The Shared-Risk Contract: A Buyer's Guide

What well-meaning buyers of custom software need to know

March 2026

Contents

The Shared-Risk Contract: A Buyer's Guide

What well-meaning buyers of custom software need to know: why fixed-price contracts don't protect you, what shared risk actually means in practice, and what a good engagement feels like from both sides.

Opening

There is a moment in most failing fixed-price custom software projects when the buyer reads the contract.

Not the whole contract — they read it before signing, or someone did. They read one clause. The one about change orders, or scope, or the definition of acceptance criteria. They read it because a change order has arrived, or because the vendor is arguing that a feature is out of scope, and the buyer is trying to find their ground.

They read it, and for the first time they understand what they actually signed. Not what they understood they were signing — what they signed.

The spec was a document they wrote as their best understanding of what they needed. They understood it as a starting point, a description of intent. The vendor understood it as a boundary. Both parties signed the same document with different mental models of what it meant. The contract has been running on those two incompatible mental models ever since, and the moment of reading is the moment they become visible.

This essay is for buyers before that moment arrives.

I. Why Custom Software Projects Fail

The most reliable finding in software project research is that most custom software projects fail to deliver what they promised, on time, within budget. The Standish Group has tracked this since 1994. The percentages shift by year and methodology, but the pattern holds: roughly one in three projects is outright cancelled or delivered in a form the organization doesn't use; more than half are over budget, over time, or missing significant functionality. Only a small fraction deliver everything promised on schedule.

Buyers often attribute this to vendor incompetence. The research says otherwise.

The primary causes of software project failure are: incomplete or changing requirements, lack of user involve-

ment, lack of executive support, unrealistic expectations, and scope creep. These are buyer-side failures as much as vendor-side ones. The buyer who hands a vendor a completed spec and expects delivery has already made the foundational error: they have treated uncertain requirements as settled, and the vendor has priced a project accordingly.

Custom software is not like construction in the ways buyers typically assume. A building has physics: the structural loads are calculable, the materials have known properties, the methods have centuries of documented practice. A software project starts with a problem statement, not a blueprint. The requirements are the buyer's current best understanding of what they need — and that understanding is always incomplete, always subject to revision once users interact with actual software.

The construction metaphor breaks down precisely where it would be most useful. A building's blueprints can be complete before ground is broken. A software spec cannot, because the act of building reveals information that changes what should be built.

The Real Risk Distribution

In a typical fixed-price software engagement:

- The buyer assumes they are transferring risk to the vendor by locking a price.
- The vendor assumes they are managing risk by scoping tightly and preserving change order rights.
- Both assumptions are partially correct and together produce a worse outcome than either party wanted.

The vendor's fixed price is based on an estimate made at the moment of maximum uncertainty — before any code has been written, before any technical assumptions have been tested, before the buyer's actual users have interacted with anything. The vendor prices this uncertainty into the quote or absorbs it later through change orders or quality shortcuts. The buyer pays either way; they just don't know it at signing.

The buyer's protective instincts — fixed price, complete spec, contractual recourse — each produce the problem they're designed to prevent. The fixed price converts requirements uncertainty into scope dispute. The complete spec locks in requirements at the moment of maximum uncertainty. The contractual protection creates incentives for the vendor to document against the buyer rather than collaborate with them.

II. The Contract Trap

How Fixed-Price Contracts Work in Practice

A fixed-price contract is a bet. The vendor is betting that the project will cost less than the quoted price. The buyer is betting that the quoted price reflects the actual scope. Both bets are made with inadequate information at the moment of signing, and both bets are usually wrong.

The mechanism of failure is predictable:

1. **The spec gap.** No specification survives contact with implementation. Requirements that seemed clear in a document reveal ambiguities when code has to implement them. Edge cases that weren't contemplated become common cases. The vendor encounters something real; the spec says something different.

2. **The change order.** The vendor issues a change order. The buyer reads the spec. Both parties argue about whether the feature was "in scope." This argument is not about technical facts — it's about contract interpretation. The vendor has every incentive to interpret the contract narrowly; the buyer has every incentive to interpret it broadly. Neither party is acting in bad faith. Both parties are rationally responding to the incentive structure they created together.

