Wicked Problems

https://gerolds.github.io/textbook/textbook/posts/wicked-problems/

Contents

Wicked Problems

There is no shortage of advice for running a game project. Teams are told to find the fun early, cut scope, hire for taste, prove the loop before scaling, commit to a vision, test with players, and ship with discipline. Most of this advice is sound. The problem is that it is usually presented one principle at a time, as though teams were dealing with one clean problem at a time.

Small studios rarely have that luxury. They face several valid demands at once, and the demands often conflict. A team may need to prove the loop, but have funding tied to feature milestones. It may need stronger taste in the room, but only be able to afford juniors. It may need to cut scope, but already be contractually tied to the larger pitch. It may need a clear decision, but have leadership that avoids conflict. None of these constraints is unusual. The difficulty comes from their interaction.

This essay is about that interaction. It is a taxonomy of recurring patterns in small-scale game development: 4–40 people, mixed incentives, limited funding, and teams where each person is structurally load-bearing. At that scale, unresolved contradictions do not stay local. They accumulate in the game, the schedule, the team structure, and the decision-making process.

The goal here is diagnostic more than prescriptive. Some of these contradictions can be solved. Some can only be managed. A few are terminal. The useful step is to name the pattern accurately enough that the team can stop treating it as a vague feeling and start treating it as a real constraint with real consequences.

---

1. Why the advice doesn’t help

Advice fails here for a structural reason, not an intellectual one.

Good advice assumes you can isolate one problem, apply one solution, and observe the result. In practice, a small studio’s problems are coupled. The hiring problem is coupled to the budget problem, which is coupled to the funding structure, which is coupled to the scope, which is coupled to the commitment (or lack

of one), which is coupled to the taste of the people in the room, which is coupled to the hiring problem again. Pull on one thread and three others tighten.

This is why frameworks that work well in talks break down on Monday morning. The framework says “find the fun first.” Leadership agrees. But finding the fun requires a prototype, and the prototype requires someone who can build one quickly, and that person doesn’t exist on the team, and hiring them requires money, and the money is locked to milestones that specify features, not prototypes. The framework was correct. It was also useless, because it assumed a degree of freedom the studio doesn’t have.

A second failure mode of advice is that it describes what a healthy studio does without describing how to become one. It says what stable teams do once they are already stable. It is much less useful on the path from instability to stability.

A third failure is that advice rarely models its own failure modes. It tells you what to do when things go right. It says less about what failure looks like from the inside, which is usually when teams most need guidance. From inside a failing project, every failure mode feels specific. “We know the advice says X, but our situation is different because…” is often enough to keep the current process intact.

Rule: advice that does not model its own failure modes will be absorbed by the problems it was meant to solve.

---

2. **The patterns**

What follows is a catalogue of wicked problems that recur in small studios. They are not bugs. They are structural traps that emerge from the interaction of real constraints: limited money, limited people, misaligned incentives, and the emotional difficulty of honest assessment. Each pattern has a surface that looks reasonable and a cost that is invisible until it compounds.

2.1. **The taste gap**

The studio wants to make good games. Nobody on the team has the taste to know what “good” means at the level the market requires.

This is the most painful pattern to name because it sounds like an insult. It is not. Taste is a specific skill, not a general virtue. It is the ability to look at a prototype, a mechanic, a piece of art, a UX flow, or a moment of play and know whether it serves the commitment. It is the ability to distinguish “this is competent” from “this is the thing people will love.” *Hiring for Games (https://gerolds.github.io/posts/hiring/)* describes taste in detail: it is not aesthetic preference but diagnostic precision.

The taste gap shows up as a studio that ships games that are functional and forgettable. The games work. They have the correct features. Nothing is broken. But nothing creates the pull that makes a player tell a friend about it. The team can describe what they intended but cannot see the distance between the intention and the artifact.

The trap is that the taste gap is invisible to the people inside it. If you could see the gap, you would already be closing it. From inside, the games feel like they should work. The logic is sound. The hours were spent. The market must be the problem, or the budget, or the marketing, or luck.

Why it resists advice: “Hire people with better taste” is correct but circular. If the team could evaluate taste accurately, they would already have it. The taste gap means the studio cannot reliably distinguish between a candidate who has taste and one who merely has credentials or confidence. *Hiring for Games (https://gerolds.github.io/posts/hiring/)* calls this the problem of “the nose”: you need someone who can smell competence, and finding that person requires the ability to smell competence.

The ground-level tactic: The instinct is to find someone who shipped a great game and ask them to evaluate your work. This pitfalls into survivorship: a maker who shipped something excellent may have gotten lucky, may not understand *why* it worked, and almost certainly cannot transfer their instinct to your team by

pointing at your game and saying “this isn’t good enough.” That is a diagnosis without a treatment.

