90–90 Rule

Idea In Short

The 90-90 Rule, a programming aphorism from Bell Labs engineer Tom Cargill, holds that the first 90% of a project's work takes 90% of the allotted time and the remaining 10% takes another 90%. The math does not add up and that is the point: it captures how integration, edge cases and stakeholder sign-off routinely blow past even generous estimates. For executives, the lesson is not to pad schedules further but to change what gets measured. Track completion by verified, tested outcomes rather than by lines written or tasks touched and treat the final stretch of any initiative, a system cutover, a merger integration, a product launch, as a distinct phase with its own budget, owner and contingency. Projects that fail rarely fail in the middle. They fail in the last mile, when the easy 90% is done and the hard 10% is still ahead.

Tom Cargill, an engineer at Bell Labs, is credited with a line that has outlived nearly every formal estimation method built to replace it: the first 90% of the code accounts for the first 90% of the development time and the remaining 10% accounts for the other 90% of the development time.1 The arithmetic does not close and it was never meant to. It is a joke that describes a real pattern: a team can look 90% finished for months while the actual finish line keeps receding. What started as commentary on software timelines has since become shorthand for a much broader phenomenon in consulting, product development and corporate transformation, where the last portion of a complex undertaking consumes far more time, budget and attention than the bulk of the work that preceded it.

Where the Rule Came From

Cargill's line first circulated inside Bell Labs in the early 1980s as an inside joke among engineers who had watched project after project stall near the finish. Jon Bentley gave it a wider audience in September 1985, when he included it in his "Programming Pearls" column in Communications of the ACM under the heading "Rule of Credibility", using it to warn readers against trusting confident schedule estimates. The rule caught on because every

experienced developer recognized the shape of it immediately, even if they had never heard it stated so precisely. A codebase can compile, pass early tests and demo well long before it handles the exceptions, load conditions and integration points that determine whether it actually ships.

The staying power of the joke comes from how well it survives outside its original context. Replace "code" with "deliverable", "system", or "product" and the pattern still fits: consultants who have built 90% of a model, manufacturers who have assembled 90% of a plant and regulators who have reviewed 90% of a filing all report the same experience, a long, steady middle followed by a final stretch that refuses to move on schedule. The humor works because it is specific enough to be useless as a formula and accurate enough to be immediately familiar.

> The first 90 percent of the code accounts for the first 90 percent of the development time. The remaining 10 percent of the code accounts for the other 90 percent of the development time

That specificity matters for how the rule gets applied today. It is not a forecasting tool and nobody should try to use it as one. It is a warning against a particular kind of overconfidence, the belief that steady early progress predicts a steady finish.

Why the Final Stretch Behaves Differently

The mechanism behind the pattern is not mysterious once it is named. Early work on any complex project tends to address the problems that are well understood, the requirements that are clear and the components that can be built in isolation from everything else. That work is genuinely 90% of the volume and it genuinely does take roughly the time allotted for it. What remains is not a smaller version of the same work. It is a different kind of work entirely:

> the exceptions nobody scoped, the integration points between systems built by separate teams, the edge cases that only appear once real data or real users touch the system and the approvals that were quietly waiting on other approvals to finish

first

Research on large-scale technology delivery backs this pattern with numbers rather than anecdote. An analysis of more than 5,400 large information technology projects found that half of those with initial budgets above $15 million ran over budget by an average of 45%, over schedule by 7% and delivered 56% less value than promised, with each additional year of duration adding roughly 15% more cost overrun on top.2 Those overruns rarely show up at the midpoint of a project. They surface once integration begins, once a stakeholder group that was quiet through the build phase gets its first look at the finished product and asks for changes, or once a dependency that everyone assumed would be ready turns out not to be.

Levi Strauss and the Cost of a Late Surprise

The mechanism is visible in specific, well-documented failures. Levi Strauss began an enterprise resource planning migration with a budget under $5 million and a perceived risk profile low enough to treat as routine. The project ran into unexpected integration problems with Walmart's supply chain systems, disrupted distribution operations badly enough to force a week-long shutdown of distribution centers and ultimately cost the company a $192.5 million charge against earnings.3 Nothing about the early phases of that project predicted the outcome. The integration work, the part that connects a system to everything around it, is exactly where the 90-90 pattern concentrates its cost.

Rework as the Hidden Multiplier

A second mechanism compounds the first: rework accumulated earlier in a project tends to surface and demand correction, precisely during the final phase. Analysis of large software efforts has found that specialists routinely spend 40% to 50% of their time on avoidable rework rather than new, value-added work and that the cost of fixing a defect discovered late in a project, or after deployment, can run roughly 100 times higher than fixing the same defect during design.4 A defect introduced early and left unaddressed does not disappear. It waits, often compounding as later work builds on top of it, until integration or testing forces it into view, at which point fixing it costs far more than it would have earlier and lands squarely inside the phase that was supposed to be the easy 10%.

