SUPPLY CHAIN RESEARCH · EDITORIAL FEATURE

The Change Management Deficit

Why the Largest Cause of Software Failure Gets the Smallest Budget

The software works. The integration holds. The data is adequate. And the program still fails, because the people who were supposed to use it did not. This is the deficit at the center of enterprise technology, and it is the one every budget underfunds.

Independent, vendor-neutral research for supply chain and technology leaders
Supply Chain Research | supplychainresearch.com | 2026

Contents

A decision framework for the AI era

01 Executive summary 01
02 The deficit, stated plainly 02
03 The evidence: people, not technology 03
04 Go-live is not adoption 04
05 Why the budget never matches the risk 05
06 The axe falls upward 06
07 Resistance is rational 07
08 The sequence of individual change 08
09 Sponsorship: the single greatest predictor 09
10 The customization trap 10
11 What the famous failures actually teach 11
12 Training is not change management 12
13 How to resource it properly 13
14 Measurement, governance, and a scoring rubric 14
15 Conclusion: fund the people 15
16 Methodology, caveats, and sources 16

01 Executive summary

There is a finding in the research on enterprise software failure so consistent, so well documented, and so thoroughly ignored that it deserves to be stated at the outset without hedging. The software is almost never the reason these programs fail. The systems involved are proven, are used successfully by thousands of organizations, and do broadly what they claim to do. What fails is the organization: the people who were supposed to change how they work and did not, the processes that were supposed to be redesigned and were not, and the sustained effort of getting several thousand human beings to abandon a way of working they understand for one they do not. That effort is called change management, it is repeatedly identified as the single largest cause of failure, and it is routinely allocated a single-digit percentage of the program budget and cut first when the schedule slips.

This is the change management deficit, and it is the most predictable and most avoidable failure in enterprise technology. We say honestly that the discipline suffers from a credibility problem it has partly earned: it is sold by consultancies with an obvious interest, it is described in vocabulary that invites cynicism, and it has produced its share of expensive workshops that accomplished nothing. None of that alters the evidence. Programs with strong change management meet their objectives at a dramatically higher rate than those without. Most large system implementations fail to deliver the return that justified them, and the cited reasons are overwhelmingly about adoption rather than architecture. This article sets out what the discipline actually is, strips away the jargon that has made it easy to dismiss, and specifies what it costs, how to resource it, and how to tell whether it is working.

~42%
of enterprise system failures attributed to inadequate change management, the largest single cause

6x
more likely to meet objectives, for programs with excellent change management

55-75%
of ERP implementations that fail to achieve their expected return, driven by low adoption

The argument in brief

- **The failure is human, not technical.** Across independent analyses, inadequate change management is the largest single cause of enterprise system failure, and the software is rarely the culprit.
- **The budget is inverted.** The leading cause of failure receives a single-digit share of the program budget, and is the first line cut when the schedule slips, which is precisely why it remains the leading cause.
- **Go-live is not adoption.** The change team is usually disbanded at the moment the adoption work actually begins, and users revert to spreadsheets, which is how a system goes live and delivers nothing.
- **Resistance is information.** Some resistance is fear, and some is accurate intelligence from the people closest to the work. A program that treats all of it as an obstacle discards its most valuable signal.

- **Sponsorship is the variable that matters most.** Active, visible sponsorship by a senior executive is the strongest predictor of success in the research, and it cannot be delegated to a program manager.

02 The deficit, stated plainly

Begin with a definition, because the term has been so thoroughly colonized by consulting vocabulary that it has stopped meaning anything specific. Change management, in the sense that matters here, is the work of getting an organization to actually operate differently after a system goes live. It is not communication, though it involves communicating. It is not training, though it requires training. It is the sum of everything that has to happen for a planner who has used a spreadsheet for eleven years to stop using the spreadsheet, for a warehouse supervisor to trust a screen rather than a clipboard, for a buyer to accept a system recommendation they would previously have overridden, and for the finance team to close the books using a process they did not design and did not want.

Stated that way, the scale of the task becomes obvious, and so does the absurdity of funding it at eight percent of the program. An organization implementing an enterprise system is asking several thousand people, most of whom did not ask for the change, many of whom are good at the old way and will be temporarily bad at the new way, and some of whom correctly suspect the new way is worse for them personally, to abandon their working habits simultaneously and adopt new ones under production conditions. That is a hard thing to ask of one person. It is an extraordinarily hard thing to ask of an organization, and the fact that it is attempted routinely, on a schedule, with a fraction of the budget devoted to the software, explains most of what this article is about.

The deficit has a further, subtler dimension that is worth naming early. Technology programs are run by people who are good at technology, and technology is tractable in a way that human behavior is not. A configuration problem can be solved by a competent engineer in a defined period. An adoption problem cannot be solved by anyone in a defined period, because it depends on the choices of thousands of people whose incentives, fears, and workloads the program does not control. Faced with a tractable problem and an intractable one, a program under schedule pressure will attend to the tractable one, and will tell itself that the other will resolve at go-live. It does not resolve at go-live. It begins at go-live.

It should be said plainly that this discipline has earned some of the skepticism it attracts. It is sold by firms whose interest in its importance is not disinterested. Its literature is full of models with acronyms, and its practitioners sometimes speak in a register that invites mockery from engineers who are trying to ship something. A leader who has sat through a change management workshop that produced a communications plan and no change is entitled to be cynical. But the cynicism should be directed at the execution rather than the thesis, because the thesis survives every attempt to falsify it: the programs that invest in this succeed at a far higher rate than the programs that do not, and the ones that skip it fail for reasons that were entirely foreseeable and were, in fact, foreseen.

A useful way to see the deficit is to compare how the two halves of a program are treated when they encounter difficulty. If the integration is behind, the program adds engineers, escalates to the vendor, extends the timeline, and reports the risk to the steering committee, because everybody understands that an integration that does not work will stop the launch. If adoption readiness is behind, which is to say that the business units have not been engaged, the process design has not been agreed, and the people who will use the system have not been consulted, the program notes it in the amber column and

proceeds, because adoption readiness does not stop the launch. It stops the benefit, and the benefit is somebody else's problem, in a later quarter, in a report that nobody will trace back to this decision.

The second thing worth noticing is that the deficit is self-concealing. A program that underfunds change management does not fail visibly on the go-live date; it fails invisibly over the following year, in the form of a benefit that does not materialize. And because the benefit is attributed to the system rather than to the behavior, the organization concludes that the system underdelivered, and the next program is scoped to buy a better system. The lesson that would actually have helped is never learned, because the mechanism that produced the failure is never identified. Organizations therefore repeat this failure across successive programs, each time blaming the technology, each time buying different technology, and each time producing the same result.

The deficit also has a vocabulary problem that is worth confronting directly, because it costs the discipline credibility it cannot afford to lose. The field speaks of journeys, of hearts and minds, of bringing people along, and an engineer under deadline pressure who hears that language concludes, not unreasonably, that nothing of substance is being proposed. The remedy is to translate. Awareness means that people know why. Desire means that somebody has answered the question of what happens to them. Reinforcement means that the old system has been switched off and somebody is checking whether the new one is being used. Stated in that register, none of it sounds soft, and all of it sounds like work that obviously has to be done by somebody. The discipline would be funded far more often if it described itself in the language of the people who control the budget.

Consider, as a matter of arithmetic, what the organization is actually buying when it approves an enterprise system. It is not buying software; software is a means. It is buying a change in how several thousand people do their work, and the software is the instrument through which the change is delivered. Priced that way, the allocation looks absurd on its face: the organization has spent ninety-two percent of its budget on the instrument and eight percent on the change, which is the thing it actually wants. No other category of corporate investment is structured this way. A company opening a factory does not spend ninety-two percent on the machines and eight percent on hiring and training the people who will run them, because the absurdity would be obvious. In enterprise software it is not obvious, because the people already work there, and their willingness to work differently is assumed rather than purchased.

03 The evidence: people, not technology

The claim that failure is human rather than technical is not an assertion of preference. It is what the analyses of failed programs consistently find, shown in Figure 1. Across industry research into why enterprise system implementations miss their objectives, inadequate change management emerges repeatedly as the single largest attributed cause, at something in the order of forty percent, ahead of data migration problems, ahead of team capability, ahead of timeline compression, and far ahead of anything to do with the software.

