Case study

# Recovering a worldwide security program

How lateralworks brought Cisco’s late ACS 5.0 program back onto a converging schedule and protected a multi-hundred-million-dollar hardware roadmap.

**FTTM engagement series.**

A development program run across multiple sites and continents, hundreds of engineers, eleven end-to-end teams, coordinated from a single program office in Israel — and roughly a year behind.

**Prepared by**
lateralworks
Program management

**Engagement**
Cisco ACS 5.0, 2008
Case study, 2026 edition

**Online**
lateralworks.com
FTTM engagement series

# Table of contents

## Recovering a worldwide security program



**Core thesis.** A late program rarely recovers because people start working harder. It recovers when the schedule becomes a living instrument the whole team can see and pull on. lateralworks made the gap visible, gave eleven teams ownership of their own end-to-end work, and refreshed and pulled in the plan two and three times a week until the predicted delivery date stopped drifting and started converging on the target.

Overview

# Abstract

**Cisco’s next-generation access-control platform was running about a year late.** The Secure Access Control System 5.0 (ACS) was a ground-up rebuild of the policy server that authenticates and authorizes users and devices on enterprise networks — the AAA core that sits behind RADIUS and TACACS+. The previous generation could not carry Cisco’s roadmap forward, and several switching businesses were waiting on 5.0 to ship their own dependent products. The release was late, the feature set kept moving, and the program had lost the one thing a late program needs most: a credible, shared view of when it would actually finish.

lateralworks was brought in to recover the schedule. The work was not a reorganization and not a new tool rollout. It was the disciplined installation of the lateralworks Fast Time to Market (FTTM) operating system on a live, distributed program: build one integrated macro-to-micro schedule with explicit doneness criteria, measure the real gap between the target and the honest prediction, then refresh and pull in the plan continuously with the people doing the work. [4, 5, 6]

The program ran across multiple development sites on several continents, with hundreds of engineers and eleven end-to-end teams, coordinated from a single program office in Israel. Each end-to-end team owned its slice of the product from requirements through integrated, tested delivery — a lateral, cross-functional pattern that the FTTM method was built to support. [4] Within weeks the predicted delivery date stopped sliding and began moving toward the target. The program shipped, and the dependent hardware roadmap it gated was protected.

This case study describes the starting condition, the FTTM approach as it was actually applied, the turnaround as the team experienced it, and what the program meant for Cisco both at the time and in the architecture that followed it. The named client and product are used with the engagement on the record; individual team members are referred to by role.

# 01

## The program

## What was at stake

ACS 5.0 was not a routine point release. It was the platform on which Cisco intended to build the next decade of identity-driven network access, and it was load-bearing for businesses far larger than the access-control product itself.

To a reader outside network security, the stakes are easy to miss, so it is worth stating plainly what the system does. Every time a person or a device asks to get onto an enterprise network, something has to decide whether to let them on and what they are allowed to reach. That decision engine is the access-control system — the authentication, authorization and accounting (AAA) server. It is quiet infrastructure, and it is also the gate in front of almost everything else.

Portfolio context

# A release other businesses were standing on

The previous generation, ACS 4.x, had reached the end of its usefulness for Cisco's cross-company stakeholders. The 5.0 rebuild introduced a rule-based policy model and a new administration platform, and it was the version Cisco's switching business units were depending on. Their own products were sequenced behind it, which meant a slip in ACS did not stay inside ACS — it pushed out revenue across a much larger portfolio. By Cisco's own reckoning at the time, the release sat upstream of a multi-billion-dollar business and hundreds of millions of dollars in dependent hardware sales, and it strengthened the company's position in the enterprise market. [4]

That is the right way to read the urgency. Success was never going to be measured in ACS license revenue. It would be measured in whether the businesses downstream could launch on time — the market-window logic the return-map literature formalized [9] — and in the confidence of the large internal stakeholders who were betting their own schedules on it. A late access-control release was a tax on the whole roadmap.