Taste is not uncovered by talking to successful makers. It is developed through *deliberate practice*, and deliberate practice requires a teacher, not a role model. The distinction matters. An excellent teacher does not need to be able to make the thing they teach. They need to know how excellence is built: how to read where a person currently is, design a problem that stretches them exactly one level beyond that, evaluate the result, and repeat. The teacher’s skill is not taste itself but the ability to *produce taste in others* through a sequence of challenges calibrated to the learner.

What the studio needs, then, is not a celebrated developer on retainer. It is someone who can look at the team’s actual output, identify the specific perceptual gap (not “this isn’t good” but “you cannot yet distinguish X from Y, and here is an exercise that will teach you the difference”), and then do that repeatedly, pushing the team to the limit of what the humans executing the advice are capable of reaching. Each round closes the gap a little. Over months, the team’s diagnostic precision improves because it has been trained, not because someone else’s precision was borrowed.

This is the territory described in *The Price of Seeing Clearly (https://gerolds.github.io/posts/the-price-of-seeing-clearly/)*: the floor still has to collapse before a new one can form. But a good teacher controls the collapse. They make it survivable and productive rather than random and destructive.

2.2. **Building without proving**

The team builds systems, art, infrastructure, and content before proving that the core loop works. By the time someone asks “is this fun?”, the answer is buried under months of sunk cost and emotional investment.

This is the single most expensive failure in small-scale game development, and it is staggeringly common. It happens because building feels like progress. A new inventory system, a character model, a dialogue tree, a camera rig: these are visible, demonstrable, and they make the project feel real. Proving the loop

is none of those things. It is ugly, uncertain, and it might reveal that the entire direction is wrong.

Prototyping the Loop *(https://gerolds.github.io/posts/prototyping-the-loop/)* explains what proof looks like: finding the invariant, the relationship that makes visible what moves when the player acts and why they'd want it to. *The Studio Primitive* *(https://gerolds.github.io/posts/studio-os/)* frames proof as the second step of the operating loop (Promise → Proof → Cut → Ship), a step that must happen before production scales.

Why it resists advice: Everyone on the team has heard "find the fun first." They agree with it in meetings. They do not do it because the funding milestones specify features, because building systems feels productive while prototyping feels like stalling, and because an honest prototype might reveal that the game doesn't work. The advice is correct and the incentives point the other way.

The ground-level tactic: The naive version of this tactic is "make the proof slice the first milestone." That is directionally correct but hides three deeper problems.

First: proof is not a milestone, it is a practice. Teams that treat finding the proof as a one-time event will find it (or convince themselves they found it), check the box, and proceed into production without ever revisiting the question. But the loop needs to be re-proven at every scale transition. A loop that works in a three-minute prototype may collapse when content, progression, and pacing are layered on. Proof is not a gate you pass through. It is a question you keep asking: *does the thing that made this work still work, given everything we've added since?* Funding structures that tie money to sequential milestones are allergic to this kind of iteration. They reward forward motion and penalize revisiting. The team must develop the internal discipline to separate external reporting (milestones the funder sees) from internal organization (how the work actually happens). Milestones are a communication contract with a funder. They are not a production methodology.

Second: proof should precede funding, not follow it. Ideally, the proof slice is why the funding was obtained. The studio walks into the pitch with a playable artifact that demonstrates the loop, and the money is for scaling what has already been proven. If that

did not happen, if the funding was won on a pitch document and the loop has never been tested, the team is in a harder position but not a hopeless one. The move is to negotiate a contract and project-management structure that enables “late proof”: a funded discovery phase where the deliverable is not a feature list but a validated loop, with explicit room for the pivots that discovery requires. This is uncomfortable for funders who want certainty, but the alternative is funding a production that builds on an unproven foundation and discovers the problem at ten times the cost.

Third: this puts enormous pressure on the pitch. The people selling the project need to be specific enough to be credible and open enough to not lock the team into a contract that makes honest discovery impossible. Selling a rigid feature list feels safe but creates the trap. Selling a process, with a clear commitment and a defined proof gate, requires a funder who understands how games are actually made. Finding that funder, or educating the ones you have, is itself a load-bearing skill that most studios underinvest in.

If none of this is achievable, if the funding is locked, the milestones are rigid, and the proof was never done, the studio needs to accept that it is operating inside the trap described in *The Dream That Won’t Compound (https://gerolds.github.io/posts/the-dream-that-wont-compound/)*: the money teaches the team to complete a checklist, not to make a game.