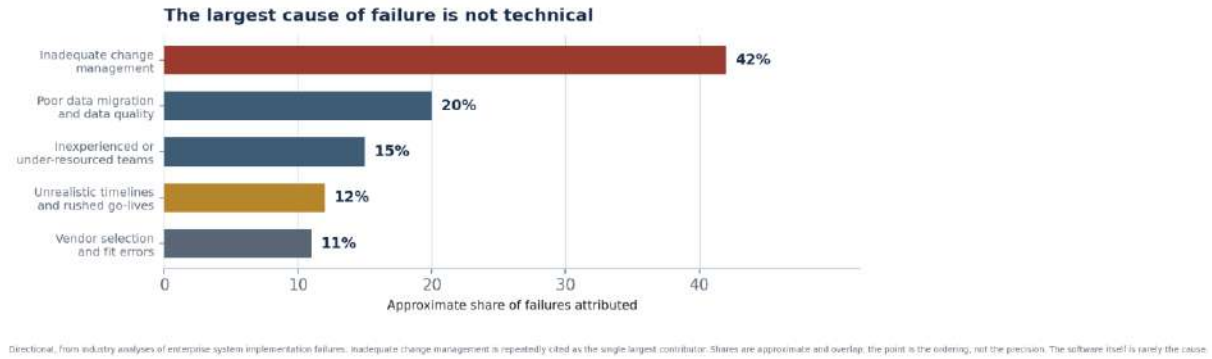

Figure 1. The attributed causes of enterprise system failure. The largest is organizational, and the software barely registers.

The corresponding positive finding is equally consistent, and equally ignored. Benchmarking research from the change management field reports that programs with excellent change management are roughly six times more likely to meet or exceed their objectives than programs with poor change management, shown in Figure 2. It is right to note that this figure comes from a firm that trains and certifies change management practitioners, which is not a disinterested source, and a careful reader should treat the precise multiple as directional. But the direction is corroborated by every independent analysis of failure causes, and the magnitude, whatever the exact number, is large enough that no executive approving a technology budget can reasonably ignore it.

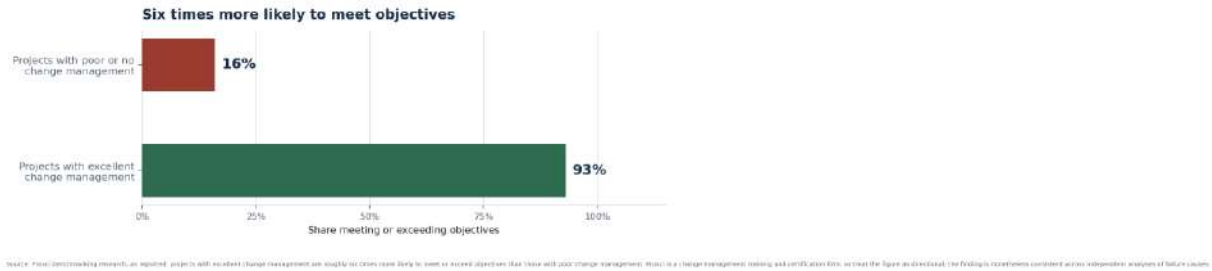

Figure 2. The reported effect of change management on whether a program meets its objectives. The source is an interested party; the direction is corroborated everywhere.

The consequence appears at the other end of the program, in the returns that never arrive, shown in Figure 3. Analysts have been cited as finding that somewhere between fifty-five and seventy-five percent of enterprise resource planning implementations fail to achieve their expected return on investment, and the causes given are low user adoption, weak data governance, and inadequate post-implementation optimization, all of which are organizational rather than technical. More pointedly for readers of this publication, a major analyst firm has predicted that sixty percent of supply chain digital

adoption efforts will fail to deliver their promised value by 2028, and has attributed the failure specifically to insufficient investment in learning and development. That is a prediction about people, made about a technology market, by a technology analyst.

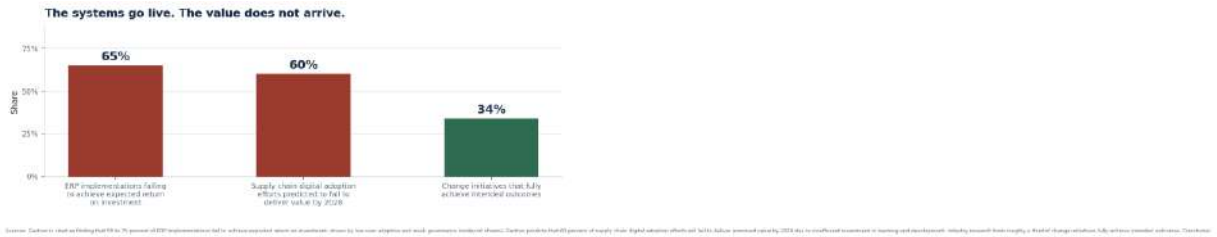

Figure 3. The value that does not arrive. Systems go live and returns do not, and the reasons cited are consistently about adoption rather than architecture.

Set these findings alongside one another and the picture is unambiguous. The largest cause of failure is organizational. The intervention that addresses it has the largest measured effect on success. The failure of the intervention produces exactly the outcome that the majority of programs report, which is a system that works and a business that did not change. And yet the intervention remains the smallest line in the budget and the first one cut. That is not a research gap. It is a management failure, repeated at scale, in full view of evidence that has been available for two decades.

There is a further piece of evidence that deserves attention because it comes from a technology analyst rather than from a change management firm, and therefore carries no commercial interest in the conclusion. The prediction that a majority of supply chain digital adoption efforts will fail to deliver their promised value by 2028 is notable not for the number but for the stated cause: insufficient investment in learning and development. A firm whose business is advising organizations on which technology to buy has concluded that the binding constraint on value is not which technology they buy but whether anybody learns to use it. When the analysts who sell technology research are telling the market that the problem is not the technology, the market should probably listen.

It is also worth confronting the objection that these figures are inflated by definitional slack, since failure is a word that can be stretched. The objection has some force: a program that goes live six weeks late and eventually delivers most of its benefit is recorded as a failure by some methodologies and as a success by others. But the objection cuts less deeply than it appears to, because the finding that matters is not the absolute failure rate; it is the relative ordering of causes. Whatever definition of failure is applied, and whichever study is consulted, organizational causes outrank technical ones by a wide margin. An organization may reasonably dispute whether the failure rate is forty percent or seventy. It cannot reasonably dispute that when these programs go wrong, they go wrong because of people.

It is worth being precise about what change management is not, since much of the skepticism it attracts is aimed at things it should never have been confused with. It is not a communications campaign, and an organization that has produced a newsletter, a poster, and a launch video has done marketing rather than change. It is not a series of town halls at which executives explain the strategy to an audience that is checking email. It is not a survey measuring sentiment. And it is emphatically not a workstream that runs in parallel to the real project and reports on its own activities. Change management, done properly, is the mechanism by which the design of the system is shaped by the people who will use it and by

which their working practices are altered to match it, which means it is not adjacent to the implementation. It is the implementation, viewed from the side that determines whether it works.

The final piece of evidence is negative, and it is the most persuasive of all to anyone who has run one of these programs. Ask a room of technology leaders how many have seen an enterprise implementation fail because the software could not do what was required. A few hands will go up, and the cases will be unusual: a genuine capability gap, a vendor that misrepresented a module, a scale the product had never been tested at. Then ask how many have seen one fail because the organization would not change how it worked, because the business experts were never released, because the training was inadequate, or because the users went back to their spreadsheets. Every hand goes up. The evidence is not merely in the research; it is in the direct experience of nearly everyone who has done this work, and the fact that it has not changed how these programs are budgeted is the most remarkable thing about it.

04 Go-live is not adoption

The most consequential misunderstanding in this entire field is the belief that go-live is the finish line. It is not. It is the moment at which the actual work begins, and the fact that most programs disband their change team, release their consultants, and declare victory at precisely that moment explains a great deal about why so many systems are technically live and operationally irrelevant.

Figure 4 shows what happens next in the two cases. Where change management is sustained past go-live, adoption climbs steadily as people acquire fluency, workarounds are closed off, and the new way becomes the normal way. Where it is not, something more insidious occurs. Adoption spikes in the first weeks, because the old system has been switched off and people have no choice. Then it erodes. Users discover which fields they can leave blank without the system stopping them. They find that they can export to a spreadsheet and do the analysis they used to do. They keep a private tracker because they do not trust the system's numbers. A workaround emerges, then a second, and within a year the organization is running on a shadow process while paying maintenance on a system that is used as a system of record rather than a system of work.

