Case study

100,000 activities, one critical path

How the macro-micro roll-up in fastProjectAI let a $7 billion greenfield fab run eight independent module schedules — and still read a single end-to-end program critical path

Field cases in Fast-Time-to-Market.

A deep dive into the schedule architecture behind the Fab8 startup in Malta, New York: more than 100,000 activities, eight process-module micro-schedules refreshed twice a day, and first silicon two weeks early. Illustrations are cropped from lateralworks program schedule archives.

Prepared by
lateralworks
FTTM field methodology

Date
July 2026
Case study

Online
lateralworks.com
Field cases series

Table of contents

100,000 activities, one critical path

Core thesis. A program of 100,000 activities cannot be read in one schedule, and it cannot be run in fifty disconnected ones. The macro-micro roll-up keeps every schedule small enough for one team to own, and connected enough that a single critical path runs from the first procurement task to first silicon.

Overview

Abstract

Fab8 was too big for a schedule and too interconnected for many. The $7 billion greenfield fab in Malta, New York needed more than 100,000 activities across its combined schedules to reach first silicon [1, 2]. No program manager can read a network that size, and no scheduling tool of the day could hold it in one file with dozens of people updating it. The conventional escape, splitting the program into independent sub-schedules, trades one failure for another: the pieces stay manageable, but nobody can see the program.

lateralworks resolved the dilemma with a schedule architecture called macro-micro. One macro-schedule carried the cross-functional flow of the program at summary level. Eight process-module micro-schedules carried the detail, each owned and refreshed by its own module team. Custom roll-up code built on Microsoft Project synchronized the levels twice a day; that machinery later became the Macro-Micro Roll-up function in fastProjectAI [2, 3].

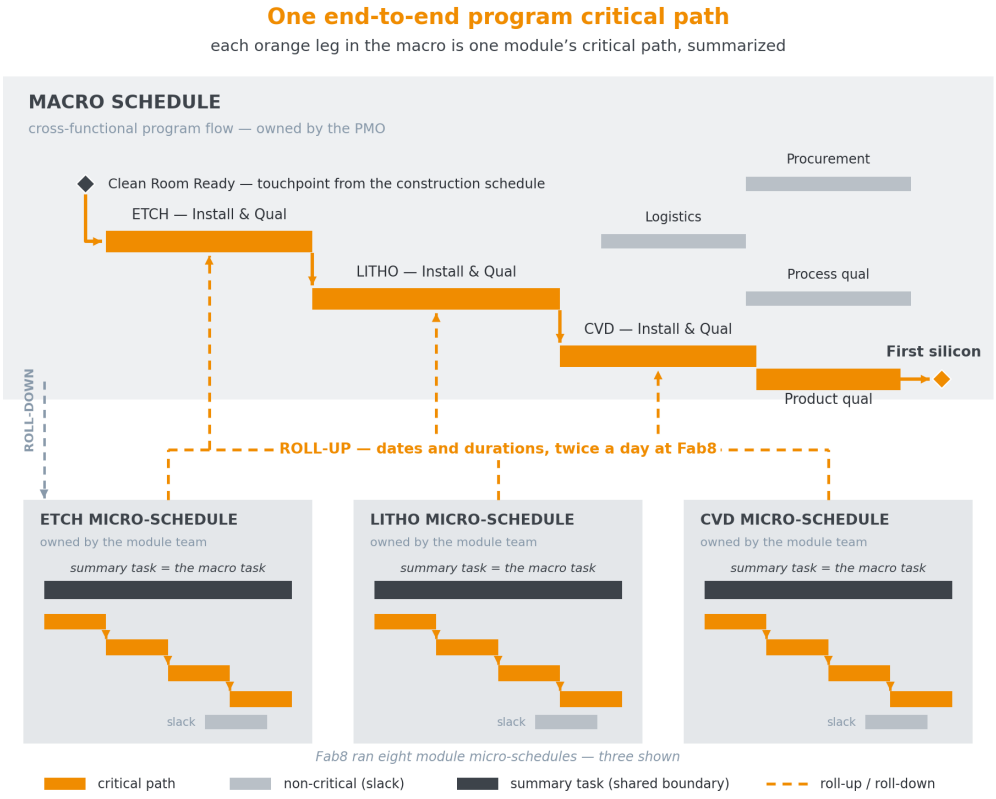


Figure 1. The whole idea in one picture. Each module micro-schedule computes its own critical path (bottom). The roll-up carries each module's dates and durations into its macro summary task, and the macro links those tasks laterally — construction touchpoints, module-to-module hand-offs, qualification — so the critical-path calculation at the top yields one end-to-end program critical path: a summarized version of the detailed micro critical paths.

The distinctive part is what the synchronization carries. Most roll-up reporting copies dates upward into a summary that management reads as a status page. The macro-micro roll-up instead rebuilds the

macro-schedule as a live critical-path model: each macro task takes its dates and durations from the detailed network beneath it, and the macro links those tasks laterally across modules, suppliers, construction, and qualification. The result is a program critical path that is a summarized version of the detailed micro critical paths — short enough to read in one sitting, real enough to drive pull-in decisions.

This case study walks through the architecture as Fab8 ran it: why single-file and master-subproject scheduling both break at fab scale, how roll-up, roll-down, and touchpoints keep nine .mpp files behaving as one model, how the tool template and dock-leveling rules turned 170-plus process tools into a repeatable schedule pattern, and what the operating rhythm looked like at two roll-ups a day. Fab8 delivered first silicon two weeks ahead of schedule, on a program where the team costed delay at roughly $5 million a day [1].

The technique also works at ordinary scale. The closing section shows the same roll-up running a three-schedule firmware and hardware program, and the published Project Everest case applies it to a battery storage platform [4]. The scale changes; the architecture does not.

01

Section 01
The scale problem

A greenfield fab startup is a scheduling problem with few equals in industry. The building itself — clean room, subfab, utilities, automation — is only the container. The program inside it must procure, ship, install, hook up, and qualify hundreds of process tools, in a sequence constrained by construction milestones, supplier lead times, dock capacity, gas and chemical availability, and the process flow of the silicon itself [2].

Figure 2. Fab8, Malta, New York — the most advanced pure-play semiconductor foundry campus in the U.S., with 3,300-plus employees and a 460,000 sq ft clean room at full build-out. From the Fab8 program overview [12].

The schedule is structured around ramp steps: a pilot line to reach Risk Start (the point where the first production wafers start, at risk, ahead of full qualification), then capacity steps of 4,000 and 7,000 wafer-outs per month. Each ramp step needs a specific population of tools qualified by a specific date, set by industrial-engineering models of the process flow. The pilot line alone required more than 170 process tools [2]. Each tool is a small project in its own right: negotiation, purchase requisition, purchase order, lead time, shipment, customs, dock, move-in, two stages of hookup, supplier qualification, fab-owner qualification, unit module recipe. Multiply thirty-odd tasks by hundreds of tools, add facilities, permits, automation, logistics, and qualification lots, and the pilot line already carried more than 6,500 tasks at summary level [2]. The full program, detail included, passed 100,000 activities.

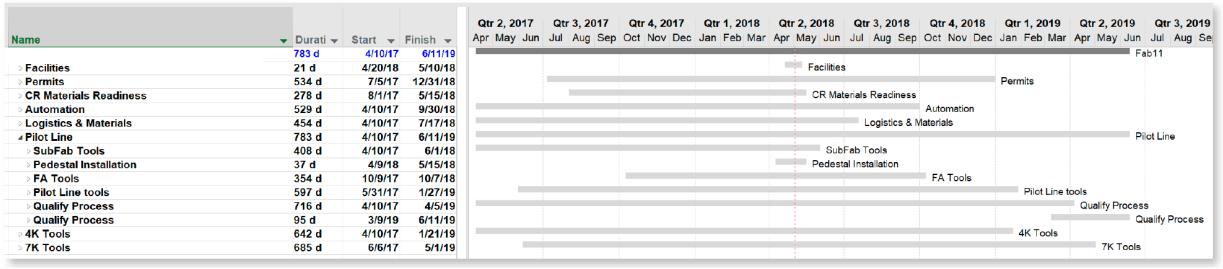


Figure 3. The top level of the fab macro-schedule: facilities, permits, clean-room materials, automation, logistics, and the pilot line, each decomposing into tool groups and ramp steps. From the lateralworks program archive.

Three ways to run a large program

Practitioners have three basic options for a program this size [3]. The first is one integrated schedule. It is the right answer for most projects, and lateralworks recommends it for small and medium programs: every task in one file, every change visible instantly, one person trained. At fab scale it collapses. The program manager becomes an information bottleneck, spending the week feeding the schedule instead of accelerating it. Teams cannot own their piece because there is only one file and, in Microsoft Project, effectively one user in it. And with tens of thousands of tasks on the network, the critical path becomes a wall of bars that hides the strategic pull-ins it is supposed to reveal.