3. **The quality trade-off.** If the change order is denied, the vendor still has to deliver something. They have three options: absorb the cost (lose margin), delay delivery (breach), or cut quality (ship something that technically meets the spec but isn't well-built). Most vendors, under margin pressure, choose option three. The buyer gets software that works today but is expensive to maintain and extend. The spec said "functional." The software is functional. The contract is closed.

4. **The documentation shift.** As scope disputes develop, the vendor shifts from collaboration mode to documentation mode. Meeting notes become more formal. Emails become more careful. "As agreed in the specification" begins appearing. The buyer doesn't notice immediately, but the vendor has started preparing for a dispute.

What "Fixed Price" Actually Transfers

The buyer who signs a fixed-price contract believes they are transferring financial risk to the vendor. What they are actually doing is transferring the *expression* of risk from a dollar amount to a scope dispute. The uncertainty doesn't disappear — it converts.

In a T&M engagement, uncertainty appears as a cost range: "this will take 3–5 months." In a fixed-price engagement, it appears as a scope dispute: "that feature isn't in the contract." The buyer experiences T&M uncertainty as financial exposure; they experience fixed-price uncertainty as betrayal. The second experience is worse, and the outcomes are often similar in dollar terms.

Reading a Bid

Before signing any custom software contract, a buyer should be able to answer the following questions by reading the proposal:

What are the assumptions? Every estimate rests on assumptions: that the buyer's existing data is clean, that third-party APIs behave as documented, that a stakeholder will be available to answer questions. These assumptions should be explicit. If they're not, ask. If the vendor won't list their assumptions, they may not have made any — which means the estimate is speculative.

What are the client responsibilities? Who provides what, by when? Delays caused by the buyer are often explicitly excluded from the vendor's delivery obligations. Know what you're signing up to deliver.

What is the change order rate? This number — the hourly rate for out-of-scope work — is the effective price of any ambiguity in the spec. A change order rate of $250/hour on a $200K project means that 100 hours of scope dispute costs $25K, or 12.5% of the project. Know this number before you need it.

What are the acceptance criteria? "Functional per requirements" is not an acceptance criterion. It is a pointer back to the spec that both parties will interpret differently. Good acceptance criteria describe user behavior: "a user can complete checkout in fewer than 4 clicks with a valid credit card."

What happens when we disagree about scope? This is the question that predicts more about how a project will go than any other. Ask it explicitly. A vendor who says "we'd look at the spec and determine which interpretation is supported" is operating in fixed-price mode. A vendor who says "we'd look at what you're trying to accomplish and find the least expensive path to get there, and if it's more than we estimated, we'd have a conversation" is invested in the outcome.

III. What T&M Is (and Isn't)

Time-and-materials (T&M) engagements bill actual hours worked. The buyer pays for effort; the vendor delivers against a backlog the buyer controls. This structure is not inherently more expensive than fixed-

price — it is more *honest* about where the cost uncertainty lives.

T&M does not mean open-ended spending. The mechanisms that prevent this:

A prioritized backlog. The buyer owns the backlog — the ordered list of features and work items. At any point, the buyer can stop, redirect, or reprioritize. The budget controls the engagement; the backlog controls what the budget buys.

Fortnightly working demos. Every two weeks, the team demonstrates working software deployed to a staging environment. Not mockups, not slide decks — software that the buyer can click. This is the primary quality-control mechanism: the buyer sees what is being built while there is still time and budget to change direction.

Transparent burn rate. The buyer knows, at any point, how much has been spent and what has been built. There are no surprises. Cost uncertainty is visible in real time rather than hidden until a change order arrives.

A definition of done. T&M doesn't mean perpetual scope expansion. Each item in the backlog has acceptance criteria. Items are complete when they meet those criteria, not when the vendor says they're done.

The Shared Risk Structure

In a well-run T&M engagement, both parties have skin in the project outcome:

- The **buyer** is incentivized to prioritize clearly (wasted time comes out of their budget), give honest demo feedback (course corrections are expensive to defer), and protect the team's focus (context-switching is a real cost).
- The **vendor** is incentivized to surface problems early (delays they hide cost the buyer money and erode the relationship), push back on bad ideas (a feature that takes too long destroys value), and build quality code (they will maintain it and extend it throughout the engagement).

Neither party can succeed unless the other does. This is the structural basis of shared risk. It is not a contractual clause — it is a consequence of the billing model.

IV. The Discovery Phase

The single highest-leverage investment in a custom software project is a paid discovery phase before any development begins.