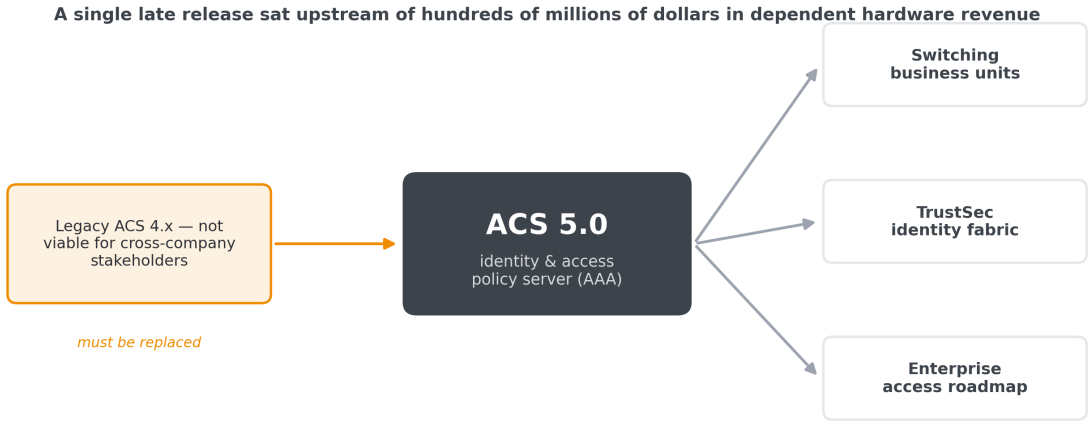


*Figure 1. The strategic position of ACS 5.0. A single late release sat upstream of the switching businesses, the emerging identity fabric, and the enterprise access roadmap. Cropped and genericized from the engagement assessment.*

Program shape

# Worldwide, lateral, and coordinated from one office

The program was genuinely distributed: development ran across multiple sites and several continents, with hundreds of engineers, and it was coordinated from a single program office in Israel. The team had already adopted advanced engineering methods — extreme programming and agile development [12] — and organized the work into eleven end-to-end teams. Each end-to-end team owned its part of the product across the full path: requirements, design, code, and out through integration and test until its element was delivered to specification in the final build. [4]

This lateral, cross-functional ownership is a core fast-team principle — what Wheelwright and Clark named the heavyweight team [8] — and it is one the FTTM method was designed to support rather than replace. Many of the ideas behind extreme programming and agile — flexible requirements, self-organizing teams, continuous improvement — were anticipated by the lateralworks research into fast teams that began in Silicon Valley in 1990. [5] The program did not need a different philosophy. It needed a way to plan and steer that many moving parts toward one date.

02

## The problem

# A program that had stopped converging

A program can be late for good reasons and still be in control. This one was late and had lost control of its own forecast. That distinction is the whole problem.

lateralworks opened the engagement with a structured assessment of the program — not of the code, but of how the program understood itself. The assessment walked nine areas, from portfolio fit and market segmentation through scope, time-to-market drivers, success criteria, cost of delay, the customer and the value proposition, requirements stability, and the as-is data already in hand.

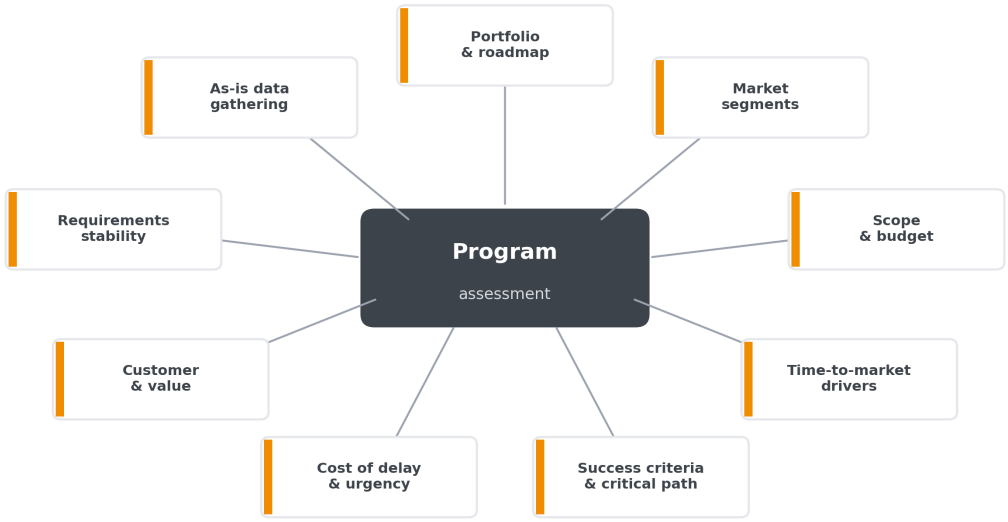


*Figure 2. The program assessment. Nine lines of inquiry used to establish the real starting condition before any planning began. Genericized from the engagement assessment map.*