The second option is Microsoft Project's own answer: embed sub-projects in a master schedule, with live links between files [5]. The idea is appealing and the workload does distribute. In practice the linked-file mechanism is brittle, and Microsoft's support forums document the failure modes: crashes on opening files that contain cross-project links [6], sub-projects that a master file can no longer open or expand [7], and broken links that multiply as files move or get renamed. There is a structural problem too. Sub-projects must be cut along the natural flow of the program, which rarely matches how teams actually work. And because every detailed task lives on the integrated network, the critical path threads through thousands of micro tasks — too big to follow, and of little value when hunting for pull-ins [3].

The third option is the one Fab8 ran: keep the detail out of the master file entirely, and synchronize summary tasks with detailed sub-schedules through software rather than live links. lateralworks calls this macro-micro. The split is finer-grained than sub-projects (it happens at the task level, not the file level), and the synchronization is a controlled, validated batch process instead of a standing web of file links. One later greenfield program using the same architecture carried more than 35,000 tasks below a macro-schedule that stayed readable [3].

The choice among the three is a judgment about scale and structure. As working guidelines: a single schedule for small and medium programs; embedded sub-projects when suppliers already maintain their own .mpp files that roughly follow the program flow; macro-micro for large programs where functional teams need to own their detail independently [3].

02

Section 02

The macro-micro architecture

The macro-micro architecture rests on one structural rule: the lowest level of the macro-schedule is the top level of each micro-schedule. A Supplier Qual task that appears as a single bar in the macro is a summary task in the Litho module's micro-schedule, where it breaks into power-on, recipe setup, calibration, handshaking, and test tasks. One level's micro is the next level's macro [2].

Information moves through that shared boundary in a disciplined cycle. Before work starts, the macro carries the initial end-to-end plan — management's statement of what the business needs, and the first gap analysis. The module teams then build bottom-up plans in their micro-schedules — the second gap analysis, this time against engineering reality. The roll-up reconciles the two views and the program baselines. After that, estimates give way to actuals and the cycle repeats as a refresh: micros update, the roll-up rebuilds the macro, and the macro's dates roll back down.

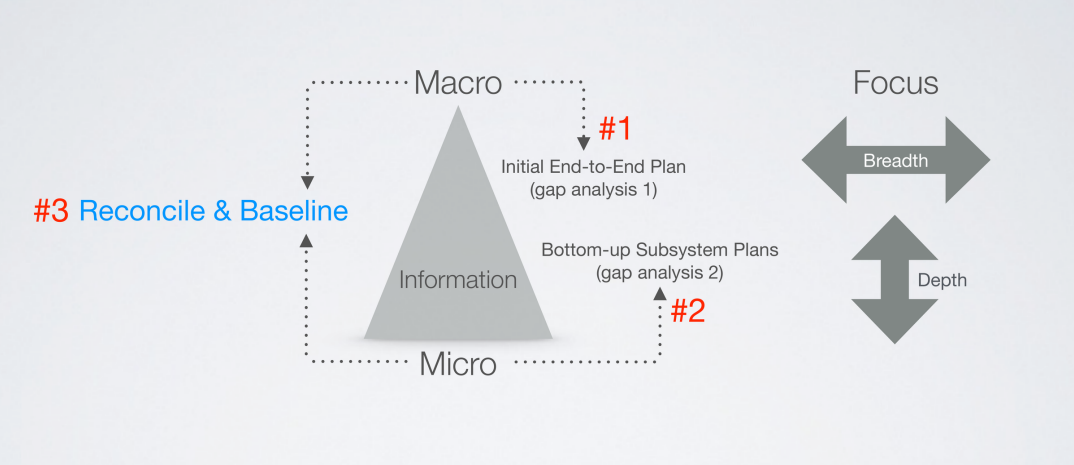


Figure 4. The reconciliation cycle. Top-down: the initial end-to-end plan (gap analysis 1). Bottom-up: subsystem plans from the teams (gap analysis 2). The roll-up reconciles and baselines. The macro gives breadth; the micros give depth.

Two different flows, deliberately

The macro-schedule is cross-functional. Its tasks run laterally, the way the program actually flows: front-end tools before back-end tools, process qualification before product qualification, construction touchpoints ahead of everything they gate. It is owned by the program management team, and its job is to keep management's eyes on the big picture and the strategic pull-ins [2].

The micro-schedules are functional. Fab8 partitioned them by process module — the organizational unit that owns tools, people, and qualification work. Each module manager creates, tracks, and reports their own schedule without ever touching the macro file. This is the point most alternatives miss: because the micros are synchronized through the roll-up rather than embedded in the master network, they do not have to follow

the macro's flow at all. The macro can be a process-flow schedule while every micro is a functional schedule [3]. Freeing each level to take its natural shape is what makes distributed ownership work. Reinertsen argues for the same division of labor in product development generally [8], and McChrystal's "shared consciousness, empowered execution" made the idea famous in a different domain [9].

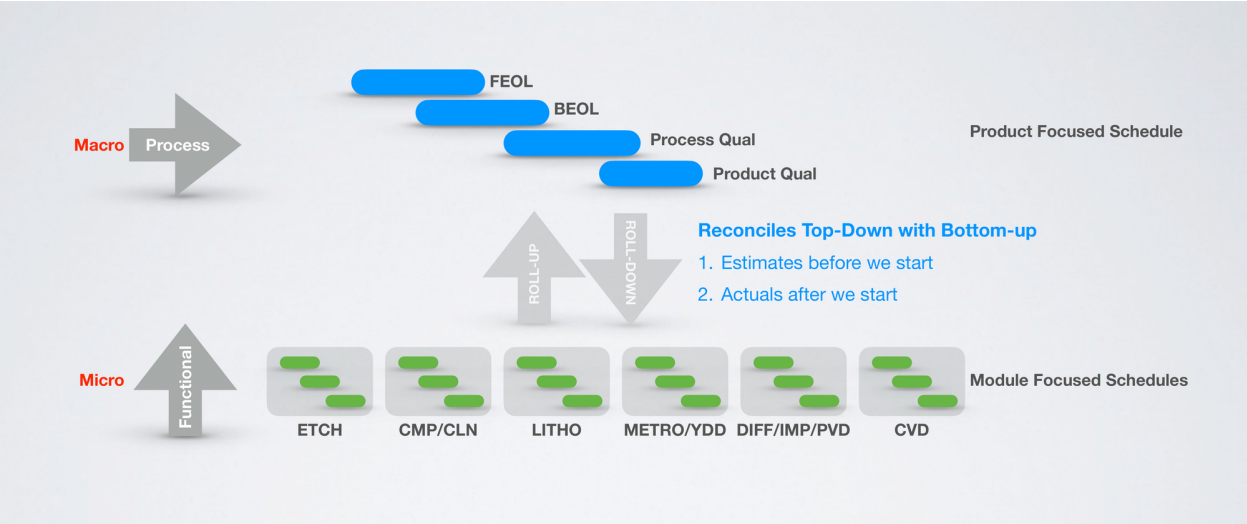


Figure 5. Process flow versus functional flow. The macro (top) follows the product: FEOL, BEOL, process qual, product qual. The micros (bottom) follow the organization: one schedule per process module. Roll-up and roll-down reconcile the two — estimates before the start, actuals after.

Virtual dependencies, not live links

Mechanically, the roll-up synchronizes separate .mpp files through what lateralworks calls virtual dependencies: relationships recorded and enforced by the roll-up software, not by Microsoft Project's linked-file machinery [3]. No file holds a live reference to another, so the failure modes of the master-subproject method (broken links, corrupted views) have nothing to break. Each file stays small, opens fast, and can be edited by its owner all week without coordination.

Synchronization moves three kinds of information. Dates the micro needs roll down from the macro. Dates and durations the macro needs roll up from the micros. And dependencies between two micro-schedules pass through the macro — the roll-up carries them across in the same pass, a lateral hand-off the software calls roll-over [3].

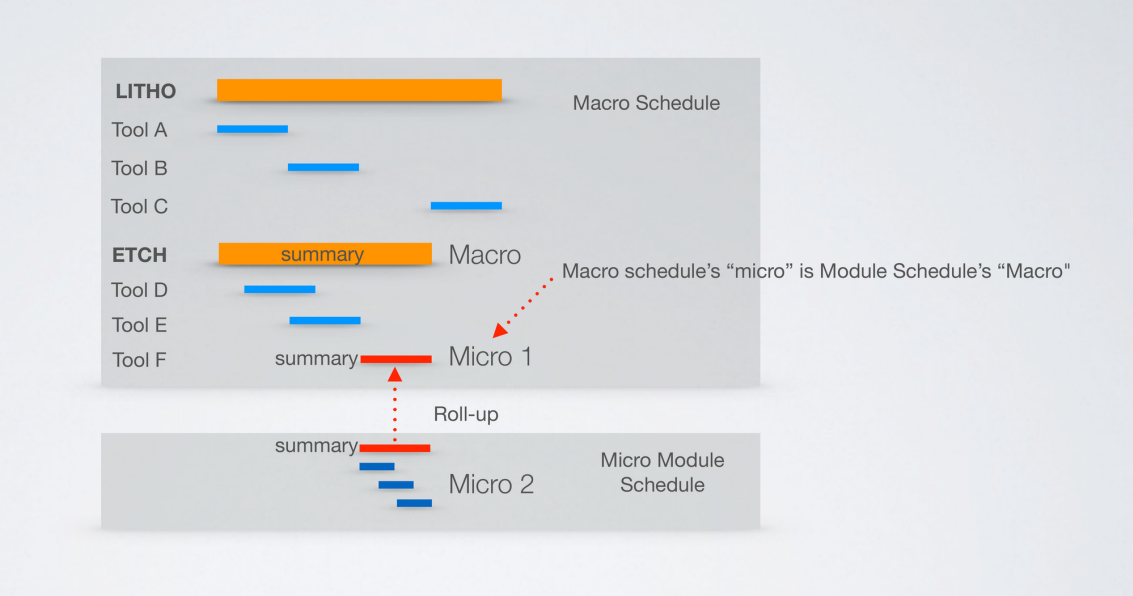


Figure 6. The shared boundary. A summary task in the macro (ETCH) is the top level of the Etch module's micro-schedule. The macro schedule's micro is the module schedule's macro.