Acknowledging the trap does not escape it, but it stops the team from mistaking compliance for progress. It may also free one or two people to work on the proof in parallel, even if the official schedule does not account for it. Sometimes the most important work on the project is the work that does not appear on the milestone chart.

2.3. **Compensation mythology**

> *We can’t afford good people, so we’ll compensate with process / passion / long hours / clever scope management.*

This is the myth that allows a studio to operate below the quality threshold required to compete while feeling responsible about it. It takes many forms:

- *We'll train juniors up.* Training requires a senior who has time and ability to teach. If the team had that person, they would not need the juniors.
- *We'll use middleware and templates.* Middleware solves implementation problems, not design problems. The gap between a game built from templates and a game with real pull is not in the code.
- *We'll work harder.* Effort compounds only when pointed in a productive direction. Effort pointed at the wrong thing produces a polished artifact that nobody wants.
- *We'll be scrappy.* Scrappiness works when the team knows what to cut and what to protect. Without taste, scrappiness is just under-resourcing dressed in virtue.

Why it resists advice: "Hire better people" is not actionable when the budget is fixed. The studio knows it needs stronger talent and cannot afford it. The compensation myth is not delusion. It is a coping mechanism for a real constraint. The problem is that the coping mechanism prevents the team from confronting the constraint honestly: *if we cannot afford the people required to execute this vision, the vision must change to match the people we have.*

The ground-level tactic: Scope to the team, not the aspiration. *Making the Thing (https://gerolds.github.io/posts/making-the-thing/)* frames the governing commitment as the thing you are unwilling to betray. If the team cannot execute a commitment at the quality level required, the commitment must shrink until it fits the team's actual capability. A small, proven, coherent game built by the people you have is worth more than an ambitious, unproven, incoherent game built by the people you wish you had.

2.4. **The dual narrative**

Every pitch is a simplification. That is fine. The problem starts when the studio maintains two simplifications that point in different directions and treats both as true.

One story faces outward: investors, publishers, grant bodies. It emphasizes market size, feature scope, monetization, return. The other story faces inward: the team's private belief about what the

game really is. It emphasizes feel, authorship, the thing that makes this game worth making. The two stories overlap somewhere, but studios are rarely honest about how small the overlap is. Instead they maintain both narratives in parallel and let the project live in the gap.

The gap is where the lying starts. Not malicious lying. Structural lying. The pitch promises a scope that justifies the budget. The team believes in a different, smaller, more specific game. Neither side names the discrepancy because the money depends on not naming it. Over months, the artifact under construction drifts toward neither description. It tries to satisfy both. Features demanded by the pitch get bolted onto a core that was designed for a different kind of experience. A systems layer appears because the investor story mentioned “emergent gameplay.” A narrative layer appears because the team believes in authored moments. A progression economy appears because the pitch deck cited retention metrics. Each addition is locally justified by one of the two stories. None of them cohere, because they were never derived from a single commitment. The result is multiple games inside one executable. If the same effort had been spent on any one of them, it might have shipped something coherent. Instead the studio ships an incoherent compromise, or doesn’t ship at all.

The lies are not only about scope. They can be about genre, about promised features, about the feel of the game, about what kind of player it serves. A pitch that says “action RPG with deep crafting” and a team that believes “tight combat game with minimal downtime” are describing different products. The investor expects one. The team builds the other. The game that results is neither, and both sides feel betrayed.

The team-size trap. The dual narrative is also what escalates headcount beyond what a specific game needs. A pitch promising a large-scope game justifies a larger team. But team size shapes what gets built, not just how fast. Six people prototyping a concept will inevitably prototype a game that a six-person team cannot finish, because the prototype absorbs the full bandwidth of everyone in the room and scales to fill their collective ambition. The same prompt given to two people produces a smaller prototype, one that a two-person team *can* finish and prove. The six-person prototype looks more impressive in the milestone

review. It is also a more expensive lie, because it implies a production that requires twenty people and a budget the studio does not have. The team was sized to the pitch, not to the game. The game that would actually work at this budget was never explored because the room was too crowded to find it.

Why it resists advice: “Align your stakeholders” assumes the alignment is possible. Often it is not. The investor wants a product that serves a market. The team wants a work that expresses a vision. The dual narrative is not a communication failure. It is a structural conflict between who pays and who builds. And the lies compound: a lie about scope becomes a lie about team size, which becomes a lie about timeline, which becomes a lie about what the game is.

The ground-level tactic: Find the overlap before the money arrives, and have the courage to reject funding that requires a narrative you do not believe. If the funding is already locked, name the conflicts explicitly within the team. A team that knows it is navigating a contradiction can make deliberate compromises. A team that pretends the contradiction doesn’t exist will discover it as production friction, morale erosion, and a game that tries to serve two masters and satisfies neither. And when scoping the prototype, match the prototype team to the production team. If you will ship with four people, prototype with two. The prototype must prove the game that *this team* can finish, not the game that would impress a funder in a slide deck.