Diagnosis

## What the assessment found

The findings were consistent and they reinforced each other. The 5.0 feature set for first customer shipment (FCS) was unstable and still changing. Requirements moved during development, and closed decisions were reopened. There was no product roadmap for the generations beyond 5.0, so every request felt equally urgent. Planning and tracking skills were thin at the team level, the program team structure — roles, leadership, control — was unclear, and there was no integrated plan, no schedule trending, and no tracking system that the whole program could see. A year of direction changes had stretched the program, and the target for FCS was aggressive given that history.

The team's own words from the pre-engagement survey were blunter than any diagnosis. Decisions were made by large groups with too many arguments. Too many people were involved in low-level decisions. The prevailing reflex, in one engineer's words, was a "can't do" attitude rather than a search for a way through. None of this is unusual on a program that has been slipping for a year. All of it is fatal to a date.

The real symptom

## The forecast had stopped converging

Underneath the cultural symptoms was a measurable one. When lateralworks built the first honest model of the schedule and compared it to the committed targets, the gap was large and, worse, it was not closing. Major milestones were predicted weeks to months beyond their targets, and because no one was tracking the prediction over time, the drift was invisible until a date was missed. A program that cannot see its own gap cannot create the urgency to close it.

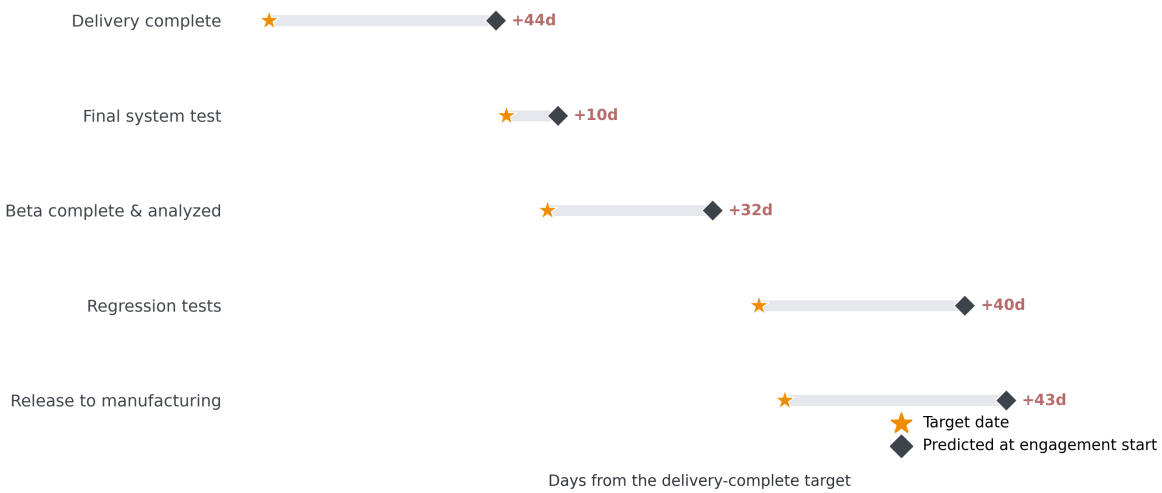


*Figure 3. The gap at engagement start. Each milestone's honest prediction (diamond) sat well beyond its target (star). Without trend tracking, this drift had been invisible. Reconstructed from the engagement schedule analysis.*

Pre-engagement, January 2008

## The starting condition

**Mostly a “can’t do” attitude. Too many people in low-level decisions. No integrated plan, no schedule trending.**

Composite of ACS 5.0 team members

lateralworks pre-engagement survey, January 2008

# 03

## The approach

## How lateralworks recovered the schedule

The recovery was not improvised. It was the installation of a defined system — the lateralworks Fast Time to Market method — in the specific order that makes a late program start converging.

FTTM rests on a simple, counterintuitive idea: the way to deliver the right product at the right time is not to freeze everything, but to keep changing the product and the schedule deliberately, in public, with the team. [5, 6] Speed is the consequence of a few disciplines installed together. The sections below describe the four that mattered most here, in the order they were applied.