Touchpoints

Dates that originate outside a schedule arrive as touchpoints: milestones that mark the place where two independent schedules touch. Clean Room Ready is the canonical example. The construction schedule is managed separately, but tool move-in cannot start without it, so Clean Room Ready appears in the fab macro as a touchpoint and rolls down into every module micro that it gates [2]. Touchpoints are visually distinct from ordinary milestones, so they can be filtered and audited as a class. When a construction date moves, the next roll-down moves it everywhere at once, and the roll-up shows the program-level consequence the same day.

What rolls down at a fab startup: facility dates — building, utilities, gases, chemicals; the environmental impact report; clean-room materials readiness; support tools and accessories; logistics infrastructure; dangerous goods; subfab readiness; predecessor-tool qualifications; dock dates; FOB dates. What rolls up from each module: Hookup 0, move-in, Hookup 1, Hookup 2, hardware installation, supplier qual, fab-owner qual, unit module recipe setup [2].

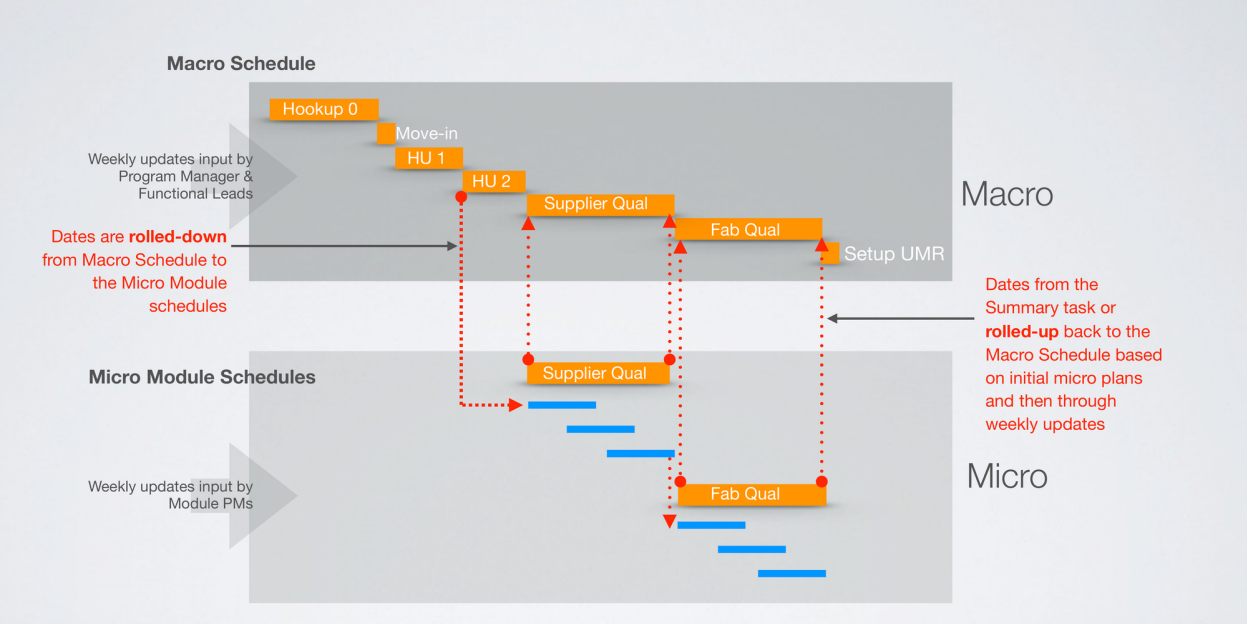


Figure 7. The full loop. Weekly (at Fab8, twice-daily) updates from module PMs roll up; dates from the macro roll down. Red dotted lines are the synchronized boundary tasks; blue bars are module detail that never leaves the micro.

The lateral idea

What made Fab8 different

We were not rolling up dates. We were rolling up critical paths.

The design premise of the macro-micro roll-up

fastProjectAI, lateralworks

03

Section 03

One critical path, end to end

Roll-up reporting is as old as project management: copy the dates upward, publish the summary, repeat next week. What the macro-micro roll-up produces is different in kind, and the difference is the reason Fab8 adopted it. The macro-schedule is not a report. It is a live critical-path model whose inputs happen to come from other schedules.

Every macro task that represents module detail takes its duration and dates from the summary task at the top of the corresponding micro-schedule — which in turn is computed by Microsoft Project's own network logic from the detailed tasks below it. The macro then links those tasks laterally: tool to tool, module to module, construction touchpoint to hookup, predecessor-tool qualification to dependent qualification. Run the critical-path calculation on that network [10] and the longest path threads across modules and suppliers and facilities from today to first silicon. Each leg of that path is a summarized stand-in for the critical path inside one module's detail. The program critical path is a critical path of critical paths.

That is the property date-copying cannot give you. A status roll-up tells you the Litho supplier qual now ends June 29. The macro-micro network tells you whether that slip propagates, through the fab-owner qual and the tools it gates, and by how many days at the program level. It also tells you when a slip does not matter, which is just as valuable: slack absorbs it, nobody escalates, the module fixes it locally.

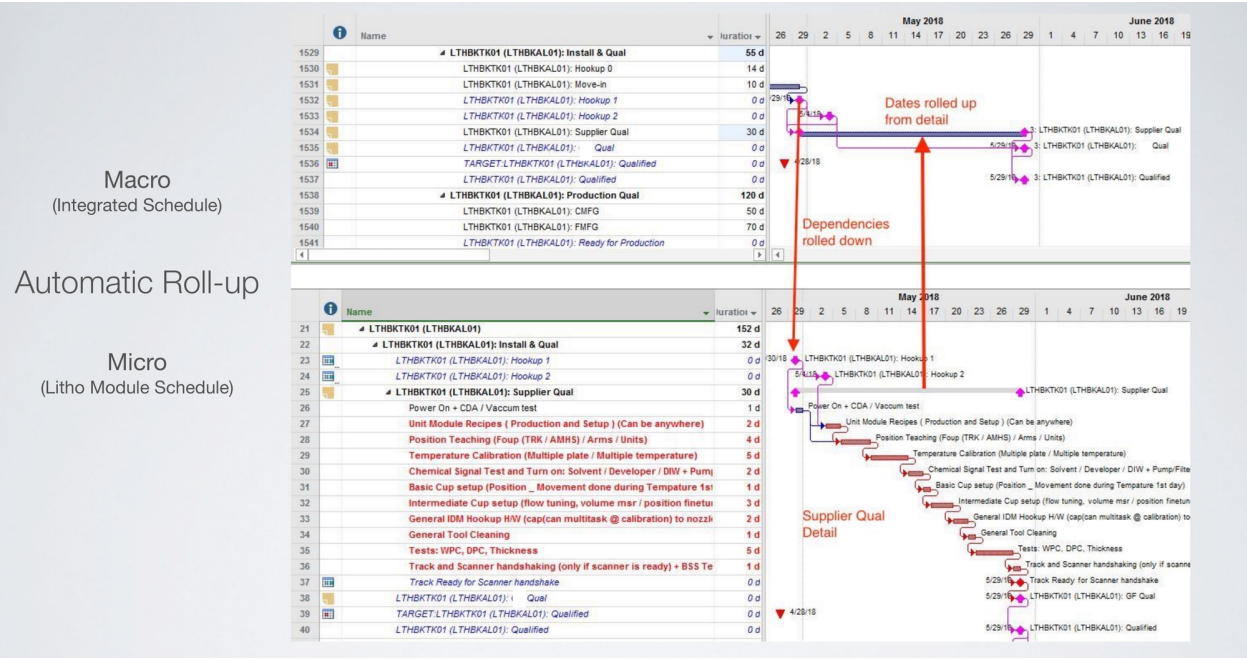


Figure 8. Macro and micro views of the same tool, live in the software. Top: the integrated macro-schedule, where the tool's Supplier Qual is a single 30-day task. Bottom: the Litho module micro-schedule, where the same task is a summary over recipe setup, calibration, tests, and handshaking. Dates roll up from detail; dependencies roll down.