2.5. **Comfort as default**

The team optimizes for comfort: avoiding conflict, avoiding hard calls, avoiding cuts, avoiding honest assessment. Meetings end with vague agreement. Critiques are softened until they carry no information. Decisions are deferred because making them would hurt someone’s feelings.

The Studio Primitive (https://gerolds.github.io/posts/studio-os/) names this directly: “Kindness gets expensive when it refuses to distinguish between care for people and avoidance of decisions.” Comfort feels like team health. It is the opposite. A team that cannot have hard conversations is a team that cannot improve,

and a team that cannot improve is on a trajectory that ends at mediocrity or exhaustion.

Where comfort hides:

- **Avoiding cuts.** A feature stays because someone worked hard on it. The cost of keeping it is distributed across the project as diluted coherence. Nobody prices the dilution because pricing it would mean telling someone their work doesn't serve the game.
- **Not heeding canaries.** Someone on the team sees the problem early. They say it once, maybe twice. The room doesn't want to hear it. They stop saying it. Six months later the problem is a crisis, and the person who saw it has either left or disengaged. *The Price of Seeing Clearly (https://gerolds.github.io/posts/the-price-of-seeing-clearly/)* explains why the machinery of comfort rejects threatening information: the information is not processed as data but as an attack on the model the team lives inside.
- **Leadership hedging.** The creative director sees two paths and won't choose. Both get resourced. Both get built halfway. Neither is tested. The team does twice the work for half the clarity, and the eventual choice, forced by deadline rather than conviction, produces the worst of both.
- **Expensive detours.** Someone in leadership gets excited about a feature, a tool, a technology. The team pivots to accommodate. Weeks or months are spent. The detour doesn't serve the commitment. Nobody says so until the cost is sunk. The pattern repeats because the social cost of challenging leadership exceeds the perceived cost of the detour, until the detours accumulate into a project that cannot ship.

Why it resists advice: "Make hard calls early" is advice that assumes someone wants to hear it. The structure of comfort is specifically designed to prevent hard calls. The advice is repelled by the same immune system it is trying to penetrate.

The ground-level tactic: Install a mechanism that forces confrontation with evidence at regular intervals. *The Studio Primitive (https://gerolds.github.io/posts/studio-os/)* uses the proof slice for this: a playable artifact that makes the state of the game visible to everyone, including the things nobody wants to see. If the proof

slice is ugly, the game is ugly. If the loop doesn't hold attention, no amount of meeting consensus will fix it. The artifact does the arguing that people are afraid to do.

2.6. **Toxic optimism**

Toxic optimism is not about saying "it'll come together." That is just the verbal tic. The real pathology sits deeper: a fundamental rejection of failure as a possibility, or — distinctly — a rejection of failure as something that could happen *to this team, on this project*.

These two are different. A team that rejects failure as a possibility believes the game will work because it must. A team that rejects failure as applicable believes it will work because *they* are the ones making it. The first is naïveté. The second is exceptionalism wearing different clothes. Both produce the same result: the team does not build the safeguards that failure-aware teams build, because safeguards are for projects that might fail, and this one won't.

The toxic part is not the belief itself. Belief is necessary. You cannot sustain multi-year effort without some conviction that the destination is reachable. Good optimism is belief in the solvability of hard problems. It says: "This problem is enormous and we respect that, and we believe that disciplined, terrified, daily effort can solve it." Good optimism is terrified today of not doing enough to keep the foundations stable, simple, coherent, and aligned. It earns tomorrow's confidence by protecting today's core.

Toxic optimism is the irrational belief that coherence, alignment, and quality will emerge on their own. That the pieces will snap together later. That the core will hold without active protection. That the architecture will stay clean without daily vigilance. It does not have to be verbalised this way. People on the team may still see problems, talk about them, perform cuts and pivots. But if the decisions do not scream *protect the coherent core*, the optimism is toxic. Decisions speak louder than post-mortems. A team that says "we need to find the fun" and then spends three months building inventory systems has told you what it actually believes.

The poison pill is not a sentence anyone says out loud. It is the assumption, embedded in daily choices, that the hard part will solve itself if the team keeps building. It won't. The hard part — proving the loop, protecting the commitment, maintaining coherence under production pressure — requires active, uncomfortable, often frightening work every day. A team that is not frightened is not paying attention.