Discipline one

# Build one schedule, macro then micro

The team first built a macro map: a single top-level architecture for the whole program, anchored in explicit doneness criteria — the measurable exit conditions for each milestone and for FCS itself. Doneness is where FTTM planning starts, because you cannot schedule toward an end you have not defined. [6] The macro normalized several disconnected data streams into one integrated picture and exposed inconsistencies that had been hiding in separate spreadsheets.

From the macro, the team developed the master schedule down to the next level of detail and then connected the eleven end-to-end schedules beneath it. Detailed team work rolled up into the master; cross-team interface points were driven down from the master into each team's plan. The result was one information system, built and integrated with the lateralworks macro-micro scheduling tools [7], not a stack of schedules that disagreed with each other. Macro and micro were reconciled — top-down expectation against bottom-up reality — which is the alignment step FTTM treats as non-negotiable before a program is allowed to run hard. [6]

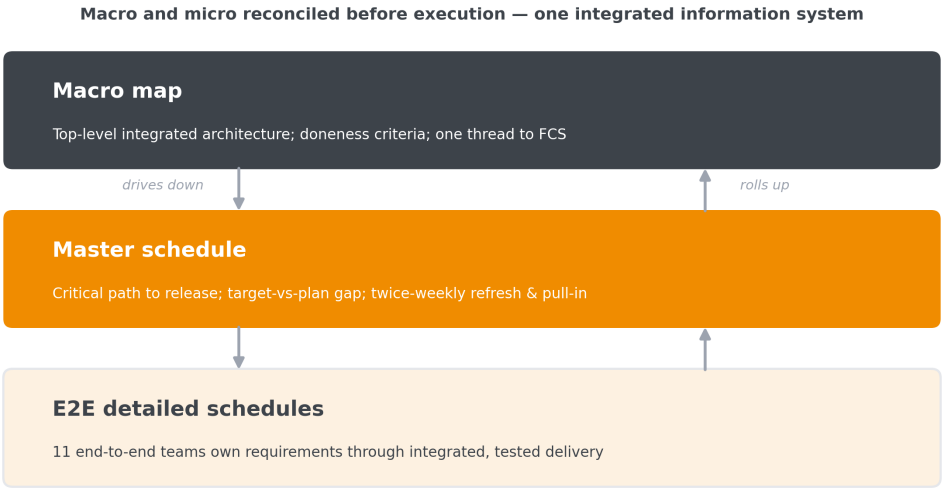


*Figure 4. The macro-micro schedule architecture. One integrated system: detailed end-to-end work rolls up; cross-team interfaces drive down. Genericized from the engagement schedules.*

Discipline two

# Find the real gap, then create urgency

With one honest schedule in place, the team measured the gap between the target dates and the realistic prediction, and used the critical path to find where time could be recovered. This is the FTTM move that manufactures urgency before a deadline forces it: when a program is nominally on schedule, nothing changes, because nothing has to. When the real gap is visible and large, people start looking for different ways to work so the gap can close. [6] The team also looked past the single critical path to the second and

third paths, where unmanaged slack quietly turns into the next delay. The economics are well understood: a cost-of-delay view quantifies what each week of slip is worth, which is what gives pull-in its urgency. [10]

Discipline three

## Refresh and pull in, two and three times a week

The engine of the recovery was cadence. The program refreshed the schedule two to three times a week with the team: update what actually happened, analyze the gap to target, and then pull in — compress the critical path by resequencing, rescoping, or reallocating. Refresh is what keeps a schedule honest; pull-in is what makes it move. Run often enough, the pair converts the abstract urgency of the gap into concrete schedule compression every few days. [5, 6]