Reading the macro, drilling to the micro

In the macro, a slip announces itself as the gap between a gray baseline bar and its task. The explanation lives one level down. Standing on a roll-up task, the scheduler clicks Goto Task In Linked Schedule; fastProjectAI opens the module's micro-schedule and lands on the same task, where the offending detail (a late parts delivery, a failed calibration cycle, a resource clash) is visible by inspection [3]. The macro answers what and how much. The micro answers why.

At the macro level a supplier qualification is one bar: 53 days. Open the micro and the 53 days decompose into power-on, carousel positioning, reticle-handler teaching, OHT verification, dry cycles, host communication. Management never scrolls through that detail in the program review, and the module team never loses ownership of it. Both views stay true because both come from the same network.

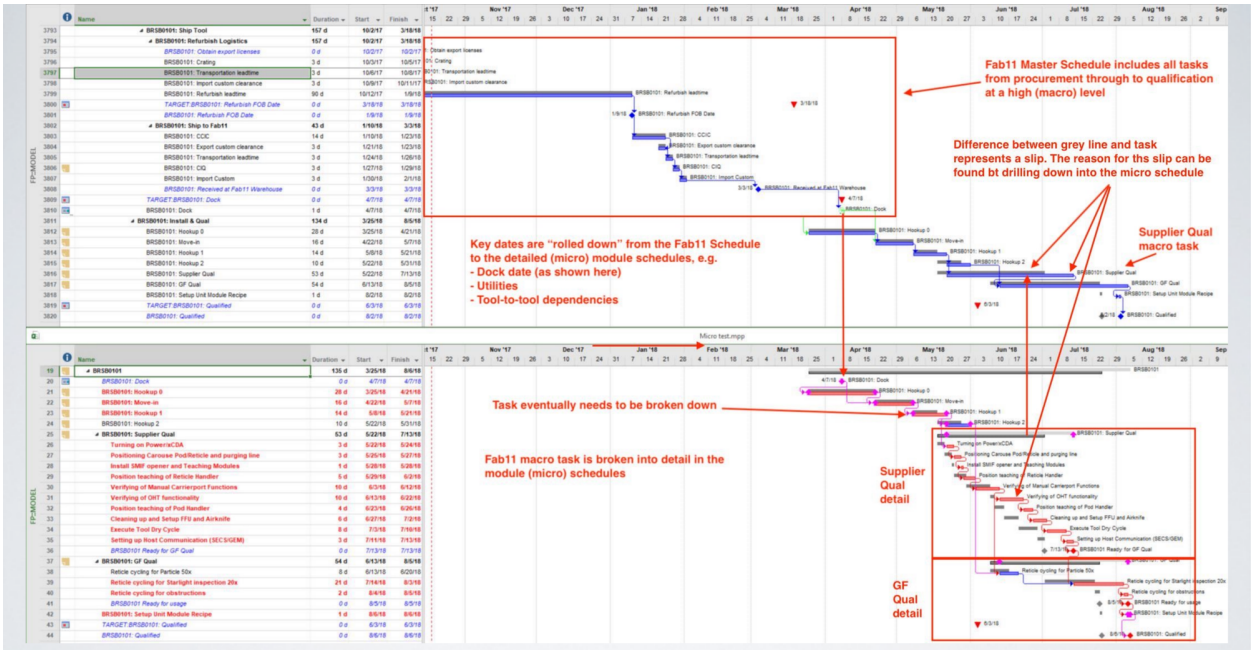


Figure 9. A master schedule (top) and one module micro-schedule (bottom) from the lateralworks program archive. Key dates — dock date, utilities, tool-to-tool dependencies — roll down; the macro task is broken into detail in the module schedule; the gray-line-to-task gap flags a slip whose cause is found by drilling into the micro.

Independent scheduling standards ask for exactly this property and rarely get it at scale. The GAO's schedule assessment guide, the de facto audit standard for large federal programs, requires a single integrated network with a valid program-level critical path, and treats a fragmented schedule — healthy pieces, nothing connecting them — as grounds for finding the forecast unreliable [11]. Fab8's architecture met the requirement without forcing 100,000 tasks into one file: integration lives at the macro level, detail lives with the teams, and the roll-up keeps the two provably consistent.

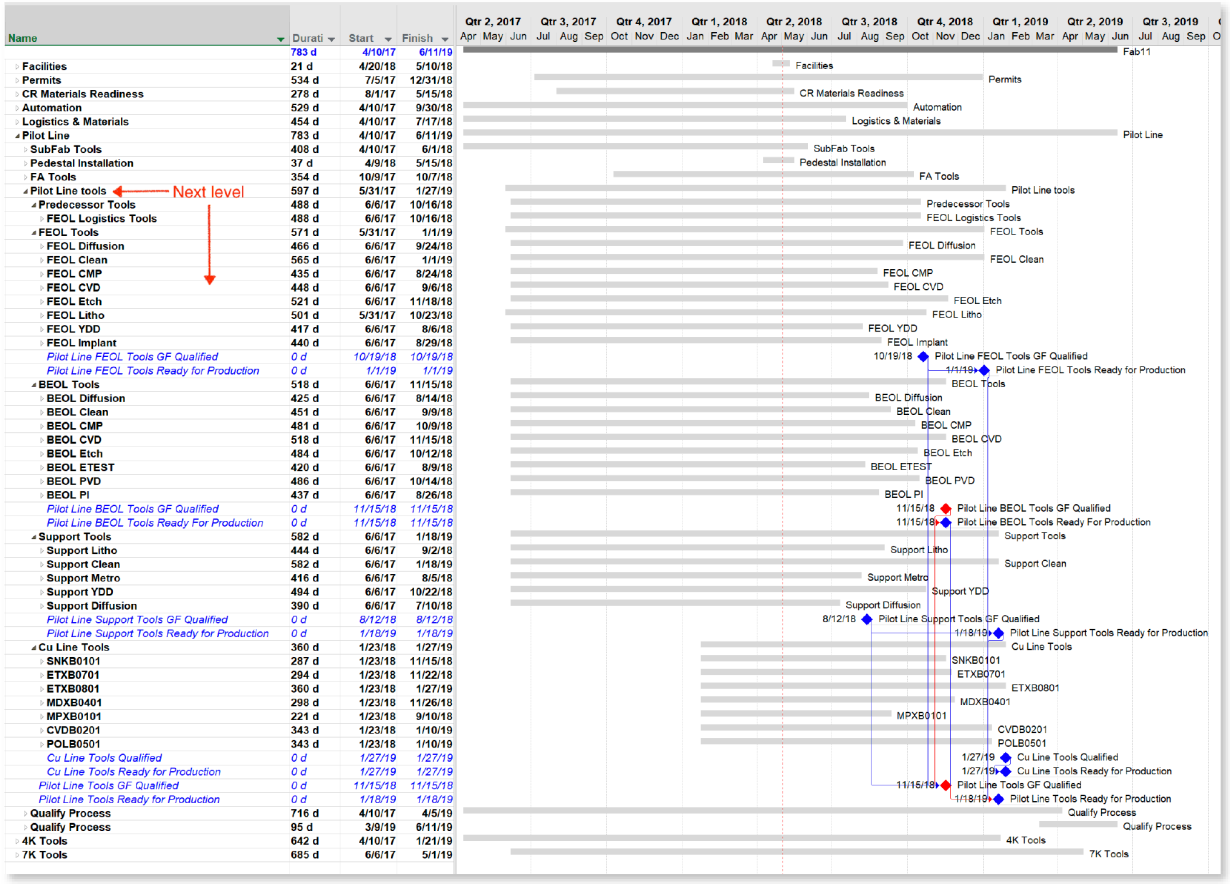


Figure 10. Depth without clutter. The macro hierarchy runs Ramp Step → Tool Group → Process Flow Step → Module → Tool; the schedule stays navigable at every level.

04

Section 04

The machinery of a fab startup

An architecture is only as good as the content flowing through it. Most of the fab program's content is tools, so the schedule is built around a tool template: a standard task pattern capturing the complete life cycle of a tool from negotiation to release to manufacturing [2].

The template has three major blocks. Procurement — negotiation, approval to raise the purchase requisition, PR, PO, lead time — is owned by the procurement organization and exists only in the macro. Install & Qual — move-in, hookups, supplier qualification, fab-owner qualification — is owned by the process modules, and its detail lives in the module micro-schedules. Production Qual follows for tools beyond the pilot line. Refurbished tools get an extra logistics block for export licenses, crating, and refurbishment lead time. Every tool, new or refurbished, looks the same in the schedule, which is what makes hundreds of them buildable and analyzable [2].