The telltale signs:

- The game has never been playtested by strangers, or playtests are dismissed as "they didn't get it."
- Internal demos are narrated by a founder explaining what the experience *will be*, rather than showing what it *is*.
- Criticism is met with "yes, but when we add X it'll all make sense."
- The team consistently estimates that the game is 70% done, for months running.
- Post-mortems focus on external factors (market, timing, budget) rather than the quality of the work itself.
- Decisions do not visibly prioritise the core loop over secondary systems. The team builds outward before building down.

Why it resists advice: Toxic optimism is not ignorance. It is a defense mechanism. The team knows, at some level, that the evidence is thin. But confronting that would mean questioning the project, the investment, the years of work, and possibly the team's ability to do what it set out to do. The cost of honesty feels existential. The cost of optimism feels like nothing, until the money runs out. And because the optimism lives in decisions rather than in words, it survives every conversation about process improvement. The team agrees to change. The decisions don't.

The ground-level tactic: Create tests you cannot talk your way out of. Put the game in front of strangers with no explanation and watch. Record the session. Watch it together as a team. The recording is the mirror that optimism cannot fog. If the player is confused, the game is confusing. If the player quits, the game is not holding. No narrative can survive a recording of a player having a bad time.

2.7. **Scope blindness**

The team cannot see that the project exceeds what the budget, team, and timeline can deliver. Features accumulate because adding is easier than cutting, and because each addition seems small in isolation. The cumulative scope is never honestly assessed against the actual resources.

This is related to compensation mythology but distinct. Compensation mythology is about people. Scope blindness is about the project's shape. A team with excellent people can still be scope-blind if nobody is willing to do the arithmetic: *how many person-months does this game require, and how many do we have?*

Why it resists advice: "Cut scope" is universally agreed upon and almost never done. Cutting is painful because every feature has an advocate, and removing a feature is experienced as removing that person's contribution. The social cost of cutting exceeds the perceived production cost of keeping, right up until the moment the project collapses under its own weight.

The ground-level tactic: Price every feature in person-weeks, publicly, on a board the whole team can see. Put the total next to the actual available person-weeks. The gap is the project's reality. If the gap cannot be closed by cutting, the project is already failing and the only question is whether the team acknowledges it now or later. *Making the Thing (https://gerolds.github.io/posts/making-the-thing/)* frames this as a coherence question: every feature that does not serve the commitment is guilty until proven innocent.

2.8. **Ignoring the ossification clock**

Decisions harden whether the team manages the process or not. A name chosen in week two becomes the word the team thinks with. A prototype hack promoted to production becomes the architecture the game ships on. A placeholder mechanic survives because removing it would mean rewriting everything downstream.

What We Do Here Today Will Echo in Eternity (https://gerolds.github.io/posts/echo/) describes this mechanism in detail: ossification is the process by which reversible decisions become load-bearing

structure. It happens in code, art, naming, player expectations, and team vocabulary. The further toward the frozen end of the gradient a decision sits, the more it deserves deliberate evaluation before production grows around it.

The wicked problem: Teams that don’t recognize ossification treat every decision as equally revisable. “We’ll fix it later” becomes the mantra. But “later” never arrives because the cost of fixing increases with every decision that builds on top of the original. By the time the team realizes the early decision was wrong, the cost of correction exceeds the cost of living with it, and the project is shaped by its worst early choices.

Why it resists advice: “Get the important decisions right early” is correct but assumes the team can identify which decisions are important. The ossification gradient is not obvious in advance. A name, a folder structure, a data model: these feel trivial at the time of decision. Their importance becomes apparent only after other decisions have grown around them. And “audit at phase transitions” assumes there are phase transitions. In practice, ossification does not wait for milestones. It happens daily, in every commit, every naming choice, every promoted placeholder, every meeting where “good enough for now” becomes permanent by default.

The ground-level tactic: Accept that you will rebuild. Not might. Will.

A team without exceptional skill and exceptional luck will build the game two to five times before it is good. The first build teaches the team what the game is not. The second teaches them what it might be. The third is where the loop, the structure, and the feel begin to converge. Lucky guesses early on can shorten this, but budgeting for luck is toxic optimism by another name.

The tactic, then, is not to prevent ossification. It is to *budget for the cost of demolition*. Assume that ossified structure will be expensive and emotionally hard to throw away. Assume that people will resist because they built it and because it works, in the narrow sense of “the code runs.” The cost of keeping bad structure is distributed invisibly across the project as friction, workarounds, and incoherence. The cost of tearing it out is concentrated and painful.

Teams that cannot eat the concentrated pain end up living with the distributed poison.

This means the schedule, the budget, and the team's psychological contract with itself must include rebuilding as a normal activity, not a failure. A team that treats a rebuild as evidence that something went wrong will avoid rebuilds until avoidance is more expensive than the rebuild would have been. A team that treats rebuilding as the expected cost of learning will do it early, when the cost is small, and will produce a game whose foundations were chosen deliberately rather than inherited from a prototype that was never meant to ship.