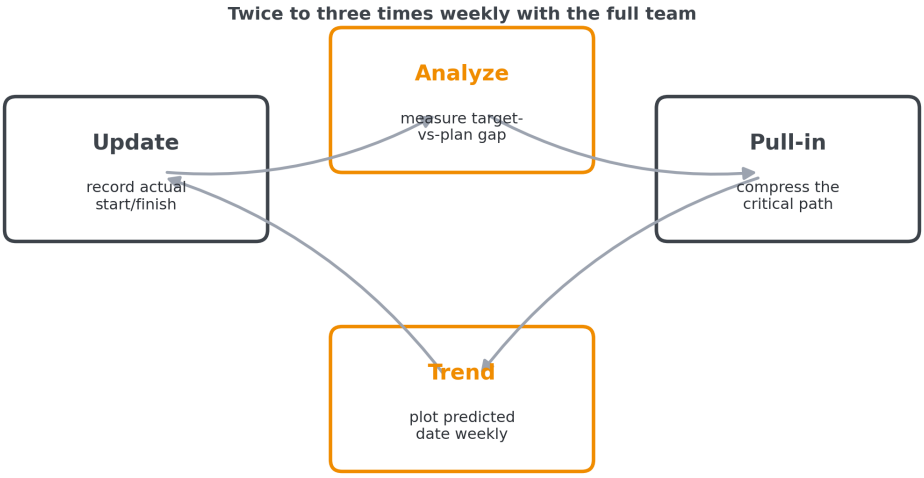


*Figure 5. The refresh and pull-in cycle. Update, analyze the gap, pull in the critical path, and trend the prediction — run two to three times a week with the full team.*

Discipline four

## Trend the prediction so drift is visible

Every refresh produced a new prediction for each major milestone, and those predictions were plotted over time as a trend — the lateralworks wigglechart. Instead of a single date that is either met or missed, the program could now watch its forecast move week over week, see whether it was converging on the target or drifting away, and read how many days a week it still needed to pull in to land on time. Early-warning indicators flagged milestones that required more aggressive action. Drift that had been invisible for a year became a line on a chart that everyone could read.

# 04

## Execution

## Closing the gap, week by week

The disciplines are simple to describe. Installing them against the drag of a distributed, year-late program is the hard part, and it is where the engagement lived day to day.

The signature view

# The forecast starts to converge

The clearest evidence of the turnaround was the milestone trend itself. After an early period where the predicted delivery date jumped further out as the honest schedule absorbed reality, the refresh and pull-in cadence took hold and the prediction began a steady march back toward the target. At one tracking point the delivery-complete milestone needed thirty-two days pulled in over the next seventy to hit its target — a demanding but now quantified and shrinking number, the kind of gap a team can actually attack rather than fear.


*Figure 6. Milestone trend (wigglechart) for the delivery-complete milestone. The predicted date spikes as the schedule turns honest, then converges toward target as refresh and pull-in take hold. Marker color signals risk to target. Reconstructed from the engagement trend reports.*

Resourcing

# Matching the accelerated plan to capacity

Pulling in a schedule is only real if the people exist to do the pulled-in work. As the plan accelerated, the team analyzed load against capacity and surfaced the over-allocation that an aggressive schedule always hides. Making the over-allocation visible let the program rebalance deliberately rather than discover the shortfall when a milestone slipped. FTTM treats balanced resourcing as a host responsibility: provision the team for the plan you are asking it to deliver, or change the plan. [5]