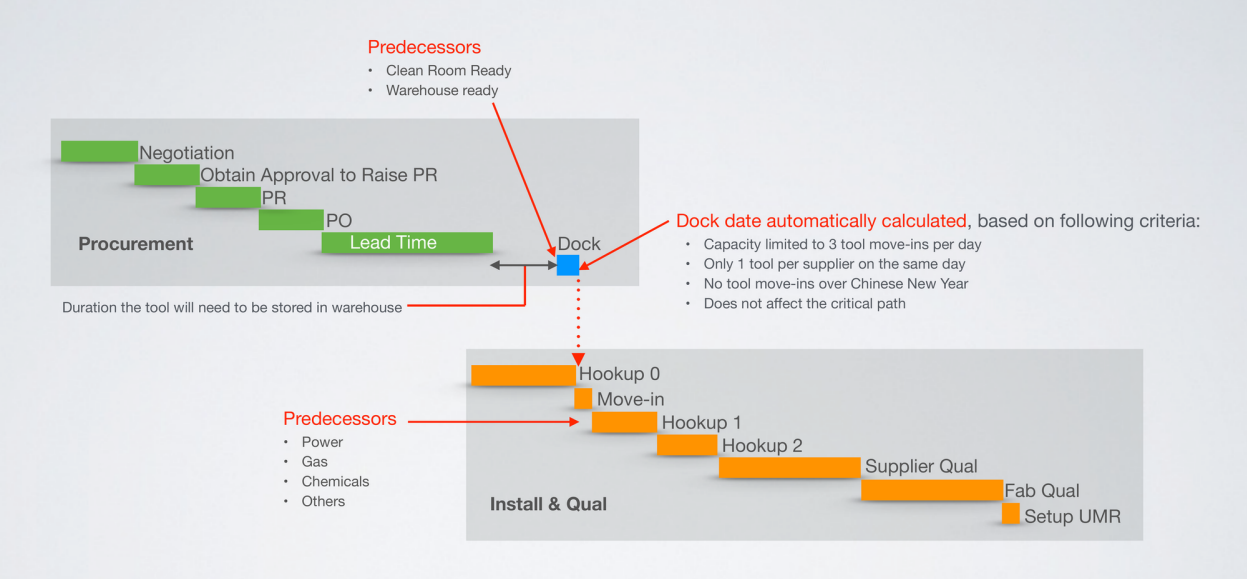


Figure 11. The tool template. Procurement (green) runs in the macro; Install & Qual (orange) is the module's work, with detail in the micro. The dock milestone joins them, gated by Clean Room Ready and warehouse-readiness touchpoints.

Targets and the three ways a tool shows late

Each tool carries three target dates set by industrial engineering and supplier commitments: a target FOB date (ship-ready at the supplier), a target dock date, and a target qualified date, drawn as red triangles against the live network [2]. The system therefore flags a late tool three separate ways: predicted dock against target dock, predicted qualification against target qualification, and predicted ramp step against target ramp step. Lead times keep belief and commitment distinct: an estimate is a task with a duration, updated weekly; a supplier commitment is a milestone with a date [2].

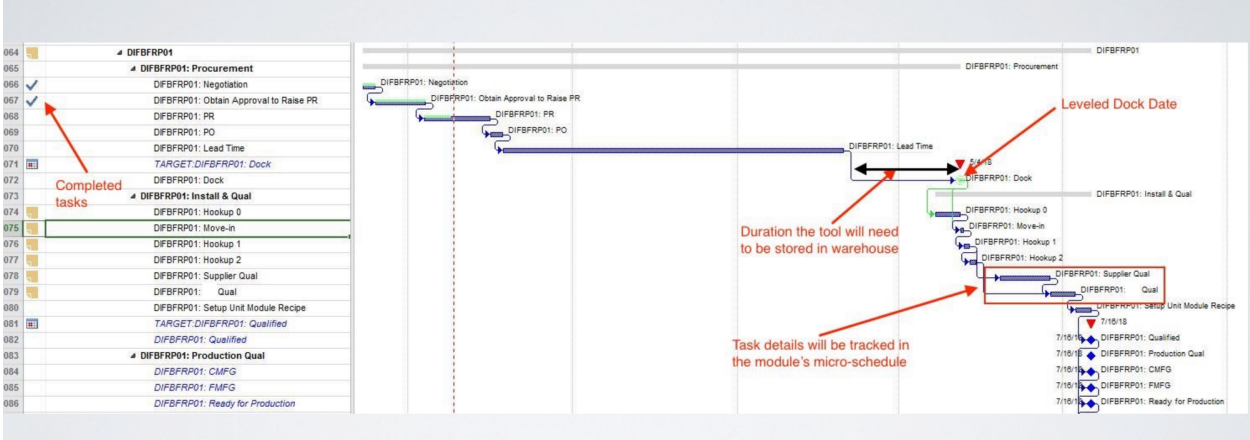


Figure 12. The template instantiated for one tool in the live schedule. Completed tasks check off at left; the red triangles are IE-set targets; the black span is warehouse dwell between an early dock and move-in.

Constraints the schedule enforces by itself

Dock capacity was a physical constraint: three docks, tools arriving from many suppliers in crates of very different sizes. The schedule levels dock dates automatically under explicit rules — at most three tool move-ins a day, one tool per supplier per day, no move-ins over Chinese New Year — and does it without ever letting the leveling push the critical path [2]. A tool that arrives after its leveled slot risks missing its ramp step; a tool that arrives early buys warehouse time, which the template models as a dwell duration ahead of move-in.

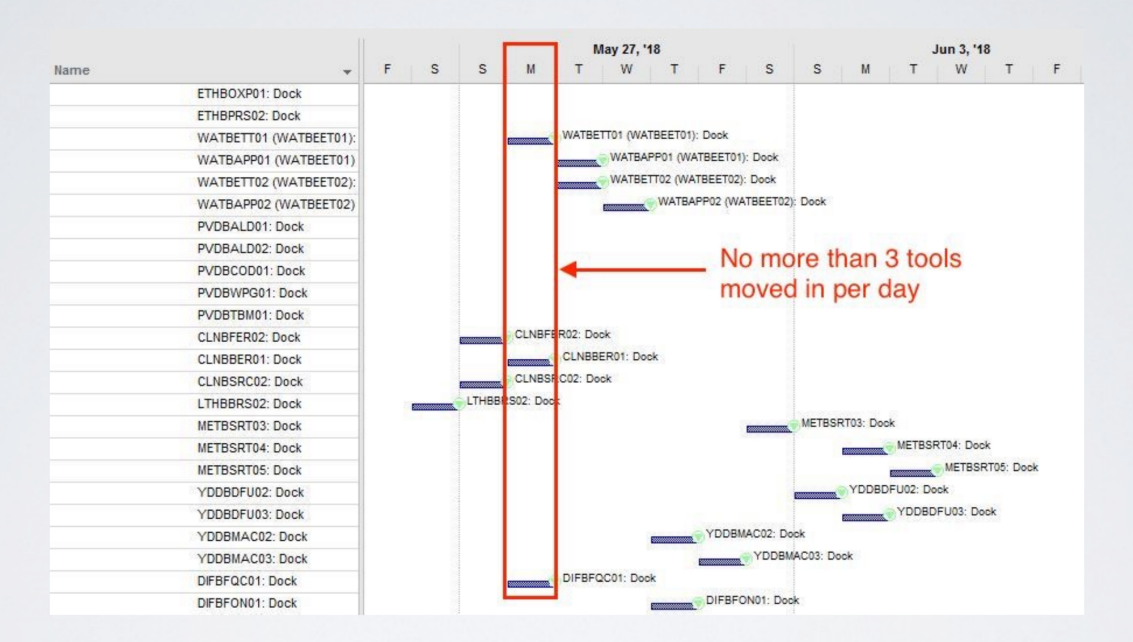


Figure 13. Dock loading, leveled. The boxed Monday shows the rule at work: no more than three tools moved in per day.

Utilities hookups were governed the same way. The facilities utilities matrix — which gases and chemicals each tool needs — lives in spreadsheets owned by facilities engineers. An importer reads the matrix, creates a dependency from each utility's availability milestone to the tool's Hookup 1, and cross-checks both directions: every tool in the matrix exists in the schedule, every gas and chemical in the schedule exists in the matrix [2]. Hand-entering thousands of these dependencies would have been slow the first time and

wrong by the second month.

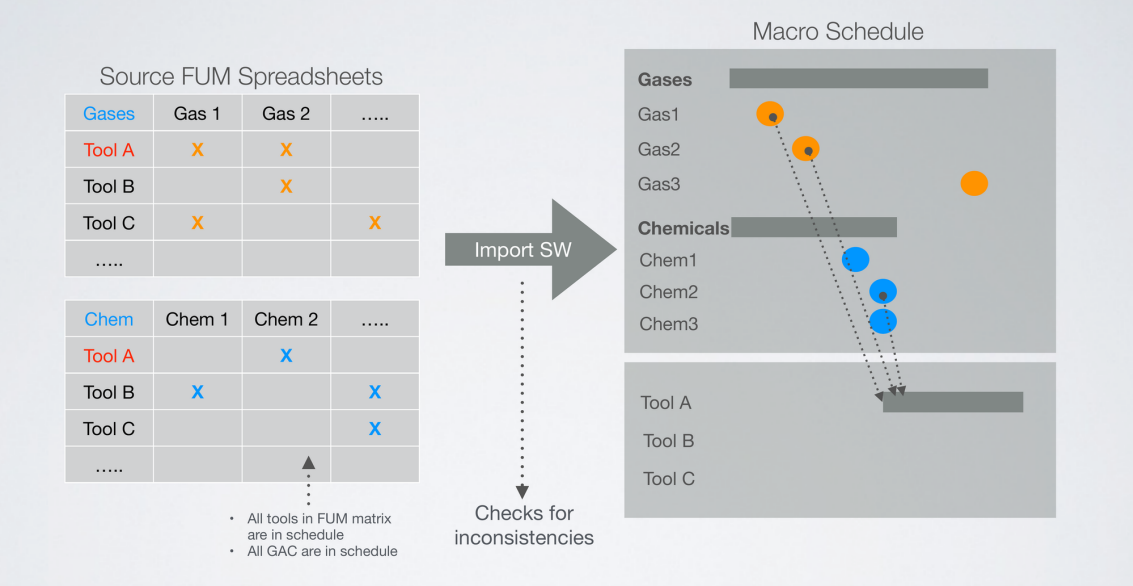


Figure 14. The utilities matrix importer. Source spreadsheets map tools to gases and chemicals; the importer builds the dependencies in the macro and checks for inconsistencies in both directions.