2.9. **Pattern blindness**

The team has someone, usually an experienced lead, a producer, or a consultant, who can see the trajectory of the project based on patterns, extrapolation, and first principles. They have seen this before. They know what happens next. The team does not listen.

This is not a failure of communication. It is a failure of epistemology. The person who can see the future is not making a prediction. They are reading a pattern: "When a team has no proof slice at month six, the project ships twelve months late or not at all." "When the creative director cannot describe the loop in one sentence, the loop does not exist." "When playtests produce polite feedback instead of excitement, the game is forgettable."

These are funnels, not prophecies. The pattern-reader is saying: "Given this input, the output is almost always this." The team hears a pessimistic opinion and files it under "they don't believe in us."

Why it resists advice: The team's identity is invested in the idea that their situation is special. *The Price of Seeing Clearly (https://gerolds.github.io/posts/the-price-of-seeing-clearly/)* explains the mechanism: the old mindset is not a position you hold but a lens you see through. The team cannot examine its own assumptions while looking through them. Pattern warnings are absorbed by the immune system and reframed as negativity, lack of vision, or not understanding the project.

The ground-level tactic: Make the patterns testable. Instead of arguing about whether the project is on a bad trajectory, agree on a test: “If the next playtest does not produce at least one moment of unprompted enthusiasm from a stranger, we revisit the loop.” The test removes the argument from the realm of opinion and puts it in the realm of evidence. People who resist patterns will sometimes accept experiments.

2.10. **The exceptionalism trap**

Underlying many of these patterns is a single meta-belief that almost nobody states out loud and almost everybody holds: *the rules apply to other teams, not to us.*

This is not hubris in the dramatic sense. Nobody stands in a meeting and declares “we are exceptional.” The belief is quieter than that. It lives as a feeling: a background conviction that the team’s passion, taste, intelligence, or sheer willingness to suffer will compensate for structural deficits that would sink a lesser group. It is a coping mechanism, not a boast. Facing the alternative — that the team is ordinary, that the structural problems are real, that the project might fail for the same boring reasons most projects fail — threatens identity. Not professional identity. *Personal* identity. The story the founders tell themselves about who they are and what they are capable of.

This is why exceptionalism does not respond to argument. If you tell someone caught in it “you aren’t good enough and you won’t make it,” they are not merely offended. They are *shocked*. Even in private, even when the evidence is laid out, the idea is rejected at a level deeper than reason. It is not processed as feedback. It is processed as an attack on the self. The machinery described in *The Price of Seeing Clearly* *(https://gerolds.github.io/posts/the-price-of-seeing-clearly/)* applies in full: the old mindset is not a position you hold, it is a lens you see through. You cannot examine a lens while you are looking through it.

Because exceptionalism is felt rather than stated, it does not show up in the words a team uses. It shows up in the decisions. A team that skips the proof slice is not saying “we don’t need proof.” They are feeling “we know this works.” A team that doesn’t hire for taste is not saying “we don’t need taste.” They are feeling “we have

enough.” A team that doesn’t cut scope is not saying “we can ship all of this.” They are feeling “we’ll find a way.” The feeling precedes the decision, and the decision is never examined because the feeling was never articulated.

The result is a team that does everything a good team would do — holds meetings, runs sprints, reviews builds, talks about the player — but none of it connects to the actual state of the game, because the actual state of the game is filtered through a belief system that cannot admit the game might not be working.

Why it resists advice: Exceptionalism is the immune system that repels all other advice. Every framework, every diagnostic, every warning can be absorbed by it. Not dismissed — that would require conscious engagement. Absorbed. The team nods, agrees, and continues unchanged, because the agreement happened at the verbal level and the exceptionalism lives below that.

The ground-level tactic: Exceptionalism breaks when the evidence becomes undeniable, and undeniable means *felt*, not argued. This is why *The Price of Seeing Clearly (https://gerolds.github.io/posts/the-price-of-seeing-clearly/)* describes mindset change as a wound, not a decision. The floor collapses. Until it does, the best tactic is to create as many opportunities for evidence as possible: playtests, proof slices, honest external critiques, shipped artifacts that face the market. Each one is a chance for reality to penetrate the mythology. Most will be absorbed. Eventually, one won’t.

2.11. **Negotiating with the inevitable**

Someone identifies a constraint. The constraint produces a hard conclusion: “We cannot do X and Y. We must choose.” Instead of choosing, the team negotiates. They look for the compromise that lets them keep both. They muddle through. They split the difference. They do a little of X and a little of Y, poorly, instead of all of one thing, well.