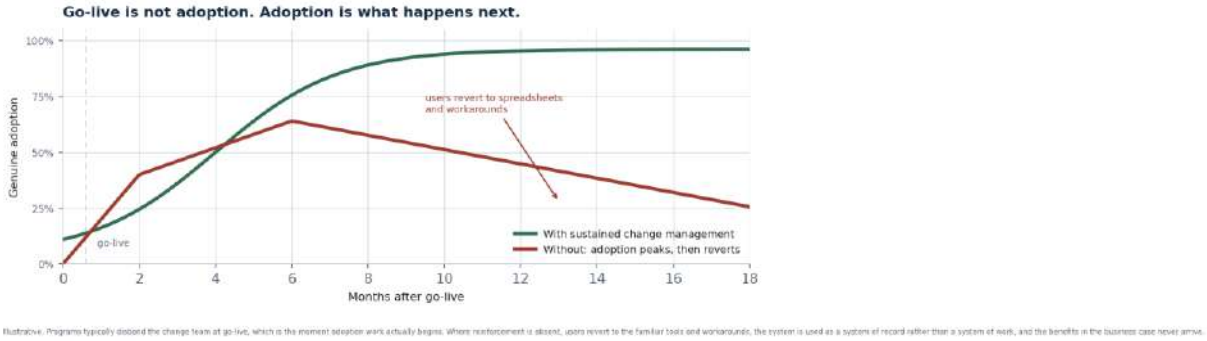

Figure 4. What happens after go-live. The change team leaves at exactly the moment the adoption curve is decided.

The reversion is rarely dramatic and is therefore rarely noticed. Nobody announces that they have gone back to the spreadsheet. The system continues to show green on the dashboard, transactions continue to flow, and the program is recorded as a success in the annual report. What does not happen is the benefit: the inventory reduction, the forecast improvement, the productivity gain, the visibility that justified the investment. Those benefits depended not on the system existing but on people using it in a specific way, and the people are not using it in that way, and nobody is looking closely enough to notice.

The organizational implication is precise and is almost never acted upon. The change management budget should not taper at go-live; it should peak shortly afterward, because that is when reinforcement, coaching, workaround closure, and the measurement of actual usage determine whether the investment produces anything. A program that plans its change spend as a curve rising to go-live and falling to zero has planned to fail, and has done so in a spreadsheet that everybody approved.

The shadow process deserves a closer look, because it is the specific form the failure takes and because most executives have never seen it described. Six months after go-live, walk the floor and ask a planner to show you how they actually do their job. In a program that has failed in the way this article describes,

what you will see is a spreadsheet on a second monitor. The planner exports from the system each morning, does the real work in the spreadsheet, where they have their own logic and their own adjustments accumulated over years, and then enters the result back into the system so that the record is correct. The system has become a place where answers are recorded, not a place where they are produced. Every benefit in the business case assumed the opposite.

What makes this so difficult to detect is that everyone involved is behaving reasonably and nobody is doing anything wrong. The planner is not sabotaging the program; they are getting their job done with the tool that lets them get it done, which is exactly what a conscientious employee should do. Their manager is not undermining anything; they are being measured on output, and output is higher with the spreadsheet. And the program team, if they visit at all, see the transactions in the system and conclude that adoption is proceeding. The failure is invisible from every vantage point except one, which is standing behind a user and watching what they actually do, and that is an activity that almost no program budgets for.

There is a supply chain dimension to this that deserves specific attention, because it makes the problem harder than in most functions. A supply chain change does not stop at the boundary of the organization. When a company implements a new transportation system, its carriers must change how they receive tenders. When it implements a new supplier portal, its suppliers must change how they confirm orders. When it implements a new warehouse system, its three-party logistics providers must change their operating procedures. These are not employees; they cannot be trained, mandated, or held to an adoption target, and they have no particular reason to prioritize somebody else's transformation. Change management in a supply chain therefore extends across a boundary the organization does not control, and any plan that addresses only the internal population has addressed perhaps half of the population whose behavior determines the outcome.

The external dimension has a further sting. A carrier or supplier that is asked to change and finds the new process burdensome does not resist openly; they simply continue to send the message the old way, and the organization, needing the freight moved, accepts it. Within months an exception process has become the standard process for that partner, and then for the partners who hear about it, and the new capability that justified the investment is being used by a minority of the network. The remedy is to plan partner onboarding as a change program with its own resourcing, its own sequence, and its own measurement, rather than as a technical integration task that will be handled by a mapping specialist, and almost no organization does this.

The claim that inadequate change management causes roughly forty percent of failures should be read alongside a second and equally important observation about the other sixty percent. Look at the categories beneath it: data migration problems, under-resourced teams, unrealistic timelines. Each of these is also, on inspection, a human failure rather than a technical one. Data migration fails because nobody was made accountable for the data. Teams are under-resourced because the business would not release its experts. Timelines are unrealistic because somebody committed to a date before understanding the work, and nobody with the standing to object did so. Sort the causes of enterprise

software failure candidly and the technical residue is remarkably small. Almost all of it is organizational, which means almost all of it is within the direct control of the executive who approved the program.

This is the point at which a reader may reasonably ask whether the argument proves too much. If everything is a change management failure, the category has become so broad that it explains nothing. The objection is fair and the answer is that the categories are distinct in remedy even where they are related in cause. The remedy for a data migration failure is an owner and a quality bar for the data. The remedy for an under-resourced team is a funded backfill. The remedy for an unrealistic timeline is a sponsor willing to move a date. These are different actions, taken by different people, and they should not be collapsed. What they share is that none of them is a technology decision, and that all of them are routinely deferred in favor of technology decisions, which are easier and feel more like progress.

05 Why the budget never matches the risk

If change management is the leading cause of failure and has the largest effect on success, the obvious question is why it is funded at a fraction of the software. The answer is not that executives are unaware of the research. Many are. The answer is structural, and it consists of at least four reinforcing pressures, shown in aggregate by Figure 5.

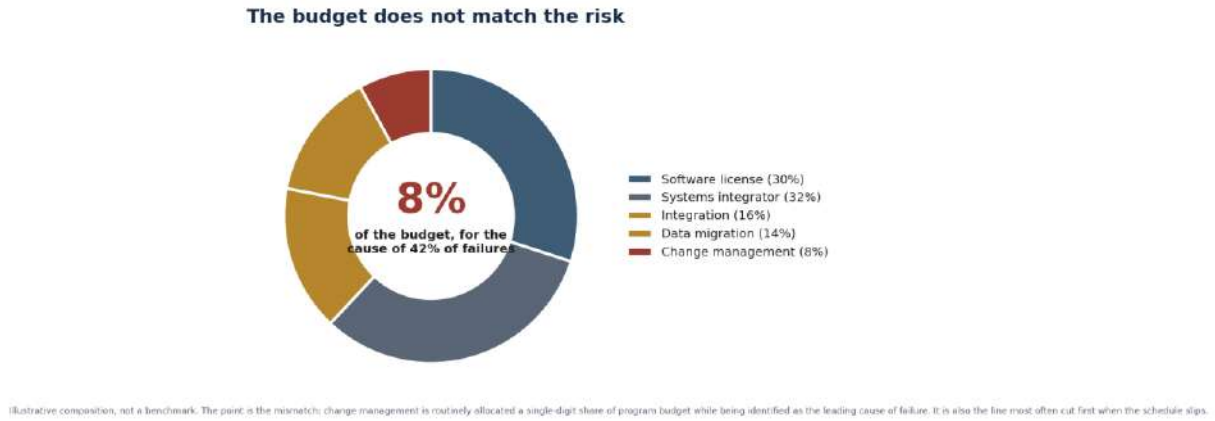

Figure 5. The mismatch. The leading cause of failure receives a single-digit share of the budget.

It cannot be specified, so it cannot be quoted

A software license has a price. An integration has a scope. Change management has neither, in any form a procurement function recognizes, because the deliverable is a behavioral state in a population rather than an artifact. Asked what it will cost to get four thousand people to work differently, an honest answer is a range with a wide confidence interval, and a procurement process that compares vendor quotes cannot process a wide confidence interval. So the line is set at whatever seems defensible, which is usually a small percentage, and the number acquires a spurious authority by being written down.

Its return cannot be attributed

When a program succeeds, the credit goes to the system. Nobody says the enterprise system delivered because the change team closed the workarounds; they say the enterprise system delivered. And when a program fails, the change team is not usually blamed either, because the failure is attributed to the vendor, the integrator, or the data. Change management therefore occupies an unusual position in which it receives neither the credit for success nor the blame for failure, and a discipline that is never held accountable for outcomes is a discipline that is never funded for them.

It asks the business for something it does not want to give

Real change management requires the time of the people who are busiest: the experienced planner, the senior buyer, the warehouse supervisor who knows how everything actually works. Those people are needed for design workshops, for testing, for training their peers, and for the difficult conversations about process. Their functions do not want to release them, because releasing them means the day job suffers. So the program asks for less of their time than it needs, gets less than it asks for, and proceeds

with a design that the people who will use it did not shape, which guarantees the resistance it will later be surprised by.