Outputs the organization actually read

The same schedule that drove the network drove the reporting. A tool dashboard exported straight from the live data listed every tool against every stage gate, shaded green where predicted dates held their targets and red where they did not [2]. Because the export came from the network rather than from a hand-maintained tracker, the dashboard was current on the day of every roll-up, and arguments about whose spreadsheet was right did not happen.

Tool ID	Module	Negotiation + CEPD	Obtain Approval to Raise PR	PR	PO	Leadtime	Hookup 0	Target Dock Date	Dock Date	Move-In Date	Hookup 1	Hookup 2	Supplier Qual Tier 1	Qual Tier 2	Tool Required Date	Tool Release Date
CLNBCST02	CLEAN	6/12/2017	7/10/2017	7/31/2017	8/7/2017	2/3/2018	5/11/2018	5/11/2018	5/11/2018	5/14/2018	5/17/2018	5/21/2018	5/28/2018	6/11/2018	6/12/2018	6/12/2018
CLNBCST01	CLEAN	6/12/2017	7/10/2017	7/31/2017	8/7/2017	12/5/2017	4/19/2018	4/19/2018	4/19/2018	4/22/2018	4/25/2018	5/7/2018	5/6/2018	5/14/2018	5/15/2018	5/15/2018
CLNBFER01	CLEAN	6/12/2017	7/10/2017	7/31/2017	8/7/2017	1/19/2018	4/29/2018	4/29/2018	4/29/2018	5/10/2018	5/13/2018	5/17/2018	6/10/2018	6/24/2018	6/25/2018	6/25/2018
CLNBSRC01	CLEAN	6/12/2017	7/10/2017	7/31/2017	8/7/2017	2/3/2018	5/10/2018	5/10/2018	5/10/2018	5/13/2018	5/16/2018	5/20/2018	6/6/2018	6/13/2018	6/14/2018	6/14/2018
LTHBBRS01	METRO	6/12/2017	7/10/2017	7/31/2017	8/7/2017	2/3/2018	4/19/2018	4/19/2018	4/19/2018	4/29/2018	5/2/2018	5/7/2018	5/22/2018	6/5/2018	6/15/2018	6/15/2018
METBCDS01	METRO	6/12/2017	7/10/2017	7/31/2017	8/7/2017	2/3/2018	4/27/2018	4/27/2018	4/27/2018	4/30/2018	5/3/2018	5/7/2018	5/18/2018	5/23/2018	5/24/2018	5/24/2018
METBTHK01	METRO	6/12/2017	7/10/2017	7/31/2017	8/7/2017	2/3/2018	4/19/2018	4/19/2018	4/19/2018	4/22/2018	4/25/2018	5/7/2018	5/10/2018	5/15/2018	5/20/2018	5/20/2018
YDDBMIC01	YDD	6/12/2017	7/10/2017	7/31/2017	8/1/2017	12/29/2017	6/24/2018	6/3/2018	6/3/2018	6/28/2018	7/2/2018	7/9/2018	7/14/2018	7/17/2018	6/27/2018	7/18/2018
DIFBFRP01	DIFFUSION	6/12/2017	7/10/2017	7/31/2017	8/7/2017	2/3/2018	5/4/2018	5/4/2018	5/4/2018	5/6/2018	5/9/2018	5/13/2018	6/23/2018	7/7/2018	7/16/2018	7/16/2018
IMPBHEI01	IMPLANT	6/12/2017	7/10/2017	7/31/2017	8/7/2017	2/3/2018	5/2/2018	5/2/2018	5/2/2018	5/6/2018	5/9/2018	9/1/2018	6/23/2018	9/22/2018	8/24/2018	9/23/2018
METBQTX01	METRO	6/12/2017	7/10/2017	7/31/2017	8/7/2017	2/3/2018	4/25/2018	4/25/2018	4/25/2018	4/28/2018	5/1/2018	5/7/2018	5/16/2018	5/21/2018	5/31/2018	5/31/2018
METBSRT01	METRO	6/12/2017	7/10/2017	7/31/2017	8/7/2017	2/3/2018	4/20/2018	4/20/2018	4/20/2018	4/23/2018	4/26/2018	5/7/2018	5/5/2018	5/10/2018	5/11/2018	5/11/2018
METBSRT02	METRO	6/12/2017	7/10/2017	7/31/2017	8/7/2017	2/3/2018	5/11/2018	5/11/2018	5/11/2018	5/14/2018	5/17/2018	5/21/2018	5/26/2018	5/29/2018	5/30/2018	5/30/2018
YDDBMAC01	YDD	6/12/2017	7/10/2017	7/31/2017	8/7/2017	2/3/2018	5/30/2018	5/30/2018	5/30/2018	6/2/2018	6/5/2018	6/9/2018	6/20/2018	6/25/2018	7/5/2018	7/5/2018
YDDBMAC02	YDD	6/12/2017	7/10/2017	7/31/2017	8/7/2017	2/3/2018	5/31/2018	5/31/2018	5/31/2018	6/3/2018	6/6/2018	6/10/2018	6/21/2018	6/26/2018	7/5/2018	7/6/2018

Green shading = On or ahead of Target date
Red shading = Later than Target date

Figure 15. The tool dashboard, exported from the schedule. Green: on or ahead of target. Red: later than target. Every date traces back to the live network.

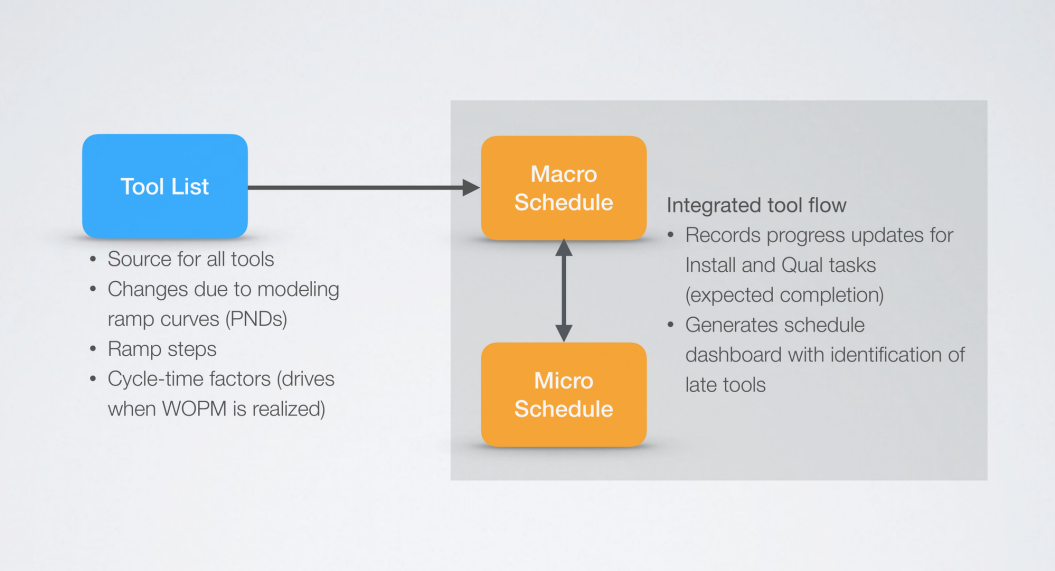


Figure 16. Data sources. The IE tool list — ramp steps, cycle-time factors, ramp-curve changes — feeds the macro; macro and micro exchange progress through the roll-up; the dashboard identifies late tools.

05

Section 05
The operating rhythm

Architecture set the structure; the operating rhythm made it move. Fab8 ran eight process-module micro-schedules, and rolled the program up twice a day, so no slip stayed invisible longer than half a working day [2]. Most programs run the same loop weekly; at $5 million a day of delay cost, Fab8 could justify the cadence [1].

The refresh loop itself is short [3]. The program team updates the macro. Each module team scrubs and refreshes its own micro-schedule — progress, remaining durations, re-sequencing — on its own machine, with no coordination required. The roll-up then runs: dates roll down, durations and dates roll up, cross-micro dependencies carry over, and the macro recalculates its network. The program team reads the new critical path, and where a roll-up task sits on it, drills into the micro to hunt for pull-ins. If the hunt changes anything, the roll-up runs again. A wigglechart tracks the trend of the Risk Start milestone across refreshes — the early-warning instrument that shows drift while there is still time to act on it.