Discovery costs money. A thorough discovery for a $400K–$600K project might cost $25K–$60K. The discovery investment is the right comparison group: not the project cost, but the cost of building the wrong thing. If discovery prevents a $500K project from delivering a product nobody uses, it has returned 10× its cost.

What Discovery Does

Discovery answers the question: are we building the right thing?

This question cannot be answered from a spec. It requires:

- **User research.** Who actually needs this software? What do they do today? What problems does the current approach cause? What would success look like from their perspective?
- **Technical feasibility.** What are the hard parts? What integrations are required? What is the state of existing data and systems?
- **Prioritization.** Which parts of the stated requirement are essential? Which parts are nice-to-have? Which parts might not be necessary at all, if the underlying problem is understood differently?

A good discovery produces a prioritized backlog, a rough architecture, a cost estimate with a stated range (not a single number), and an explicit statement of what remains uncertain. This is more useful than a complete spec.

The UK GDS Model

The UK Government Digital Service, formed after a series of catastrophic large government IT failures, built discovery-first, iterative delivery into public procurement. The GDS Service Standard requires services to pass through four phases before receiving full funding:

- **Discovery (4–8 weeks):** What problem are users actually trying to solve?
- **Alpha (6–12 weeks):** Prototype and test approaches with real users.
- **Beta (variable):** Build and open to real users. Iterate based on actual use.
- **Live:** Full deployment, with ongoing user research required.

The off-ramp structure is explicit: projects can be killed at any phase if the evidence doesn't support proceeding. This is incremental commitment — the structural embodiment of not betting everything at the moment of maximum uncertainty.

The Discovery Conversation

A good discovery session doesn't sound like a software meeting. The vendor asks questions about work, not technology: *What does a successful Monday morning look like for your team? When this is working well, what will be different about how you start your week? If you could fix one thing about how information moves through your organization, what would have the highest impact?*

The buyer finds themselves describing things they know well but have never articulated: the friction points, the decisions that get made with incomplete information, the reports everyone knows are wrong but everyone uses anyway.

The vendor is looking for the actual problem underneath the stated problem. The stated problem is usually the solution the buyer has already half-designed. The actual problem is the underlying need that solution was meant to address. Often the actual problem has a simpler solution than the stated one.

V. What Good Looks Like

The Four Questions

A useful test that applies at any stage of a well-run engagement: can the buyer answer the following four questions?

1. What was built last week?
2. Is it working?
3. What will be built next week?
4. Is the project on track to be useful within the remaining budget?

If the buyer can answer all four, information is flowing. Decisions are being made in time. The project has a steering mechanism.

If the buyer cannot answer these questions — if they need to ask the vendor and wait for a status report — they do not have the visibility that shared risk requires.

Good Vendor Behavior

A vendor who is functioning as a partner behaves differently from a vendor executing a contract:

Surfaces problems early, with options. Not "we've run into a complexity" but "we've run into an integration issue that will add 3 days to the sprint; here are two ways to handle it." The vendor who has a structural incentive to hide problems (fixed-price vendor losing margin) behaves differently from the one who has every incentive to surface them.

Pushes back on bad ideas. "You've asked us to add a feature that would require rewriting the authentication module. We think that's the wrong call for now — here's why." A vendor who always builds what the buyer asks, without raising concerns, is not serving the buyer's interests.

Admits uncertainty. "We don't know yet how long this will take. We'll know more in about 3 days. We didn't want to wait until next week to tell you." This kind of transparency is only possible in an environment where admitting uncertainty doesn't trigger a cost dispute.

Maintains working software as the output. Not documentation, not progress reports — working software deployed to staging, interactable by the buyer, every two weeks.

Good Buyer Behavior

A buyer who is functioning well in a shared-risk engagement:

Names a real product owner. Not a project manager who coordinates meetings. Someone who actually understands the problem the software is supposed to solve, has authority to make scope decisions, and is available to answer questions during the week.

Gives honest feedback at demos. "Looks good" when it doesn't is a deferred problem. The buyer who is polite at demos and then raises issues at UAT has spent 6 weeks building the wrong thing.

Accepts that requirements will change. The buyer who has internalized that the spec is a hypothesis can say "I've changed my mind about how this should work" without shame. The buyer who believes the spec is a commitment will defend wrong requirements to avoid appearing indecisive.