This is not the same as weighing tradeoffs. Weighing tradeoffs is the work of strategy: you accept that something must be lost, you name what you’re losing, and you commit to the path that remains. Negotiating with the inevitable is the refusal to accept

that something must be lost at all. It is the belief that wanting everything hard enough will make everything affordable.

The pattern fires whenever a clear analysis produces an uncomfortable “can’t” or “won’t.” The analysis might be about scope: the game as pitched cannot be built by this team in this time. It might be about people: the team does not have the skill to execute this direction. It might be about the loop: this mechanic and that mechanic are structurally incompatible. The conclusion in each case is the same — *something must go* — and the team’s response is to reject the conclusion while accepting the analysis. They agree the constraint is real. They just refuse to let it constrain.

What follows is a campaign of small compromises. Each one is defensible in isolation. “We’ll keep both systems but simplify them.” “We’ll do the ambitious version but cut corners on polish.” “We’ll ship both features but with less content.” Each compromise degrades the thing it was meant to preserve. The simplified systems are no longer deep enough to justify their existence. The ambitious version without polish feels broken. The features without content feel hollow. The team has not chosen the best path. It has chosen all paths at reduced quality, which is worse than any single path at full quality.

The damage is not just to the game. It is to the team’s ability to achieve clarity. Every negotiation with the inevitable is a refusal to commit, and commitment is what produces coherence. *Finding the Commitment* *(https://gerolds.github.io/posts/finding-the-commitment/)* argues that a held commitment compounds. The team that negotiates holds nothing. It accumulates compromises instead of compounding decisions. The result is a project shaped by what nobody was willing to give up rather than by what anyone was willing to protect.

The deepest version of this pattern is when the negotiation actively sabotages the people trying to produce clarity. Someone identifies the hard choice. Someone else starts looking for the workaround. The workaround consumes time and attention that should have gone to the chosen path. The person who identified the choice watches the team spend weeks avoiding a decision that would

have taken an afternoon. They learn that clarity is punished and muddling is rewarded. They stop trying.

Why it resists advice: “Make hard choices” is the advice. The team believes it *is* making hard choices. From inside the pattern, the compromises feel like pragmatism. “We found a way to do both” sounds like resourcefulness, not avoidance. The team cannot see that the cost of doing both is higher than the cost of doing either, because the cost is distributed across every surface of the project as diluted quality and lost coherence.

The ground-level tactic: When a constraint produces a binary — X or Y, not both — force the decision within a fixed, short window. A day, not a week. The longer the window, the more room there is for negotiation to creep in. Name what is being given up. Write it down. Grieve it if necessary. Then build the thing you chose with the full resources that would have been split. The grief is real, but it is cheaper than the slow death of trying to have everything.

---

3. **How the patterns compound**

These patterns rarely appear alone. They form clusters that reinforce each other, and the clustering is what makes them wicked.

The classic spiral: *The taste gap* means the team *builds without proving* because nobody can tell the loop doesn’t work. *Compensation mythology* prevents hiring someone who could. *The dual narrative* locks scope to a pitch that was written to win funding, not to describe the game. *Comfort as default* prevents anyone from naming the problem. *Toxic optimism* fills the silence. *Scope blindness* keeps the feature list growing. *Ignoring the ossification clock* hardens bad decisions into load-bearing structure. The person who sees the pattern is dismissed. *The exceptionalism trap* seals the system shut. And when someone finally forces a clear choice, the team *negotiates with the inevitable* and splits the difference instead of committing.

By the time the project ships, or fails to ship, the team has a coherent explanation for what went wrong that does not reference any of these patterns. The market was tough. The budget was too

small. The timeline was too short. The publisher didn't support them. All of these may be true. None of them are the root cause.

The root cause is that the team never built a process that forced honest confrontation with the state of the game while there was still time to change it.

This is what *The Dream That Won't Compound (https://gerolds.github.io/posts/the-dream-that-wont-compound/)* describes at the studio level: a team that survives on grants and subsidies, shipping competent but forgettable games, never feeling the bottom-line risk that would force a reckoning. The patterns in this essay are the project-level version of the same dynamic.

---

4. Jungle tactics

If the patterns describe the swamp, what follows are the tactics for moving through it. They are not a strategy. A strategy requires a clear position, a clear objective, and the freedom to choose a path. Most small studios have none of those. These are the actions that get you to the clearing where strategy becomes possible.

4.12. Tactic 1: Name the contradiction

Most teams sense their contradictions but never say them out loud. Saying them out loud is the first move because it stops the team from wasting energy pretending the contradiction doesn't exist.