The people funding it are not the people who will use it

The budget is approved by executives who will never enter a transaction in the system, and it is spent on a system that will be used, all day, every day, by people who were not in the room. This asymmetry is the deep source of the deficit. The pain of a badly adopted system is felt entirely by the people at the bottom, and the budget for preventing it is controlled entirely by people at the top, and there is no mechanism by which the first group's experience reaches the second group's spreadsheet.

There is a fifth pressure, and it is the most human of them. Change management asks executives to spend money on the proposition that their own organization will resist them, which is an unflattering thing to concede and an unwelcome thing to fund. The implicit message of a large change budget is that the people in this company will not do what they are told, that they will need to be persuaded, and that persuading them will take a year and cost real money. Many leaders find that proposition insulting to their organization and to their own authority, and they decline it, on the reasoning that their people are professionals who will adapt. Their people are professionals, and they will not adapt, not because they are unprofessional but because adapting is materially hard and nobody has made it their priority.

The absence of a defensible cost model has a second consequence that compounds the first. Because nobody can say what adequate change management costs, nobody can say that a given allocation is inadequate. A program that has assigned five percent of its budget to the change effort cannot be shown to have underfunded it, because there is no accepted benchmark to compare against, and so the question is never asked in a steering committee. Contrast this with a program that has underfunded its testing, where a test manager can produce a coverage figure and demonstrate the shortfall in a number. The change lead has no equivalent instrument, and so raises the concern as a judgment, which is easily deferred. Building the instrument, an honest estimate of the effort required per affected user for a change of this depth, is the single most useful thing a change function can do for itself.

The reversion described above has a technical accomplice that deserves naming, because removing it is one of the cheapest interventions available. Users revert to spreadsheets because the spreadsheets are still there. The old system is still switched on, in read-only mode, because somebody wanted a safety net. The shared drive still contains the templates. The report that the new system was supposed to replace is still being produced, by a script nobody has turned off, because turning it off felt risky. Every one of these is an open door back to the old way, and human beings under pressure walk through open doors. A program that leaves them open has not merely failed to close the workaround; it has funded the workaround, maintained it, and made it available at the exact moment the user is most tempted.

There is a timing point here that programs consistently get wrong. The moment to close those doors is not immediately at go-live, when users are struggling and the alternative is a genuine safety valve, and not eighteen months later, when the workaround has become the process. It is somewhere between six and twelve weeks after launch, once basic competence exists and before habit has hardened, and identifying that moment requires somebody to be watching. This is one of the many reasons the change

team should not be disbanded at go-live: the most consequential decisions about whether the system will actually be used are taken in the second and third months, by which time, in most programs, there is nobody left who is looking.

06 The axe falls upward

The underfunding is bad. What makes it fatal is what happens when the program falls behind schedule, which it will, because these programs always do. At that point a triage occurs, and the logic of the triage is entirely predictable and entirely wrong. Figure 6 shows the order in which the work is sacrificed.

Figure 6. The order in which work is cut when a program falls behind. The axe falls upward, sparing what is visibly required to launch and consuming what is required to succeed.

The program cannot cut the software configuration, because without it the system does not run. It cannot cut the integration, because without it the system does not connect. It can compress the data migration, and it does, which is how organizations end up launching on data that is accurate two-thirds of the time. It can compress the testing, and it does, which is how defects reach production. And it can cut training and change management outright, because their absence is not visible on the go-live date. The system will still start. The users will still be able to log in. The failure produced by cutting them will not appear for three months, by which time the program will have been declared complete and the team will have moved on.

This is the mechanism, and it is worth dwelling on because it explains the persistence of a failure that everybody knows about. Change management is not underfunded because leaders do not believe in it. It is underfunded because it is the only line in the budget whose removal does not prevent the launch, and every program under pressure eventually faces a choice between the date and the work whose absence will not be noticed until after the date. Given that choice, on a schedule that has been announced to a board, the outcome is not in doubt.

The remedy follows directly from the diagnosis and is uncomfortable enough that few organizations adopt it. The change management budget and the training budget must be ring-fenced, in the sense that they cannot be reallocated to close a schedule gap without an explicit decision at the level that approved the program, recorded as a decision to accept a higher risk of adoption failure. That is a governance mechanism rather than a technology one, it costs nothing, and it converts an invisible default into a visible choice. Most of the value of that mechanism lies in the fact that, once it is a visible choice, executives make it differently.

The ring-fencing mechanism deserves one further specification, because organizations that adopt it in name frequently defeat it in practice. The protection must apply not only to the money but to the time. The most common way a change budget survives on paper while dying in reality is that the money remains allocated while the business experts whose participation it was meant to buy are quietly

withdrawn to deal with a quarter-end, a customer escalation, or a competing project. The budget is intact and the program has lost the only thing the budget was for. A ring-fence that protects the line item and not the calendar protects nothing.

A related and equally corrosive dynamic occurs at the level of the individual manager. A manager whose team is being asked to absorb a new system faces an entirely rational calculation. Participation costs them output now, in a quarter they are being measured on, and delivers benefits later, in a quarter that belongs to somebody else, possibly to their successor. No sensible manager volunteers for that trade unless somebody makes them, which is why exhortation fails and why the sponsorship finding in the next section is as robust as it is. Until the change appears in the manager's objectives with the same weight as their operating targets, they will participate exactly as much as they are compelled to and not one hour more, and they will be right to.

It is worth pausing to note that the same asymmetry appears at the vendor's end of the transaction, and it is not the vendor's fault. A software company is not in a position to make a buyer change its processes, cannot compel a business unit to release its best planner, and has no authority over whether an executive shows up to the design workshops. It sells a capable product and hopes the buyer does the rest. The better vendors say so, and will tell a prospect candidly that the implementation will fail if the organization does not resource the change, and the buyer, hearing this, discounts it as a disclaimer. The vendor is not covering itself. It is telling the buyer the single most important thing it knows, learned from watching hundreds of customers, and the buyer is not listening because the message is inconvenient and the demonstration was compelling.

07 Resistance is rational

The vocabulary of this discipline does it no favors, and no phrase does more damage than managing resistance, which frames the people who will use the system as an obstacle between the program and its objectives. That framing is both insulting and analytically wrong, and organizations that adopt it lose the most valuable information available to them. Figure 7 sets out the actual sources of resistance, and the important observation is the last one.

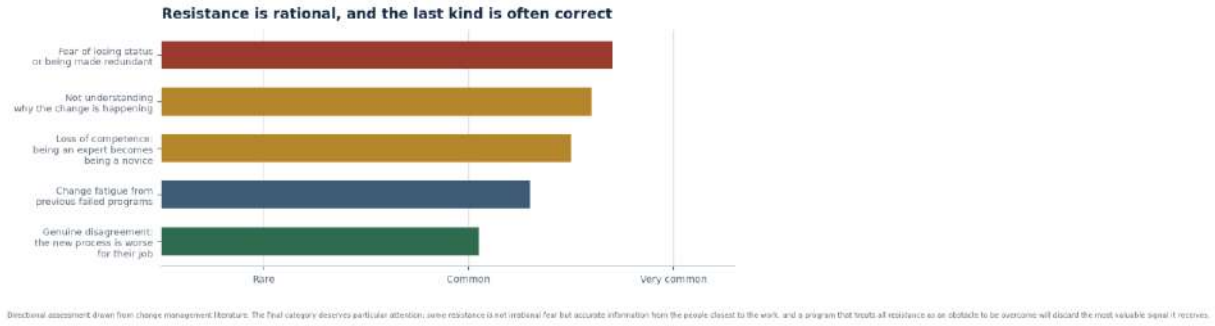

Figure 7. The sources of resistance. Most are rational, and the last category is frequently correct.

Some resistance is fear, and the fear is generally well founded. A person who has spent eleven years becoming the acknowledged expert in the old system is about to become a novice in the new one, in public, in front of colleagues who currently come to them for help. That is a real loss of status and competence, and it is not irrational to resist it. A person who suspects that the automation will eliminate their role is often correct, and telling them otherwise, when it is not true, poisons every subsequent communication the program makes. A person exhausted by two previous programs that were announced with fanfare and quietly abandoned is not being obstructive when they decline to invest in the third; they are applying evidence.

But the category that deserves the most attention is the last, and it is the one that programs are least equipped to hear. Sometimes the people resisting are right. Sometimes the new process really is worse for their part of the job, because it was designed by people who did not understand what that part of the job involves. The planner who says the new system will not let them do the thing they do every Tuesday to handle a recurring exception is not resisting change; they are reporting a design defect, and they are reporting it before it reaches production, which is the cheapest moment at which such a report can arrive. A program that has decided in advance that resistance is an obstacle will process that report as a communications problem, will address it with a message about the benefits of the change, and will discover the defect three months after go-live at a hundred times the cost.