Protects the vendor's ability to do good work. Doesn't change priority every week. Doesn't add "quick" items mid-sprint. The buyer who respects the development team's focus is investing in the quality of what gets built.

VI. The Experience of Both Kinds of Project

The Sickness

There is a particular sickness that sets in around the fourth invoice of a failing fixed-price project. It is not the sharp panic of a missed deadline, but a dull, metabolic ache.

The meetings develop a ritualistic quality. The vendor's project manager speaks in a cadence of serene, bullet-pointed assurance. "We're tracking to plan." "The integration layer is progressing." "A few complexities, but nothing out of scope." The words are fine. The body hears something else: the slight pause before "complexities," the way "tracking to plan" has become a mantra, devoid of the early excitement about the plan itself.

You find yourself clinging to stray concrete details: "When you say the API is 'securely housed,' does that mean authentication is actually working?" The answer is a paragraph that ends where it began. You leave the call with a dry mouth and a browser tab open to the Statement of Work, a document that now feels less like a blueprint and more like a cryptic poem where every noun has been redefined by the other side.

The Theater of Work

One day the "working demo" arrives. You are asked to log into a staging environment.

It feels like being shown a movie set. You can click the button. It triggers a spinning animation. It does not, however, connect to the live database, because that part is "still being optimized." The backend logic is in a branch, not yet merged. The output is mocked data.

The button is beautiful. The UI is pixel-perfect.

You are torn between a childlike desire to believe in the magic trick and a corrosive understanding that you are being shown the theater of work, not the work itself. Your praise feels hollow. Their acceptance of it feels like collusion. You have become a PR agent for a reality you no longer believe in.

The Loneliness

The most profound loneliness comes from the isolation within your own organization.

You must translate this creeping dread into status updates for your leadership. You polish the vendor's vagueness into something resembling progress. "The team is overcoming technical hurdles with the data

pipeline; this is common in complex integrations." You become a spokesperson for a project you no longer believe in.

You lie to protect your own judgment, because you were the one who championed this firm, this contract, this fixed-price arrangement. To raise a red flag is to indict your past self. So you double down on faith. You convince yourself that the next deliverable will turn the corner.

The Different Rhythm

The shared-risk engagement feels different in its body.

Updates are not status reports delivered to you — they are working sessions you're in. You are not informed of a delay; you are shown the tangled code, the confusing error log, the two possible paths forward. "We're stuck here," they say, sharing their screen. "Option A is a quicker fix but might cause pain later. Option B is cleaner but adds three days. What's more important to you right now?"

The asymmetry evaporates because the problem is placed on the table for everyone to see. Your role shifts from auditor to co-solver. Your domain knowledge — the *why* of the business — is actively mined to help solve the *how* of the technology.

The financial model codifies this psychology. Seeing a weekly or monthly invoice for time is nerve-wracking at first. You are paying for effort, not completion. But this transparency forces a brutal, healthy prioritization. Every invoice asks a silent question: "Was our time this week well-spent, from your perspective?" You find yourself protecting the team's focus, killing pet features, clarifying goals — because you viscerally understand that wasted time is shared loss.

The Pivot Without Disaster

When the shared-risk engagement reveals that the original hypothesis was wrong, it's not a crisis. It's the point.

The data from the first working slice shows that users actually need something different. In a fixed-price engagement, this moment is catastrophic — a scope violation, a reckoning, a change order. In a shared-risk engagement, the conversation is: "The user data is telling us they actually need X. Do we want to change direction?" There is no blame, because there was no guaranteed endpoint to deviate from.

VII. Warning Signs

In the Vendor's Behavior

Watch for these patterns in any engagement:

- Demos that show mockups or mocked data rather than working software connected to real systems
- Status reports that describe progress without showing working software
- Change orders for things that were clearly implied by the original spec
- Language that becomes more formal and more carefully worded over time
- Reluctance to share repository access, burn-down charts, or sprint-level data
- Answers to specific technical questions that are syntactically complete but informationally empty

In Your Own Behavior

Watch for these patterns in yourself:

- Polishing the vendor's vagueness into something that sounds like progress when reporting up
- Defending requirements you no longer believe in because changing them feels like admitting you were wrong
- Approving demos you have doubts about because the atmosphere of the meeting makes objection feel rude
- Adding scope mid-sprint because something feels "quick"
- Delaying a difficult conversation in hopes the project will turn a corner

Questions Worth Asking Any Vendor

Before signing:

1. *What happens when we disagree about whether something is in scope?*
2. *Can you show me examples of projects where requirements changed significantly, and how you handled it?*
3. *What will I see at the end of the first two weeks of development?*
4. *What access will I have to the codebase, issue tracker, and sprint burndown?*
5. *What do you do when you realize you've underestimated?*

The vendor's answers to these questions reveal more about how the project will go than anything in their proposal deck.