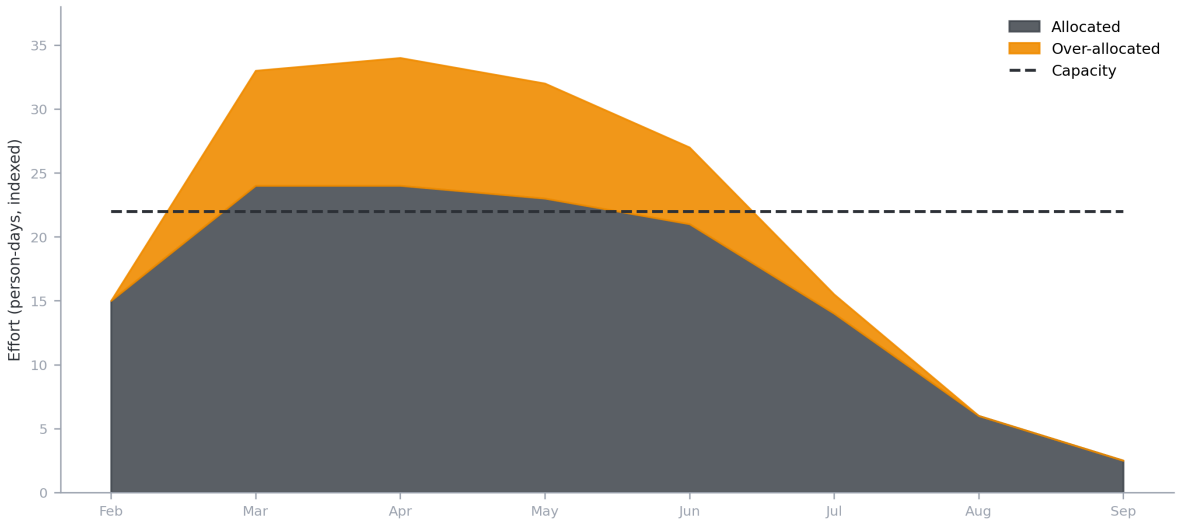


*Figure 7. Schedule versus capacity. The accelerated plan pushed several months into over-allocation (orange) above sustainable capacity (dashed), which the program then rebalanced. Genericized from the engagement resource analysis.*

Scaling the method

# Pushing planning down to the teams

The most durable change was making the method the teams' own. Each end-to-end lead reported progress at the twice-weekly refresh, and their updates fed the master schedule. Two teams went further in a pilot: their detailed plans were integrated directly into the master so progress rolled up automatically and cross-team interface dates drove down to them, which removed a standing tax of manual schedule clean-up and coordination. The program office could then manage by exception, putting its attention on the teams actually driving the critical path. One team's bottom-up plan from this period — scope, milestones, resource analysis, dependencies, and open issues, laid out by the team itself — shows the planning discipline taking root where the work happened, not just at the program office. [13]

That same bottom-up planning forced the decision that mattered most for the date: what actually had to be in the first release. The teams worked an explicit FCS feature list and, just as plainly, a list of functionality deferred past first customer shipment. Pulling a schedule in is never only resequencing work; it is deciding the right feature set for the time available and moving the rest to a later release. Choosing what not to ship was as much a part of the recovery as any scheduling move — the FTTM principle of the right product at the right time, doing its job under deadline pressure. [13]

By the closing weeks of the program the pattern was self-sustaining. The management team had adopted the refresh discipline, two remote teams were refreshing three times a week, the master schedule improved every week, and the conversation had shifted from whether the date was real to how many days were left to pull in. The gap was known, the path to the end was clear, and the focus was on closing it.

Post-program, late 2008

**The instrument that changed behavior**

**A master schedule, and we knew where we were every week. Focus on the critical path was powerful — we acted sooner.**

Composite of ACS 5.0 team members

lateralworks program survey, 25 responses, Aug–Nov 2008

# 05

## Results

## What changed, and what shipped

Two kinds of result matter here: what the program delivered, and what the people on it reported afterward. Both point the same way, and the honest version of each is more useful than a victory lap.

What shipped

# A converging program that delivered

The program moved from an open-ended slip to a managed, converging schedule with a known and shrinking gap, and ACS 5.0 was delivered. That is the result that protected the businesses downstream: their products were sequenced behind a release that now had a credible date and a plan to hit it. It is worth being precise about the mechanism rather than claiming a tidy miracle. lateralworks did not make a year of slip disappear overnight. It restored the three things a late program needs — visibility, predictability, and urgency — and gave the team an instrument to drive itself toward the date instead of away from it.

What the team reported

# The post-program survey

Late in the program lateralworks surveyed the team. Twenty-five people responded between August and November 2008, rating the program across confidence, risk management, estimation, role clarity, communication, the end-to-end approach, and the process itself. The strongest marks went to role clarity and communication, each rated good or better by four in five respondents, and to confidence in meeting the target dates — a striking reading for a program that had been a byword for missed dates a year earlier.