The practical instruction is to treat resistance as a diagnostic instrument rather than a barrier. When a group resists, the first question is not how to overcome them but what they know. Sometimes the answer is that they are afraid, in which case the remedy is honesty about what is going to happen to them. Sometimes the answer is that they do not understand why, in which case the remedy is the explanation nobody gave them. And sometimes the answer is that the design is wrong, in which case they have just saved the program a fortune and should be thanked rather than managed.

One more observation belongs here, and it is the one that most often changes an executive's mind. Ask, of any resistant group, what would have to be true for them to want this change. Sometimes the answer is nothing, in which case the program is asking people to act against their own interest and should at least be honest that this is what it is doing, and should compensate accordingly. But far more often the answer is something small and entirely obtainable: they would want it if it did not double their data entry, if it did not remove the flexibility they use to handle a recurring exception, if somebody had asked them before designing it. Those answers are available for the cost of asking, and almost no program asks, because asking implies a willingness to change the design in response, and the design has already been signed off.

There is one further category of resistance that the models tend to omit and that supply chain leaders will recognize immediately. It is the resistance of the person who has been through this before. In many organizations, the population being asked to adopt a new system contains people who adopted a new system five years ago, invested real effort in learning it, and watched it be abandoned or replaced. They are not resisting the technology. They are declining to spend their own effort on something they expect to be discarded, and their expectation is based on evidence the organization provided. The only remedy for this is a track record, which takes years to build and can be destroyed in a single abandoned program, and the implication is uncomfortable: an organization's capacity for change is a depleting asset, and every failed program depletes it further.

Two further characteristics of resistance are worth understanding because they change how a program should respond. The first is that resistance is frequently silent. The functions that argue loudly in the design workshops are not the dangerous ones; they are engaged, they are telling the program what they think, and their objections can be addressed. The dangerous function is the one that agrees in the room, signs the document, and then quietly continues doing what it was doing. Silent compliance is far more damaging than open objection, and a program that measures resistance by the volume of complaints will conclude that its quietest business unit is its most supportive, which is frequently the opposite of the truth.

The second is that resistance concentrates in the middle. Executives support the change because they approved it. Front-line staff will generally do what they are asked, if they understand it and are given the tools. The layer that determines the outcome is the one in between: the supervisors and middle managers whose targets will suffer during the transition, whose expertise is invested in the current way, and who will be blamed if throughput drops. That layer has both the strongest incentive to resist and the greatest practical ability to do so, because they control what their teams actually spend their time on. A change program that has engaged the executives and the front line and skipped the middle has skipped the only layer that could have stopped it.

08 The sequence of individual change

Underneath the organizational abstraction, change happens one person at a time, and it happens in a sequence that cannot be reordered. The best-known model in the field sets out five stages, shown in Figure 8, and the reason to reproduce it here is not to endorse a particular consultancy's framework but because the sequence itself is obviously true once stated, and because almost every program violates it in the same way.

Figure 8. The sequence of individual change. Most programs begin at the third rung, which is why the training does not take.

- **Awareness.** Does the person know why the change is happening, in terms that make sense from where they sit? Not the corporate rationale, which they will discount, but the reason that connects to their own experience of the current system's failures.
- **Desire.** Do they want to participate? This is the rung that programs find distasteful, because it requires answering the question of what is in it for the individual, and the honest answer is sometimes nothing, or worse than nothing. It must nonetheless be answered, because a person who does not want the change will not adopt it however well they are trained.
- **Knowledge.** Do they know how to work the new way? This is training, and it is where almost every program begins, four weeks before go-live, in a classroom, on a system that is not yet configured.
- **Ability.** Can they actually do it, at speed, under pressure, on a Monday morning with a queue of orders? Knowledge is not ability, and the gap between them is where most of the post-go-live productivity collapse actually lives.
- **Reinforcement.** Is the new way rewarded, is the old way closed off, and does anybody check? Without this rung, the adoption curve in Figure 4 bends downward, because human beings revert to the path of least resistance and the path of least resistance is the spreadsheet they still have.

The diagnostic power of the sequence is that it explains the specific failure that puzzles most program leaders. They trained everyone. The training was well reviewed. And nobody is using the system properly. The reason is that they started at the third rung. The people they trained did not understand why the change was happening, did not want it, and were therefore trained in a skill they had no intention of using, which is why the training evaluations were positive and the adoption was not. Training is the third rung, not the first, and a program that skips the first two has purchased a well-received course and no behavior change at all.

09 Sponsorship: the single greatest predictor

If an executive reading this article takes away one finding, it should be this one, because it is the most robust in the literature and the most frequently violated in practice. Across many years of benchmarking, the factor most consistently identified as the greatest contributor to the success of a change program is active and visible sponsorship by a senior executive, and its absence is identified as the greatest contributor to failure. Not methodology. Not tooling. Not budget, in isolation. Sponsorship. Figure 9 shows the ordering.

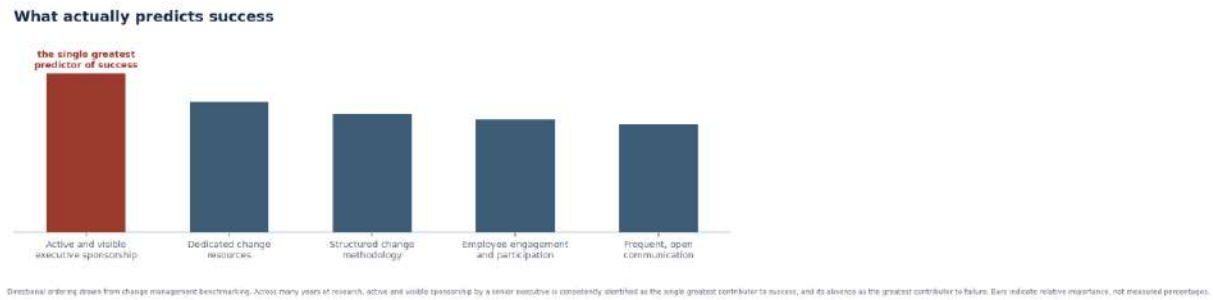

Figure 9. What actually predicts success. The variable at the top cannot be delegated, and it is the one most commonly delegated.

The word doing the work in that finding is visible, and it is worth being precise about what it excludes. A sponsor who approved the budget is not a visible sponsor. A sponsor who appears in a recorded video at kickoff and is not seen again is not a visible sponsor. A sponsor who has delegated the program to a director and asks for a status report each month is not a visible sponsor. Visible sponsorship means that the people affected by the change repeatedly see and hear the senior executive personally advocating for it, personally explaining why it matters, personally answering hard questions in rooms where the questions are hostile, and personally intervening when a function refuses to release its people or declines to change its process.

The reason this matters so much is that a change program is, at bottom, a contest of priorities. Every person asked to adopt a new system has a day job, and the day job has a manager, and the manager has targets, and none of those targets is adoption. When the program asks for time and the day job asks for output, the day job wins, unless somebody with more authority than the manager has made it unambiguous that the change is not optional. Only a sponsor can do that. A program manager cannot, a change consultant certainly cannot, and a communications campaign cannot. The organization is reading, at every moment, whether the leadership actually means it, and it reads this not from what is said but from what happens when somebody declines to comply.

Two failure modes are worth naming. The first is the absent sponsor, who supports the program in principle, has other priorities, and appears only when it is in trouble, by which time their appearance is a rebuke rather than an endorsement. The second, more common and more corrosive, is the sponsor who exempts themselves and their own function. Everybody notices. The moment one senior leader is permitted to keep the old process, or to send a deputy to the design workshops, or to miss the training, the change becomes optional for everyone, and every subsequent communication about its importance is heard as noise. A sponsor who will not hold their peers to the change has not sponsored it; they have endorsed it, which is a different and far less useful thing.

What adequate resourcing looks like

The table below sets out the components of a properly resourced change effort against what most programs actually fund. The gap between the two columns is, in most organizations, the difference between a system that goes live and a system that works.

Component	What is typically funded	What is actually required
Sponsorship	A name on the charter	Visible presence, and peers held to the change
Change staffing	A part-time role	Dedicated practitioners, scaled to the population
Business expert time	Whatever is left over	A funded share of their time, with backfill paid for
Change agents	Volunteers, named late	Respected peers, engaged early, with a design role
Training	Generic, weeks before launch	Role-specific, on real data, with support at the elbow
Post-go-live	Nothing; the team disbands	The peak of the spend: coaching and workaround closure
Measurement	Training completion rates	System logs, workaround counts, benefit realization

10 The customization trap

There is a specific decision, taken early in almost every implementation, in which the change management deficit converts itself into technical debt and financial loss, and it deserves its own section because it is the point at which the whole failure becomes irreversible. It is the decision about whether to change the process to fit the system, or the system to fit the process.

