASA. *NASA Systems Engineering Handbook.* NASA/SP-2016-6105 Rev 2, 2016. The canonical systems engineering reference developed from Apollo-era practice.

[14] Senge, P. M. *The Fifth Discipline: The Art and Practice of the Learning Organization.* Doubleday, 1990 (revised edition 2006). Extension of systems thinking to organizational learning.

[15] Standish Group. *CHAOS Report* (annual series, 1995–2020). Long-running study of IT project success rates; consistently reports large-project success rates below 10 percent and overall success around 30 percent in the 2020 edition.

[16] Stevens, R., Brook, P., Jackson, K., and Arnold, S. *Systems Engineering: Coping with Complexity.* Prentice-Hall, 1998. Practitioner text on managing cross-discipline technical complexity.

[17] Sutter, J., and Spenser, J. *747: Creating the World's First Jumbo Jet and Other Adventures from a Life in Aviation.* Smithsonian Books, 2006. Firsthand account of the 747 program by its chief engineer; documents the 28-month design cycle of "The Incredibles" team.

[18] U.S. House Committee on Transportation and Infrastructure. *Final Committee Report: The Design, Development and Certification of the Boeing 737 MAX.* September 2020. Root-cause analysis of the MCAS failure tracing the accidents to single-sensor architecture, inadequate cross-system safety analysis, and certification gaps.

[19] lateralworks. "FTTM new product development best practices." Revision 2018.002. lateralworks.com.

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

About lateralworks. lateralworks helps technology executives restore integrated systems thinking to complex product development programs through the Fast Time To Market (FTTM) methodology. FTTM pairs a named systems architect with a named systems planner, produces a single integrated master schedule grounded in the technical content, and makes the critical path and interface milestones visible and actionable at every level of the program.