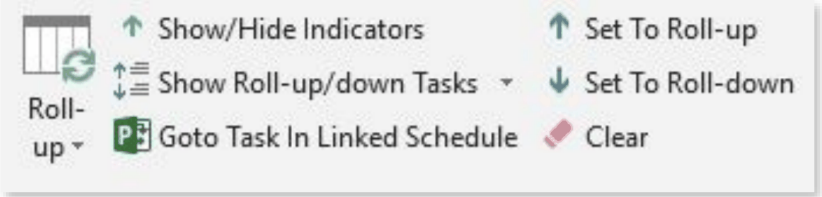
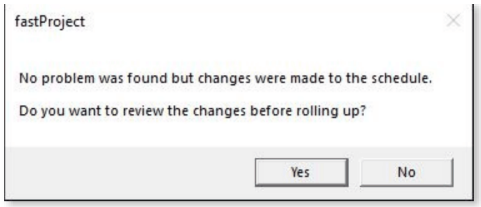


Figure 17. The Macro-Micro Roll-up group in the fastProjectAI ribbon: roll-up, indicator columns, roll-up/down filters, Goto Task In Linked Schedule, and the set/clear roll-up flags.

Guardrails on every pass

A synchronization that dozens of people depend on cannot be a black box, so every roll-up is verified and audited. Before changing anything, the software validates the whole file set against the setup rules — matching task names, unique names, auto-scheduling, dependency consistency — and stops with an error report if anything fails [3]. When it does make changes, it can list every one: which schedule, which task, what changed. A roll-up summary quantifies the damage and the good news alike (original and new duration, old and new finish, net slip or pull-in per activity), and a statistics view records files processed and time taken.



Summary of Changes

Schedule	Task IDs	Activity	Change
Macro schedule AFTER.mpp	3	Test iteration 1	Marked as roll-down
Macro schedule AFTER.mpp	7	Test iteration 2	Marked as roll-down
Macro schedule AFTER.mpp	2	Develop iteration 1	Task marked as roll-up
Macro schedule AFTER.mpp	6	Develop iteration 2	Task marked as roll-up
Macro schedule AFTER.mpp	10	Develop iteration 3	Task marked as roll-up
Micro development schedule AFTER.mpp	2 > 12	Test iteration 1 > Redesign	Added dependency with a lag of -50%
Micro development schedule AFTER.mpp	4 > 12	Test iteration 2 > Redesign	Added dependency with a lag of -50%
Micro development schedule AFTER.mpp	2	Test iteration 1	Added Roll-down milestone (is a predecessor to Develop iteration 2)
Micro development schedule AFTER.mpp	4	Test iteration 2	Added Roll-down milestone (is a predecessor to Develop iteration 3)

Figure 18. The audit trail. After validation, the roll-up lists every change it is about to make — schedule, task, and action — before the user commits.

Roll-up Summary

Macro schedule AFTER.mpp — Generated: 6/6/2018 03:06 PM

Activity	Micro-schedule	Original Duration (days)	New Duration (days)	Change in Duration (Reduced)/Increased	Old Start Date	Old Finish Date	New Start Date	New Finish Date	Net Change in Finish Date (Pulled-in)/Slipped
Develop iteration 1	Micro development schedule AFTER.	60	210	150	6/5/2018	8/3/2018	6/5/2018	12/31/2018	150
Test iteration 1	Micro development schedule AFTER.	21	21		8/4/2018	8/24/2018	1/1/2019	1/21/2019	150
Develop iteration 2	Micro development schedule AFTER.	30	123	93	8/14/2018	9/13/2018	1/22/2019	5/24/2019	253
Test iteration 2	Micro development schedule AFTER.	14	14		9/13/2018	9/27/2018	5/24/2019	6/7/2019	253
Develop iteration 3	Micro development schedule AFTER.	30	136	106	9/20/2018	10/20/2018	6/7/2019	10/21/2019	366

Figure 19. The roll-up summary: per-activity change in duration and finish date, with net slip or pull-in. The first reconciliation of a top-down plan against bottom-up reality is rarely flattering — here the widget program's end date moved out nine months.

The practices around the tool

The roll-up automated the mechanics, and a set of Fast-Time-to-Market practices supplied the behavior [2]:

- Target dates are driven by the business need; the working schedule is built bottom-up from the team's own estimates, not dictated down.
- A cross-functional team — one member per organization — scrubs the macro-schedule every week.
- Bottleneck tools are named, and the gap between target and predicted dates is worked as a pull-in backlog, not just reported.
- The management team meets weekly to look for strategic pull-ins; module teams run daily refresh and pull-in meetings at their own level.
- Schedule trends for the Risk Start milestone are tracked as an early warning, so problems surface before the fact, not after.
- Schedules and dashboards publish weekly from live data — no filtering, no re-typing, no massaging on the way up the hierarchy.

In most organizations, schedule data degrades as it climbs: each layer summarizes, softens, and delays it. Publishing the live network removed the editorial layer entirely — the executive review read the same data the module teams entered that morning.

06

Section 06
What it delivered

Fab8 reached first silicon two weeks ahead of schedule [1]. On a program whose team costed delay at roughly $5 million a day, two weeks is on the order of $70 million — earned not by heroics in the last quarter but by years of finding pull-ins a few days at a time.

The wigglechart tells that story in one line [12]. Every refresh plotted the predicted finish of First Silicon Starts, and the first bottom-up reconciliation in early 2010 showed the milestone running almost two years late. Twenty-six months of refresh-and-pull-in worked the prediction back to the line, and a final 10-day pull-in (tool processing time reduced) completed the milestone on December 27, 2011 — fourteen days ahead of the January 10, 2012 target.

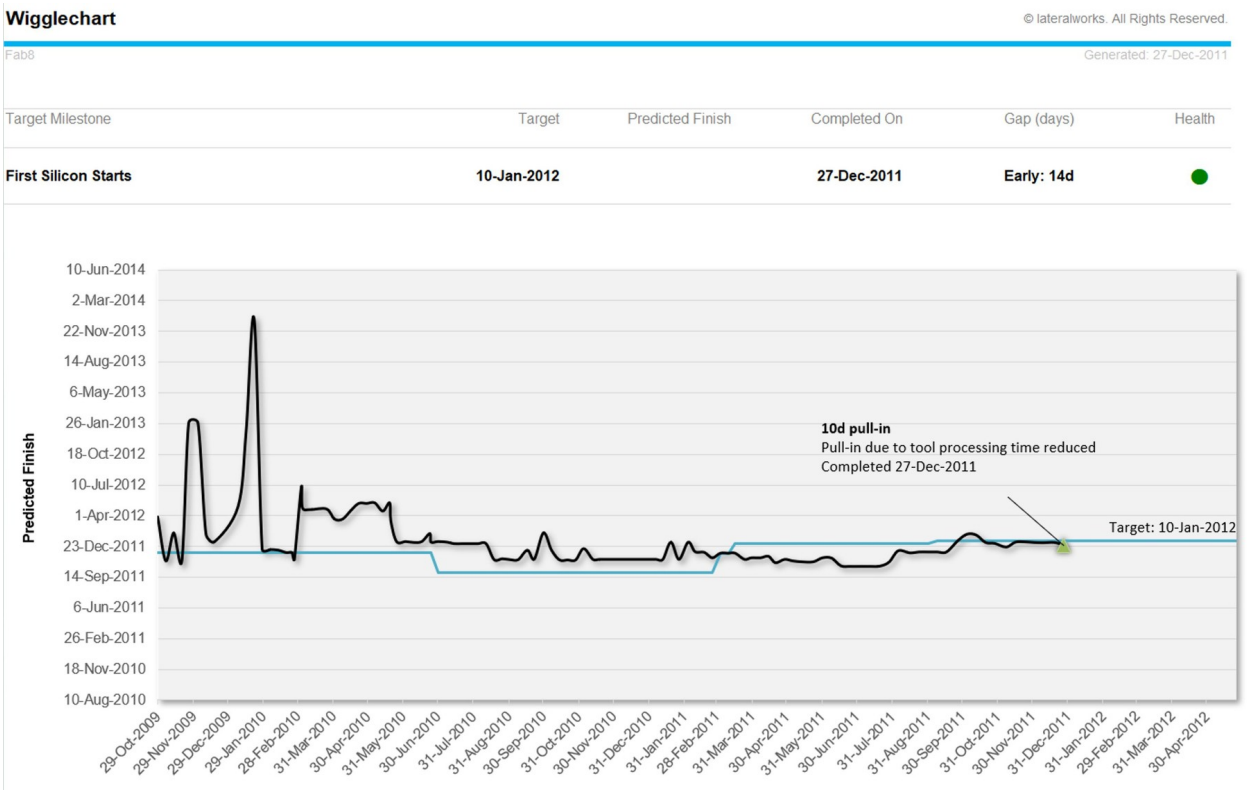


Figure 20. The Fab8 wigglechart for First Silicon Starts, generated December 27, 2011 [12]. Black: the predicted finish at each schedule refresh. Blue: the target. The early spike is the near-two-year delay the first reconciliation revealed; the milestone completed 14 days early.