The decision presents itself as a technical question and is in fact an organizational one. A business unit says that the standard process in the software does not match how they work. The program has two options. It can require the business unit to adopt the standard process, which means confronting the resistance, doing the change work, and absorbing the political cost of telling a function that its way of working will end. Or it can customize the software, which costs money and time but avoids the confrontation entirely. The first option is what change management actually consists of. The second option is what happens when change management has not been funded, because a program without the mandate, the sponsor, or the resources to change behavior has only one lever left, and it is the configuration screen.

The consequences of choosing the second option compound in a way that is invisible at the moment of the decision. Each customization adds cost, adds testing burden, and adds a piece of code that must be maintained and re-validated at every upgrade for as long as the system lives. Each one preserves a process that the organization had an opportunity to improve and chose not to. And each one is a precedent, because once one function has been permitted to keep its process, the next function's argument for the same treatment is unanswerable. The program that begins by customizing to avoid a difficult conversation ends by customizing everything, at which point it has spent an enormous sum to automate exactly the operating model it already had.

The most expensive documented instance of this trap is instructive precisely because the technology was never the problem. A large European grocery retailer spent something on the order of seven years and roughly half a billion euros attempting to replace its inventory system, and abandoned the effort. The cause most frequently cited is that the company insisted on retaining a legacy convention for valuing inventory rather than adopting the standard the new system used. That single refusal, which was a refusal to change a process, cascaded into customization that touched every module and every interface until the complexity became unmanageable. The software was capable of running the business. The organization was not willing to be run differently, and no amount of software could resolve that.

The disciplined position, and the one that a properly sponsored program can hold, is that customization is justified only where the process constitutes a real competitive differentiator that the organization would defend in a board meeting. That test is stricter than it sounds and it eliminates the great majority of requests, because most processes that feel distinctive to the people who perform them are merely familiar. A program that applies the test candidly will find that it can adopt the standard process in most places, and will find that adopting it is a change management problem rather than a software problem, which is exactly the point.

It is worth being explicit about what a sponsor should actually do, because the abstraction is easy to agree with and hard to act on. A sponsor should open the design workshops in person, not by video, and should stay long enough to hear the objections rather than long enough to deliver the message. They should be present at the difficult sessions, the ones where a function is being told that its process will end, because their presence is what makes the decision stick. They should personally communicate the change repeatedly, in their own words, and should be able to explain why it is happening without reading from a deck. They should intervene, visibly, the first time a function fails to release its people, because the organization is watching to see whether anything happens, and if nothing happens the program is over regardless of what the plan says. And they should be the person who says no to the customization request, because nobody below them can.

The sequence also explains why so many programs experience a productivity collapse in the weeks after go-live that nobody forecast. Knowledge is not ability, and the distance between them is the difference between having been shown how to do something and being able to do it at pace, correctly, on a busy morning, while a queue builds. Every organization implementing a system should expect and plan for a real drop in throughput in the first weeks, and should staff for it, because the alternative is that the drop occurs anyway, unplanned, and the business responds by improvising workarounds under pressure, which then become permanent. The productivity dip is not a sign of failure. It is arithmetic. Refusing to plan for it is what turns it into failure.

It follows that the choice of sponsor matters more than almost any other appointment in the program, and it is routinely made on the wrong criteria. The instinct is to appoint the executive whose function is most affected, or the one with the most available time, or the one who volunteered. The right criterion is different: appoint the executive who can compel their peers. Sponsorship works because it settles the contest of priorities described above, and it can only settle that contest if the sponsor outranks or can credibly overrule the leaders whose functions will resist. An enthusiastic sponsor who cannot compel a peer is not a sponsor; they are an advocate, and advocacy is not what the research is measuring when it identifies sponsorship as the greatest predictor of success.

11 What the famous failures actually teach

The canonical enterprise software disasters are usually recounted as technology stories. They are not. Read carefully, each is a story about an organization that could not or would not change, and the technology is a bystander.

Compressed schedules and the testing that was not done

A confectionery manufacturer's notorious failure in the late 1990s is remembered as a systems failure and was in substance a schedule failure. The company compressed an implementation timeline substantially, chose to go live during its peak season, and cut the testing and preparation that the compressed schedule no longer accommodated. Orders could not be fulfilled during the most important weeks of its commercial year. The technology was capable. The organization attempted to absorb a fundamental operational change in less time than the change required, and the compression consumed precisely the activities, testing and readiness, whose absence would not be felt until it was too late. This is the axe falling upward, in its most expensive documented form.

The data that nobody owned

A major retailer's failed expansion into a neighboring country is frequently described as a supply chain failure and is more accurately described as a failure to treat data as work requiring people. Product records were created at enormous speed under schedule pressure, and reporting subsequently put their accuracy at a fraction of what the company achieved in its home market. Shelves were empty, distribution could not function, and the company withdrew from the market after losses in the billions. The system did what it was told. Nobody had been made accountable for what it was told, because data entry was regarded as a task rather than as a workstream with an owner and a quality bar, and the schedule pressure that produced the bad data was permitted to override the only control that would have caught it.

The integrator who was blamed and the decisions that were not made

A large brewer sued its systems integrator over a troubled implementation, seeking damages in excess of a hundred million dollars, and the integrator counterclaimed. The specifics were settled and are less instructive than the pattern, which recurs across the litigated cases: a client that expected an implementer to make organizational decisions it had not made itself, an implementer that proceeded on assumptions the client had never validated, and a program in which nobody with the authority to change how the business worked was actually present in the room where the design was agreed. Litigation of this kind is almost always the terminal phase of an accountability vacuum that existed from the first month.

The common thread is that in none of these cases did the software fail to do what software does. In each, an organization declined to do the human work: to allow enough time, to make somebody accountable for the data, to change a process it preferred not to change, or to send its actual decision-makers into the room. The technology was then blamed, the vendor was sometimes sued, and the

lesson, which was available in every case and is available now, was not learned. It is worth stating that lesson without decoration. These programs do not fail because the software was wrong. They fail because the organization did not change, and nobody was funded, empowered, or held accountable to make it.

The customization decision has a second-order effect that is worth drawing out, because it explains why the trap is so hard to escape once entered. Every customization the organization accepts becomes a reason not to upgrade later, since each upgrade requires re-validating the custom code against a new release. Over a few years the accumulated customization makes upgrading so expensive and so risky that the organization stops doing it, at which point it is running an increasingly obsolete version of a system it paid a fortune to install, and it will eventually face a second replacement program, at which point the same conversation about process change will occur again, with the same absence of a sponsor willing to have it. This is how organizations end up replacing enterprise systems every decade and never becoming better at anything.

The honest counterargument deserves airing, because there is a version of this that goes too far. Not every standard process in a software package is superior to what an organization currently does, and a program that adopts every default on the grounds that the vendor knows best will find itself degrading processes that were, in fact, good. Some organizations have truly superior ways of working, developed over years, that constitute real advantage, and they should defend them. The discipline is not to refuse all customization; it is to require that each request survives the question of whether the process is a differentiator or merely a habit, and to have somebody senior enough to answer that question plainly rather than to accept whichever answer the affected function prefers.

There is a diagnostic question that reveals whether an organization has real sponsorship or the appearance of it, and it can be asked in a single steering committee. Ask what has happened, so far, to the function that has not complied. Every program has one: a business unit that has not released its people, has not attended the design sessions, or has quietly continued planning to keep its own process. If the answer is that the issue has been escalated and is being tracked, there is no sponsorship, and the program will fail in the way this article describes. If the answer is that the sponsor went to see them and the position changed, the program will probably succeed. Nothing else in the governance apparatus tells a steering committee as much as that one question, and it is almost never asked, because everybody present knows the answer and would rather not say it.

The customization trap also explains a puzzle that many executives will have encountered and never resolved. Why is it that two companies in the same industry, implementing the same software, with the same integrator, on similar budgets, produce completely different outcomes, one transforming its operations and the other spending seven figures to automate its existing mess? The software was identical. The difference is that one of them had somebody willing to say that the process would change, and the other did not, and every subsequent divergence in the two programs flows from that single decision, taken in the first quarter, in a room where nobody thought they were making the most consequential choice of the entire program.

12 Training is not change management

A great many organizations believe they have done change management because they have done training, and the conflation is worth breaking apart, because it is the most common way that a program can sincerely believe it has addressed the risk while having done almost nothing about it.