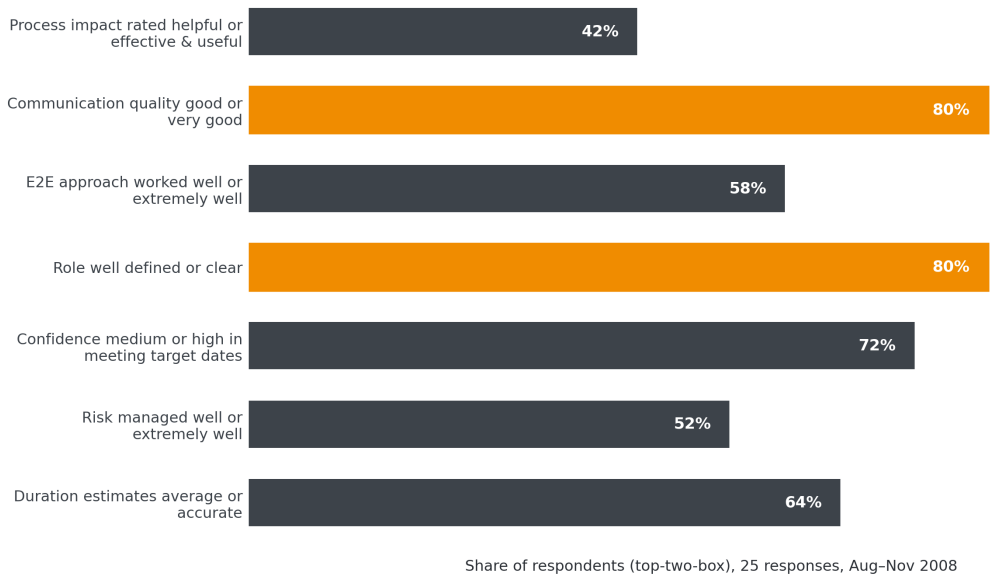


*Figure 8. Post-program survey, top-two-box responses across the rated dimensions. Twenty-five responses, August to November 2008. Reconstructed from the engagement program survey.*

The survey was not uniformly glowing, and that is to its credit. The end-to-end concept was rated well by most but exposed real friction: responsibility grey areas between teams, leads who were also functional managers and so were overloaded, and the absence of a single lead architect with a view of the whole product. Risk management was the weakest area, with a substantial minority saying the program never built a real risk process. The team's own recommendations for a next program were concrete — freeze scope sooner, push planning detail further into the teams, define interfaces so groups can work independently, and

run risk management as a standing discipline. A recovery that produces an honest critique of itself is a recovery that taught the organization something.

What stayed behind

## A capability, not just a date

The most valuable outcome was not the date; it was the capability. A standard planning and tracking process was in place, the team leads had stronger planning skills, and the refresh discipline continued after the engagement without lateralworks driving it. The program had learned to run itself the FTTM way, which is the only kind of process change that survives the consultant leaving the building.

# 06

## Significance

## Why it mattered to Cisco

It is tempting to file a schedule recovery under project management and move on. The longer view is more interesting, because of where this particular release sat in Cisco’s architecture.

At the time

## Protecting the roadmap, not the product

In the moment, the value of an on-time ACS 5.0 was almost entirely external to ACS. The switching businesses that had sequenced products behind it could proceed on their own dates; the large internal stakeholders who had bet on it kept their confidence; and Cisco avoided the worst case the team itself had named — a loss of customer confidence severe enough to consider replacing ACS with an acquisition or a third-party product. [13] The recovery defended a multi-billion-dollar business and the hundreds of millions in dependent hardware revenue that rode on it. [4]

Since

## The foundation Cisco’s access strategy was built on

The rule-based policy architecture introduced in 5.0 turned out to be the foundation for where Cisco took network access next. The 5.x line became the AAA policy server behind Cisco TrustSec, the company’s identity-driven, role-based access architecture: TrustSec’s AAA is supported by the Secure Access Control System from version 5.1 onward, with the access-control server acting as the authentication server that assigns devices to security groups. [1, 2] In other words, the platform whose schedule was recovered here became load-bearing for an architecture that outlived it.

Cisco later carried that identity-and-policy strategy forward into the Identity Services Engine (ISE), and the Secure Access Control System was eventually retired as ISE became the center of gravity. [2, 3] Read across two decades, ACS 5.0 was a hinge: the generation that moved Cisco from legacy access control to a rule-based, identity-aware model, and the proving ground for ideas that the company’s current access portfolio still reflects. That is a program worth recovering.

> **The lasting point.** The engagement protected far more than one release date. It protected the businesses downstream of ACS 5.0 and the architecture that grew out of it. When a program sits at the base of a portfolio, recovering its schedule is portfolio strategy, not project hygiene.

A

## Appendix

# The best-practice framework, applied

The engagement drew on the lateralworks best-practice framework, which reads a program along two axes: the four levers that determine whether work gets done, and the two parties responsible for them. The framework is reproduced here in genericized form, followed by the specific practices used on this program.