Distinguishing the 90-90 Rule From the Planning Fallacy

It helps to separate the 90-90 Rule from a related but distinct concept: the planning fallacy, the well-documented tendency for people to underestimate how long a task will take even when they know their own track record of underestimating. Executives evaluating major investments, from acquisitions to capital projects, consistently rely on an inside view, a forecast built from the specific plan in front of them, rather than an outside view built from how similar projects have actually performed historically. The Project Management Institute's own annual survey of practitioners found that even organizations with strong project discipline still saw scope creep on 28% of projects and lost an average of 17% of budget on projects that failed outright, evidence that the shortfall persists across well-run portfolios, not just troubled ones.5 The planning fallacy explains why total estimates run short across the board. The 90-90 Rule describes something more specific:

> where in the schedule that shortfall tends to concentrate

The distinction is practical, not academic. A team that only corrects for the planning fallacy will pad every phase equally, adding a uniform contingency across the whole schedule. That approach wastes contingency on phases that were already estimated correctly and still underfunds the phase that actually needs it. A team that understands the 90-90 pattern instead concentrates its contingency, its senior attention and its most experienced people on the final phase specifically, because that is where the risk is concentrated, not spread evenly across the calendar.

Measuring Progress Without Fooling Yourself

Modern estimation practice has developed partial defenses against the pattern, though none of them eliminate it outright. Agile teams that use story points instead of calendar estimates do so precisely because time-based estimates hide uncertainty behind a false precision and story points force a team to confront how much of a task's difficulty is still unknown.6 That shift helps, but it does not remove the underlying problem:

> a team can still burn through 90% of its story points while the remaining 10% carries

90% of the actual risk, particularly on features that depend on systems the team does not control

The more durable fix is a change in what gets measured. Percentage-complete metrics based on tasks touched, hours logged, or lines written give a comforting but misleading picture, because they treat every unit of work as equivalent regardless of how much residual risk it carries. A dashboard that shows 90% task completion says nothing about whether the remaining 10% is trivial cleanup or the single integration point the entire system depends on. Replacing that metric with one based on verified, tested outcomes, features that have passed integration testing, contracts that have cleared legal review, systems that have processed real transactions without failure, gives leaders a much more honest signal, even when that signal is less flattering.

- Track completion against tested, verified outcomes, not tasks opened or hours logged
- Give the final phase its own named owner, budget and schedule, distinct from the rest of the project
- Pull integration and validation work as early into the schedule as the architecture allows
- Reserve a disproportionate share of contingency and senior staff time for the last phase specifically

None of these steps eliminate the pattern Cargill described. They convert it from a recurring surprise into a known and budgeted risk, which is the most a project team can reasonably expect to do.

Applying the Lesson Outside Software

Consultants running transformation programs see a close cousin of the 90-90 pattern on nearly every engagement that involves changing how an organization actually works, not just what it says on paper. A cost-reduction program can identify and approve 90% of its savings targets in the first several months, then spend the remaining months negotiating the handful of changes that touch union agreements, customer contracts, or executive

compensation, the changes everyone deferred because they were hardest. A merger integration can align 90% of systems and processes on schedule, then spend disproportionate time on the finance close process or the customer data migration that neither company had fully documented.

The organizations that handle this well share a habit: they treat the final phase of any major initiative as a distinct project in its own right, with its own risk register, its own sponsor and its own realistic timeline built from what the remaining work actually requires, not from what percentage of the calendar remains. That discipline will not make the last 10% easy. It will keep an organization from being surprised, again, by how long easy work turns out not to have been.

- 1The Ninety-Ninety Rule
- 2Delivering Large-Scale It Projects On Time, On Budget And On Value
- 3Why Your It Project May Be Riskier Than You Think
- 4Why Software Fails
- 5Pmi Pulse Of The Profession 2023
- 6Agile Project Estimation

Summary

The 90-90 Rule endures because it names a failure pattern that reappears in nearly every complex undertaking: steady, visible progress through the bulk of the work, followed by a final stretch that resists every deadline set for it. The cause is not laziness or bad luck. It is that the last portion of any project concentrates the parts nobody solved earlier, the integration points, the exceptions, the approvals that depend on other approvals. Executives who plan around this pattern, by tracking verified completion instead of activity, by budgeting the final phase separately and by pulling integration testing earlier rather than later, convert a recurring surprise into a manageable risk. Those who ignore it keep discovering, one project at a time, that 90% done and finished are not the same claim.