Training addresses the third rung of the sequence in Figure 8, knowledge. It does nothing about the first two, awareness and desire, and it is largely ineffective at the fourth, ability, because ability is acquired by doing the job under real conditions rather than by attending a class. And it does nothing whatsoever about the fifth, reinforcement, which is the rung that determines whether any of it lasts. A training program is therefore a necessary component of change management and constitutes perhaps a fifth of it, and an organization that has trained its people and done nothing else has bought the least important fifth.

The typical training program compounds the problem with its design. It is delivered four weeks before go-live, which is far enough in advance that most of it is forgotten by the time it is needed. It is generic rather than role-specific, so that a warehouse supervisor sits through modules about procurement. It is delivered on a system that is not yet fully configured, using demonstration data that bears no resemblance to the records the person will actually encounter. And it is evaluated by asking attendees whether they found it useful, which measures the quality of the room and the coffee rather than whether anyone can now do their job. A program that has done this and stopped has generated a completion statistic and no capability.

What works, and it is not more expensive, is different in four respects. It is role-specific, so that a person is trained on the transactions they will actually perform. It uses the organization's own data, including its difficult data, so that people encounter the real records rather than the demonstration ones. It happens close enough to go-live to be retained and is followed by support at the elbow in the first weeks, when the questions are real and the queue is real. And it is evaluated by whether people can perform the task, not by whether they enjoyed learning about it. None of this requires a larger training budget. It requires the training to be designed by somebody who has thought about what the person will actually be doing on the first Monday.

A fourth case is worth adding because it is the one most readers will recognize from their own experience, even though it never makes the lists. It is the program that succeeded technically and was quietly abandoned. The system went live, it worked, it was announced, and within two years the organization had drifted back to the old way, kept the software running for compliance reasons, and stopped talking about the transformation. Nobody sued anybody. No amount was written off. The failure appears in no case study because it produced no event, and it is by a wide margin the most common outcome of an underfunded change effort. The money was spent, the software runs, and the organization operates exactly as it did before, which is the most expensive form of nothing available in enterprise technology.

The customization trap has a tell that a steering committee can watch for, and it costs nothing to monitor. Count the customization requests, and note where they come from. A steady stream of

requests from a single function, each individually reasonable, is not a technical signal; it is the sound of a function that has not accepted the change and is routing its objection through the configuration backlog because the front door was closed. The correct response is not to evaluate the requests on their technical merits, one at a time, which is what the program will do if left alone. It is for the sponsor to go and have the conversation that the requests are substituting for.

A fourth lesson runs through all of these cases and deserves to be extracted, because it is the one that a reader can act on immediately. In every instance, somebody inside the organization knew. Somebody knew the data was not ready. Somebody knew the schedule could not accommodate the testing. Somebody knew that the business unit had never accepted the new process and was planning to work around it. These things were known, by named people, in advance, and they were not acted upon, because the program had a date, the date had been announced, and the incentive structure at every level rewarded reporting progress rather than reporting risk. The question a steering committee should ask, quarterly, is not whether the program is on track. It is what somebody in this organization knows that is not in this report, and who would be punished for saying it.

13 How to resource it properly

For an organization that has accepted the argument, the question becomes what adequate resourcing actually looks like. The specifics below are drawn from what distinguishes the programs that succeed, and none of them requires a large budget relative to the software they protect.

1. **Ring-fence the budget.** The change and training lines cannot be reallocated to close a schedule gap without an explicit decision, recorded, at the level that approved the program. This costs nothing and prevents the single most common failure mode described in this article.
2. **Name a sponsor, and define what sponsorship requires of them.** Not an endorsement but a commitment: a specified cadence of visible advocacy, personal presence at the difficult sessions, and the willingness to hold peers to the change when a function declines to comply. A sponsor who will not do these things should not be the sponsor.
3. **Fund dedicated change resources, not a part-time role.** A common planning ratio is one dedicated change practitioner per fifty to one hundred affected users for a significant transformation, adjusted for the depth of the change. Whatever the ratio, the requirement is that somebody's whole job is this, because a part-time change role is a full-time day job with a change title attached.
4. **Buy the time of the business experts, explicitly.** The best planner, the senior buyer, the supervisor who knows how the warehouse really works: the program needs a substantial share of their time, and their function will not give it up unless the program pays for backfill. Budget for the backfill. A program designed without the people who do the work will be resisted by the people who do the work, and they will be right.
5. **Build a network of change agents inside the functions.** Peers, respected, embedded, trained early, and given a real role in the design. Adoption spreads through the informal network far more effectively than through any communication from the program, and the person a planner actually asks for help is the planner at the next desk, not the help desk.
6. **Plan the change spend to peak after go-live, not before.** This inverts the standard profile and it follows directly from Figure 4. The reinforcement, coaching, workaround closure, and usage measurement that determine whether the benefit arrives all happen in the months after launch, and they are unfunded in almost every program.
7. **Close the workarounds deliberately.** Reversion is not a failure of willpower; it is a rational response to an available alternative. Turn off the old system. Remove the shared drive. Stop accepting the spreadsheet. Where a workaround persists because the new system truly cannot do something, that is a defect to be fixed, not a behavior to be exhorted away.

The total additional cost of doing all of this is real and is a small fraction of the program. The cost of not doing it is the program, which is a trade that would be obvious if the failure arrived on the go-live date rather than a year later, in a form that nobody attributes to the decision that caused it.

The support-at-the-elbow point deserves emphasis because it is cheap, it works, and almost nobody funds it. In the first two to four weeks after go-live, the difference between a user who becomes competent and a user who reverts to the spreadsheet is frequently a single question answered at the moment they are stuck. Not a help desk ticket, resolved in two days. A person, standing nearby, who can

answer it in thirty seconds. Programs that staff this period generously report dramatically better adoption, and the cost is a handful of people for a month, which is a rounding error against the software. Programs that do not staff it discover that a user who was stuck twice and could not get help has permanently concluded that the system does not work, and no amount of later training will undo that conclusion.

The resourcing recommendations above are worth translating into a number, because a recommendation that cannot be costed will not be funded. On a program where the software and implementation together cost ten million dollars, the change effort described here, dedicated practitioners, funded backfill for the business experts, a proper training design, a network of change agents, and a post-go-live support period, will typically cost somewhere in the region of one to two million dollars, which is ten to twenty percent of the program rather than the eight percent that is typically allocated and the three percent that survives the first schedule slip. That is a real increase and it is not a large one, and it is being spent to protect the other eight to nine million from the failure mode that most reliably destroys it. Framed that way, the decision is not close.

The training design point generalizes into a principle that applies across the whole discipline. The measure of any change activity is not whether it was delivered but whether behavior changed, and almost every metric that programs actually report measures the former. Communications sent is not awareness. Training completed is not knowledge. Attendance is not desire. Sign-off is not agreement. Every one of these is a measure of the change team's output, and a program can achieve a perfect score on all of them while the organization continues to work exactly as it did before. A steering committee that accepts activity metrics is not overseeing a change program; it is being reassured by one, and the reassurance will hold right up until the benefits review.

There is one more thing a properly resourced program does that costs nothing and is almost universally skipped: it tells people the truth about what is going to happen to them. If the new system will eliminate roles, say so, early, and say what will happen to the people in them. If it will make a job less autonomous and more procedural, say that too. The instinct is to withhold, on the reasoning that the news will demotivate people during a critical period, and the instinct is exactly wrong, because the people affected already suspect and are now watching the program lie to them. Every subsequent communication is discounted accordingly, and the credibility that the change effort depends on is spent before the program has begun. Candour is not merely the decent option here. It is the operationally superior one, and the programs that manage it find that the resistance they feared largely does not materialize.

The other thing that costs nothing is to make the design visible. Publish what the new process will be, in plain terms, function by function, before it is locked, and invite the people who will perform it to say what is wrong with it. The objections will be numerous, most will be answerable, and a small number will identify defects that would otherwise have surfaced in production at enormous cost. The program will resist this on the grounds that it will slow the design phase, which it will, by weeks. The alternative is to discover the same defects after go-live, which costs months. This trade is made badly in almost every

program, and it is made badly for a reason that is worth naming: the design phase has a date, and the post-go-live period does not.

14 Measurement, governance, and a scoring rubric

None of this survives a budget cycle unless it is measured, and change management is notoriously badly measured, in ways that actively conceal failure. The typical program reports training completion rates, communications delivered, and attendance at town halls, all of which measure the activity of the change team rather than the state of the organization, and all of which can be at one hundred percent in a program that is about to fail.

Measure the organization, not the change team

- Actual system usage by role, measured from the system's own logs, not from self-report. Which transactions are being performed, by whom, and which are being avoided.
- Workaround prevalence. How many spreadsheets are still in use, how many exports are being run, how many private trackers exist. This is uncomfortable to measure and is the single most informative number available.
- Process compliance. Are people following the designed process, or have they found a faster path that bypasses the control the process existed to enforce.
- Time to competence. How long after go-live does a user return to their pre-implementation productivity. If the answer is never, the program has permanently reduced the organization's capability.
- The benefits themselves. Inventory, forecast accuracy, cycle time, cost to serve. These were the reason for the program, and it is remarkable how rarely anyone returns to check whether they moved.