The program closed carrying more than 100,000 activities in its combined schedules. Eight module micro-schedules, each owned end-to-end by its module team. Two roll-ups a day, every working day, each one validated, audited, and recalculated into a fresh program critical path. All of it ran on Microsoft Project —

a tool whose own master-subproject machinery was never designed to model a program of this sophistication, and whose documented failure modes at far smaller scale [5, 6, 7] are the reason the roll-up bypasses that machinery entirely.

The number was achievable because every level of the program had a schedule sized to its decisions. Management read a macro measured in hundreds of tasks and made pull-in decisions against a real critical path. Module managers ran schedules measured in thousands of tasks and answered for their own dates. The roll-up guaranteed the two never drifted apart. Speed came from the combination: distributed ownership with a single source of program truth.

A quieter result: the architecture outlived the program. The custom code written for the fab became the productized Macro-Micro Roll-up released with fastProject in 2019, and the same pattern has since run greenfield fabs, product programs, and — at the small end — three-schedule development efforts. Fab8 was the proof at maximum difficulty.

07

Section 07

The same tool at normal scale

Nothing in the macro-micro architecture requires 100,000 activities. The same structure runs a program of three schedules, and does so with the same division of labor: PMs own their detail, the program owns the critical path.

A current example: a firmware and software stack integrated onto a hardware platform developed in China, then integrated and certified by the client. The setup is one macro plus two micros — a FW/SW micro-schedule and a build micro-schedule, three .mpp files in all. The macro is created first and decomposed top-down. During the week each PM scrubs and refreshes their own micro independently; the weekly roll-up rebuilds the program schedule, and the program critical path reads directly in the macro. Dates roll down from the macro into the micros when the program re-plans. From team formation to certification took about twelve months — fast, given the complexity of the software stack and the certification requirements.

The published Project Everest case study applies the same architecture to a utility-scale battery storage platform — an integrated core team running hardware, firmware, and flagship-customer delivery schedules against one program critical path [4]. Between Everest and Fab8 sits the full range: the tool does not care whether the micros number two or eight, or the activities three thousand or one hundred thousand.

Where it pays, and where it does not

The guidance from section 01 stands. A small or medium program with one PM and one team belongs in a single schedule; the roll-up would add ceremony without adding control. Macro-micro starts paying when more than one team needs to own detail independently (separate companies, time zones, or functions) while management still needs one critical path it can read and act on. It also demands something the simpler methods do not: the schedule architecture must be designed top-down before the roll-up is switched on. Task names, hierarchy, and roll-up boundaries have to be thought through in advance; retrofitting them onto an organically grown schedule is possible (the Convert to Macro and Convert to Micro functions exist for exactly that [3]), but it is real work, not a settings change. This is not a technique for a quick meeting; it is a structure a program commits to.

lateralworks implements Macro-Micro Roll-up only through consulting engagement and user training, for the same reason a good structural engineer will not mail you a bridge design: the architecture decisions at the start determine everything that follows [3].

A

Appendix A
Rules and rhythm

The rules below are the working discipline behind everything in this case. They are short because the software enforces most of them; they are strict because a file set behaving as one model leaves no room for improvisation [3].

Setup rules for a macro-micro roll-up

- Task names are identical in macro and micro, including capitalization and spacing.
- Task names are unique — one version of each name across the entire file set.
- Roll-up tasks are auto-scheduled, never manually scheduled.
- A roll-up task in the micro may be a task or a summary task; its counterpart in the macro is always a task.
- Roll-up tasks at both levels live inside a summary-milestone group.
- A dependency from a touchpoint to a roll-up task in the micro must also exist in the macro.

Remaining consistency rules — matching roll-up and roll-down flags across levels, predecessors of roll-up tasks existing in the micro, dependencies between roll-up tasks existing at both levels — are handled automatically when the roll-up runs with automatic setup enabled, and reported as errors when they cannot be [3]. Use one calendar convention across every schedule in the set; a 7-day calendar works best on programs whose tasks run through weekends.

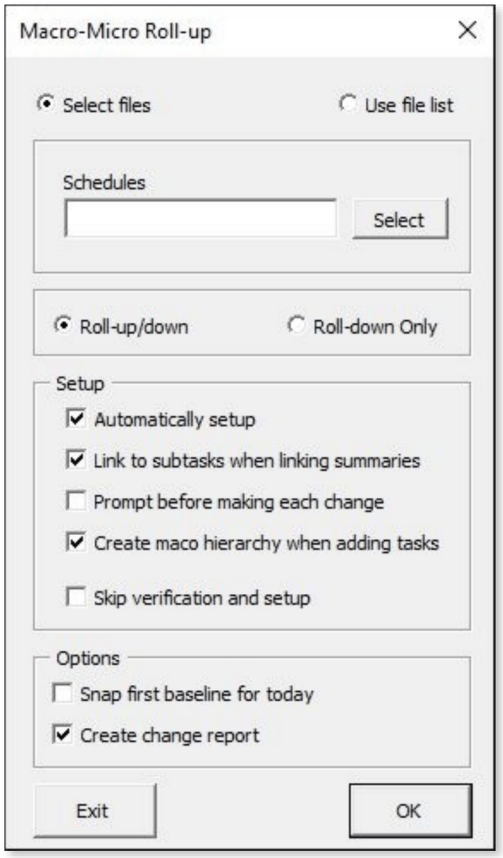


Figure 21. The roll-up dialog. Defaults are deliberate: verification on, automatic setup on, change report on. The normal case is three clicks — Roll-up, select schedules, OK.

The weekly refresh cycle

- Update the macro-schedule (skip roll-up tasks — the micros own them).
- Each team refreshes its micro-schedule (skip roll-down tasks — the macro owns them), pulls in what it can, and runs its wigglechart.
- Run the Macro-Micro Roll-up: latest dates roll down, latest micro dates roll up.
- Hunt pull-ins in the macro: on any roll-up task on the critical path, use Goto Task In Linked Schedule and work the detail at the micro level.
- If anything changed, re-run the roll-up.
- Run the wigglechart at the macro — and at the micros if needed.

At Fab8 this loop ran twice a day rather than weekly, with eight module schedules in the set. The cycle is the same; only the clock speed changes.

Sources

References

[1] lateralworks. "Results — client projects." lateralworks.com/results. GlobalFoundries Fab8, Malta, New York: $7 billion greenfield fab; first silicon two weeks ahead of schedule; program cost of delay approximately $5M/day. https://lateralworks.com/results

[2] lateralworks. "Macro-Micro Schedule Architecture — Greenfield Fab Project." Program documentation and engagement records, lateralworks archive. Schedule structure, tool template, touchpoints, dock leveling, utilities matrix importer, and FTTM practices for greenfield fab startups. Fab8 engagement records: eight process-module micro-schedules, roll-ups twice daily, more than 100,000 activities in the combined schedules.

[3] lateralworks. fastProject Macro-Micro Roll-up User Guide, version 2.0. lateralworks, 2019. Methods for managing large programs, setup rules, roll-up interface, top-down and bottom-up case studies, weekly refresh cycle. The Macro-Micro Roll-up function released with fastProject in Q2 2019; the product line is fastProjectAI today.

[4] lateralworks. "Project Everest — lateralworks case study." lateralworks.com, 2026. Macro-micro program architecture applied to a utility-scale battery energy storage platform.

[5] Microsoft. "Plans within plans: master projects and subprojects." Microsoft Project support documentation. https://support.microsoft.com/en-us/office/plans-within-plans-master-projects-and-subprojects-35b02e56-0101-4eca-ac33-82d8392d119b

[6] Microsoft Q&A.; "Microsoft Project application crash when opening a project file that contains links to other project files." learn.microsoft.com/en-us/answers/questions/5917678/

[7] Microsoft Q&A.; "Subprojects become inaccessible from master project: ‘we can’t insert or expand this project’." learn.microsoft.com/en-us/answers/questions/4878680/

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

[9] McChrystal, S., Collins, T., Silverman, D., and Fussell, C. Team of Teams: New Rules of Engagement for a Complex World. Portfolio/Penguin, 2015.

[10] Kelley, J. E., and Walker, M. R. "Critical-Path Planning and Scheduling." Proceedings of the Eastern Joint Computer Conference, 1959, pp. 160-173.

[11] U.S. Government Accountability Office. Schedule Assessment Guide: Best Practices for Project Schedules. GAO-16-89G, December 2015. https://www.gao.gov/products/gao-16-89g

[12] lateralworks. "GlobalFoundries Fab8 — Building a $7 Billion Semiconductor Fab in Under Two Years." lateralworks case study, lateralworks.com. Source of the Fab8 site photograph and the First Silicon Starts wigglechart.

A note on the illustrations. All schedule screenshots and diagrams are cropped from lateralworks program documentation and the fastProject user guide. They show tool IDs and task names only; no individual appears by name. Where an example predates or postdates Fab8, it illustrates the same architecture as deployed on lateralworks greenfield-fab engagements.