What Both Sides Are Afraid Of

A well-meaning buyer is afraid of paying for something that doesn't work. This is rational. The fear produces: wanting a fixed price, a complete spec, and contractual protection.

Each of these protective instincts produces the problem it's trying to prevent. The fixed price converts uncertainty into scope dispute. The complete spec locks in requirements at the moment of maximum uncertainty. The contractual protection creates incentives for the vendor to document against the buyer rather than collaborate with them.

A well-meaning vendor is afraid of losing money on a project they underestimated. This is also rational. The fear produces: tight scoping, broad change order rights, spec-based acceptance criteria.

Each of these protective instincts also produces the problem it's trying to prevent.

Both parties' fears are reasonable responses to real risks. The problem is that each party is responding to risk in a way that increases the other party's risk — and that creates the dynamics both parties were trying to avoid.

The shared-risk engagement is not a solution to these fears. It is an acknowledgment that both fears are real, and a structure that distributes their consequences more honestly. The buyer accepts some cost uncertainty. The vendor accepts some obligation to build toward outcome, not just spec. Both parties accept that requirements will change. Both parties invest in making the project work rather than protecting their position if it doesn't.

Conclusion

In a well-run shared-risk engagement, the final product is born from a series of small, confident choices. It may not look exactly like the original vision. It fits the real need in a way the original vision never could.

When you launch, you feel a quiet, deep ownership. You did not accept delivery of a product; you participated in building a solution. The team celebrates with you, and the celebration is genuine, because their success was inextricably linked to yours from the beginning.

The contrast with a failing fixed-price project is total. In the fixed-price failure, you are a warden, guarding

a prison of requirements you and your adversaries built together. In a shared-risk engagement, you are a fellow traveler, holding one map, pointing out the same cliffs, deciding together which path to take next.

The first feels like a slow suffocation of trust. The second feels like building something on a foundation of shared breath.

References

Standish Group, CHAOS Reports (1994–present). Annual survey of software project outcomes. Consistent finding: roughly a third of projects cancelled or unusable; majority over budget or time. Standish Group International.

McConnell, Steve. *Software Estimation: Demystifying the Black Art* (2006). Source of the "cone of uncertainty" framework: estimate ranges narrow as requirements are clarified and design decisions made. Microsoft Press.

Rasmusson, Jonathan. *The Agile Samurai* (2010). Source of the inception deck methodology: 10 exercises to surface alignment and misalignment before development begins. Pragmatic Bookshelf.

Rittel, Horst, and Webber, Melvin. "Dilemmas in a General Theory of Planning" (1973). Original formulation of "wicked problems" — problems whose solution requires understanding that only comes from attempting to solve them. *Policy Sciences*, 4(2), 155–169.

UK Government Digital Service. Service Manual (2014–present). Documented evidence of systematic procurement reform: discovery → alpha → beta → live phases with explicit off-ramps. Available at gov.uk/service-manual.

18F (General Services Administration). Modular Contracting Guidance. US federal equivalent: break large contracts into small, independently deliverable modules. Available at 18f.gsa.gov.

Beck, Kent, et al. *Manifesto for Agile Software Development* (2001). Source of "working software over comprehensive documentation" and "responding to change over following a plan." agilemanifesto.org.

De Marco, Tom, and Lister, Timothy. *Peopleware: Productive Projects and Teams* (1987, 1999). Classic study of the social and organizational dimensions of software project failure. Dorset House.

Fowler, Martin. "Is Design Dead?" (2004). Articulates the argument for emergent design over big upfront

design in agile contexts. martinfowler.com.

Brooks, Frederick. ***The Mythical Man-Month*** **(1975, 1995).** Source of Brooks's Law and foundational analysis of why software estimation is structurally difficult. Addison-Wesley.