A scoring rubric

The dimensions below allow a steering committee to assess a program's change readiness candidly, before go-live, while there is still time to act on the answer.

Dimension	What good looks like	Red flag
Sponsorship	A named executive, visibly present, holding peers to it	Sponsorship delegated to the program manager
Dedicated resources	Full-time change practitioners, funded	A part-time role added to somebody's day job
Starts at awareness	Why, and what is in it for me, answered first	The plan begins with a training schedule
Budget ring-fenced	Cannot be raided without an explicit recorded decision	The change line is the contingency fund
Post-go-live plan	Change spend peaks after launch; coaching is funded	The change team is released at go-live
Resistance as signal	Objections are investigated; some change the design	Resistance is a communications problem to manage

Dimension	What good looks like	Red flag
Adoption measured	System logs, workaround counts, benefit realization	Training completion and attendance statistics

A program scoring well on the left-hand column is one whose software will probably deliver what was promised. A program tripping the red flags on the right will go live, will be declared complete, and will not deliver, and the failure will be attributed to a vendor who did nothing wrong.

It is also worth stating what these measures are for, because a steering committee that measures adoption without knowing what it will do about a bad number has merely added a slide. The purpose of measuring workaround prevalence is to close the workarounds. The purpose of measuring usage by role is to find the function that is not using the system and to send the sponsor there. The purpose of measuring time to competence is to know when to withdraw the support, and to keep it in place longer where the number says it is needed. A measurement without a pre-agreed response is decoration, and the decision about what will be done when adoption is poor should be taken before go-live, when it can be discussed calmly, rather than after, when it will be a crisis.

A brief note on external help, since most organizations will buy some. Consultancies can supply methodology, capacity, and experience, and the good ones supply all three. What they cannot supply is authority, and this is the reason so many expensive change engagements produce documentation and no change. A consultant cannot tell a business unit head that their process is ending. A consultant cannot compel a function to release its best planner for three months. A consultant cannot make the change matter more than the quarter. Those things require an executive with standing, and an organization that hires a consultancy in the hope of avoiding the need for one has bought an alibi rather than a capability. Use external help to build the machine; do not use it to avoid deciding.

One further governance mechanism is worth adopting, and it is the cheapest of all. Require, as a condition of program closure, that somebody return in twelve months and report on whether the benefits in the business case actually arrived, measured the same way they were promised. Not a status report, not an adoption survey, but the specific numbers that justified the investment. The mere existence of that requirement changes behavior throughout the program, because it makes somebody accountable for an outcome rather than an event, and it is remarkable how few organizations impose it. Most programs are closed at go-live, the team is dispersed, and the question of whether the money produced anything is never formally asked by anyone, which is the deepest reason this failure repeats.

A final resourcing point concerns the timing of the spend, because the profile matters as much as the total. A change budget that is loaded before go-live buys communications, training, and readiness assessments, all of which are useful and none of which determine the outcome. A change budget that is loaded after go-live buys coaching, workaround closure, defect resolution, and the measurement that tells the organization whether the benefit is arriving, all of which do determine the outcome. Most programs spend eighty percent of a small budget before launch and nothing afterward. The organizations that succeed spend perhaps half before and half after, and the second half is the half that

works, because it is the only part of the spend that operates on the behavior of people who are actually using the system rather than on the expectations of people who are anticipating it.

15 Conclusion: fund the people

The argument of this article is unusual in one respect, which is that it requires no new information. Every finding cited here has been available for two decades. The research is consistent, the failure cases are public, the mechanism is understood, and the remedy is neither expensive nor technically difficult. What is missing is not knowledge. What is missing is the willingness to spend a meaningful share of a technology budget on something that is not technology, and to hold that position when the schedule slips and the easiest available saving is sitting in the change line.

It is worth restating the core asymmetry one final time, because it is the whole of the case. The software will work. It works for thousands of other organizations, it has been tested more thoroughly than anything the buyer will ever build, and it is not where the risk lives. The risk lives in whether four thousand people will change how they do their jobs, and that outcome is determined by a small number of things: whether a senior leader was visibly and repeatedly present, whether the people who do the work were in the room when the process was designed, whether the resistance was investigated rather than managed, whether the training was about the actual job, and whether anybody was still funded to help three months after the launch. None of these is a technology decision. All of them are within the direct control of the executive approving the budget.

The instruction, then, is simple enough to act on this quarter. Find the change management line in the next technology business case that comes to you. If it is a single-digit percentage, ask why the largest identified cause of failure is receiving the smallest allocation. If the plan begins with training, ask what has been done about awareness and desire. If the change team is scheduled to disband at go-live, ask who will be closing the workarounds in month four. And if the answer to any of these is unsatisfactory, understand clearly what you are approving: a system that will go live, a program that will be declared a success, and a benefit that will never arrive, at a cost that will appear in no report and will be attributed to no decision. That outcome is the norm. It is entirely avoidable, and avoiding it costs less than the software.

It is worth closing with the observation that this is, in one respect, an optimistic article. The failure it describes is not caused by a hard technical problem, an immature market, or an absence of knowledge. Everything required to avoid it already exists and is already understood. The research has been available for twenty years, the mechanisms are simple, the remedies are cheap relative to the programs they protect, and none of them requires a capability that an ordinary organization does not have. What they require is that an executive decide that the human half of the program is real, fund it accordingly, and hold that funding when the schedule slips and the easiest saving is sitting right there. That is a decision, not a discovery, and it is available to anyone reading this on the day they read it.

A closing word on how to read the failure statistics with which this article opened. It would be a mistake to conclude from them that enterprise software is a bad investment or that these programs cannot succeed, because a substantial minority of them do succeed, and the ones that succeed produce exactly the returns that the business cases promise. The difference between the two populations is not the software, the vendor, the integrator, or the budget. It is whether the organization did the human work.

That is an unusually actionable finding, because it means that the variable which determines the outcome is the one entirely within the buyer's control, and it is the one they are least likely to have funded.

16 Methodology, caveats, and sources

Methodology

- This article synthesizes analyst research, change management benchmarking, industry failure analyses, litigation and press records, and practitioner literature, current to mid-2026. Supply Chain Research is independent and accepts no payment from the vendors, consultancies, or platforms discussed.
- Where a figure originates with a party that has a commercial interest in the finding, this is stated explicitly in the text and in the figure notes.

Caveats

- The finding that programs with excellent change management are approximately six times more likely to meet objectives comes from benchmarking research published by a firm that trains and certifies change management practitioners. The source is an interested party and the precise multiple should be treated as directional. The direction is corroborated by independent analyses of failure causes.
- The attribution of roughly forty percent of failures to inadequate change management is drawn from industry analyses of enterprise implementations. The categories overlap, the shares are approximate, and different studies apportion them differently. The ordering, in which organizational causes dominate technical ones, is consistent across sources.
- Budget composition figures shown in Figure 5 are illustrative and are not a published benchmark. They are used to convey the mismatch between allocation and risk rather than to state a measured average.
- The staffing ratio cited in section thirteen is a common planning heuristic rather than a validated finding, and appropriate resourcing depends heavily on the depth of the change, the size of the affected population, and the organization's history with previous programs.
- The failure cases are drawn from press accounts, court filings, and subsequent case analyses. Root-cause attributions in such accounts are necessarily reconstructions and the parties involved have in some instances disputed them.

Sources

[1] Prosci. Change management benchmarking research, including the ADKAR model and findings on sponsorship. Interested source.

[2] Gartner. Gartner predicts 60 percent of supply chain digital adoption efforts will fail to deliver promised value by 2028.

[3] Bloch, Blumberg and Laartz, McKinsey and University of Oxford (2012). Delivering large-scale IT projects on time, on budget, and on value.

[4] CIO (Foundry). Famous ERP disasters, dustups, and disappointments.

[5] Panorama Consulting Group. The ERP Report: implementation outcomes, overruns, and root causes.

[6] Prosci. Why do ERP implementations fail: research on the people side of change.

[7] Henrico Dolfing. Project failure case studies, including Target Canada and Lidl.

[8] Gartner. Newsroom and research on ERP return on investment, user adoption, and post-implementation optimization.

Additional context drawn from industry reporting on enterprise implementation outcomes, from academic and practitioner literature on organizational change, and from published accounts of the failure cases discussed. Figures originating with interested parties are identified as such and are directional. This article is analysis, not legal, procurement, or investment advice.