The framework

# Host and team across four levers

Every delivery system depends on four levers — the systems and processes in place, the knowledge and skills available, the motivation to act, and the resources provisioned — and each lever has two owners. The host is the organization around the team that sets strategy, provisions people and budget, and either removes interrupts or creates them — and getting this wrong is expensive at scale; excess management has been estimated to cost the U.S. economy roughly three trillion dollars a year. [11] The team executes. Fast organizations get both rows right; slow ones fix the team and leave the host alone, and the improvement fades. [5]


*Figure 9. The lateralworks best-practice framework. Four levers (systems, knowledge and skills, motivation, resources) read across two owners (host and team). Genericized from the engagement analysis.*

Applied here

# Which practices did the work

| lateralworks practice | How it was used on ACS 5.0 |
|---|---|
| **Doneness criteria** | Defined measurable exit conditions for each milestone and for FCS before scheduling began. |
| **Macro-micro reconciliation** | Built one macro map, then a master schedule with eleven end-to-end schedules rolled up beneath it. |
| **Real schedule gap** | Measured target-versus-prediction to manufacture urgency before deadlines forced it. |
| **Critical-path pull-in** | Compressed the first, second, and third paths through resequencing and reallocation. |
| **Refresh cadence** | Updated and pulled in the schedule two to three times a week with the full team. |
| **Wigglechart trending** | Plotted each milestone's predicted date over time so drift and convergence were visible. |
| **Lateral / end-to-end ownership** | Eleven teams owned requirements through integrated, tested delivery; interfaces driven from the master. |
| **Heavyweight core team** | Recommended a real core-team structure with a heavyweight program manager and faster decisions. |
| **Balanced provisioning** | Analyzed load against capacity and rebalanced the accelerated plan to sustainable levels. |
| **Skills transfer** | Left a standard planning and tracking process the teams continued to run independently. |

Sources

# References

[1] Cisco Systems. “Cisco TrustSec Switch Configuration Guide: Configuring the Cisco TrustSec Solution.” AAA for Cisco TrustSec is supported by the Cisco Secure Access Control System (ACS) version 5.1 and later. cisco.com.

[2] Cisco Systems. “Cisco TrustSec Configuration Guide” (Catalyst 9300 / C9350). The authentication server in a TrustSec domain can be the Cisco Identity Services Engine (ISE) or the Cisco Secure Access Control System (ACS). cisco.com.

[3] Cisco Systems. “Cisco Secure Access Control System — Retirement Notification.” The Secure Access Control System has been retired and is no longer supported. cisco.com.

[4] lateralworks. “Projects: Cisco ACS 5.0.” Engagement summary describing the eleven end-to-end teams, the extreme-programming and agile context, and the competitive and revenue stakes. https://lateralworks.com/projects

[5] lateralworks. “Research: the FTTM best-practice framework.” Host, team, and mindset; provisioning and interrupt removal; right product at the right time. https://lateralworks.com/research

[6] lateralworks. “The FTTM system.” Macro-micro reconciliation, doneness criteria, before-the-fact urgency, refresh and pull-in. https://lateralworks.com/ideas/fttm-system

[7] lateralworks. “FTTM Tools.” Macro-micro scheduling, gap identification, and consensus pull-in for portfolios and large programs. lateralworks-software.com.

[8] Wheelwright, S. C., and Clark, K. B. Revolutionizing Product Development: Quantum Leaps in Speed, Efficiency, and Quality. Free Press, 1992. Origin of the heavyweight team / core team concept.

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

[10] Reinertsen, D. G. The Principles of Product Development Flow: Second Generation Lean Product Development. Celeritas Publishing, 2009. Cost of delay and queueing in development.

[11] Hamel, G., and Zanini, M. “Excess Management Is Costing the U.S. $3 Trillion Per Year.” Harvard Business Review, September 2016.

[12] Beck, K. Extreme Programming Explained: Embrace Change. Addison-Wesley, 1999. Context for the engineering methods in use on the program.

[13] lateralworks. “Internal engagement data: Cisco ACS 5.0 program.” End-of-engagement report; pre-engagement survey (January 2008); program survey (25 responses, August–November 2008); end-to-end high-level planning policy (January 2008). 2008.