Write them on a board. "We need to prove the loop, but the milestones require features." "We need to cut scope, but the pitch promised this scope." "We need better taste in the room, but we can't afford it." Each contradiction, once named, becomes a design problem rather than an ambient source of dread.

Not all contradictions are resolvable. Some must be lived with. But naming them changes the team's relationship to them. Instead of "why is everything so hard?", the conversation becomes "we have three structural constraints, which one are we going to actively manage and which two are we going to accept?"

4.13. **Tactic 2: Build the mirror**

Every tactic in this essay converges on a single principle: create artifacts that show the team the truth about the game, whether the team wants to see it or not.

The proof slice is a mirror. A recorded playtest is a mirror. A feature board priced in person-weeks is a mirror. A recorded demo with no narration is a mirror. A screenshot of the game next to a screenshot of the game's competition is a mirror.

Mirrors work because they bypass the social dynamics that prevent honest conversation. Nobody has to be the bad guy. The artifact does the talking. "The playtest recording shows the player quitting at minute four" is not an opinion. It is a fact. The team can argue about what it means, but they cannot argue about whether it happened.

4.14. **Tactic 3: Shrink until it fits**

When the project exceeds the team's capability, the standard advice is to "cut scope." This framing is too gentle. What is actually required is to shrink the commitment until the team can execute it at the quality level required to compete.

This is different from cutting features. Cutting features preserves the commitment and removes the parts that don't serve it. Shrinking the commitment means accepting that the team's reach has exceeded its grasp and finding the smaller version of the idea that the team can actually prove, build, and ship at a level that creates pull.

Finding the Commitment (https://gerolds.github.io/posts/finding-the-commitment/) argues that a held commitment compounds. A commitment that has been shrunk to fit is still a commitment. A commitment that has been preserved at a scope the team cannot execute is not a commitment. It is a wish.

4.15. **Tactic 4: Install the canary system**

A canary in a coal mine is useful only if someone checks on it. The person on the team who sees the pattern, who noticed the problem three months before it became a crisis, is a canary. If the team's

culture penalizes that person for speaking up, the team has disabled its own early-warning system.

Installing the canary system means creating regular, structured moments where the canary can speak without social penalty. Not a suggestion box. A ritual. A weekly session where the explicit question is: “What is the thing nobody wants to say?” with the explicit agreement that saying it is valued, not punished.

This sounds like therapy. It is production infrastructure.

4.16. **Tactic 5: Make the first move cheap**

Many of the patterns in this essay persist because the perceived cost of fixing them is enormous. Proving the loop means admitting the current direction might be wrong. Hiring for taste means admitting the team’s taste is insufficient. Cutting scope means admitting the pitch was too ambitious.

The tactic is to make the first move small enough that it doesn’t trigger the immune response. Not “let’s rebuild the game around a new loop.” Instead: “Let’s spend one week building the smallest possible prototype of the loop and testing it.” Not “let’s fire half the team.” Instead: “Let’s bring in one external critic for a day and listen to what they say.”

Small moves accumulate evidence. Evidence is harder to dismiss than arguments. And evidence, over time, is what breaks exceptionalism.

4.17. **Tactic 6: Protect the retreat**

Sometimes the honest assessment is that the project cannot be saved in its current form. The loop doesn’t work. The scope cannot be cut far enough. The team doesn’t have the capability. The funding structure is incompatible with making a good game.

In military terms, this is a retreat, and retreats need to be protected because they are when armies are most vulnerable. A studio in retreat, cancelling a project, pivoting hard, downsizing, can lose its best people, its morale, and its remaining funding if the retreat is chaotic.

A protected retreat means: naming the decision clearly, explaining it honestly, preserving the relationships and the institutional learning, and using whatever resources remain to set up the next attempt from a position of honesty rather than denial. A studio that retreats well can try again. A studio that refuses to retreat until the resources are exhausted cannot.

---

5. **The clearing**

The jungle is not permanent. Teams do find clearings. They arrive there not by following a map but by accumulating enough honest evidence that the path forward becomes visible.

The clearing looks like this: the team has a commitment it believes in and can articulate in one sentence. The loop has been proven with a playable artifact that holds a stranger’s attention without explanation. The scope fits the budget and the people. The contradictions have been named and the team has chosen which ones to accept and which to actively manage. The people who can see patterns are listened to, and the evidence is checked regularly against reality.

This is not a utopia. It is a starting position. From here, the team can actually work: making decisions that compound, building on a foundation that has been tested, shipping something coherent.

Everything else in this textbook, the commitment, the loop, the proof slice, the ossification management, the hiring, the taste, the vision, assumes you have reached the clearing. This essay is about the part before that: the jungle, and how to survive it.

Rule: the goal is not to have a strategy. The goal is to reach a position from which strategy is possible. Everything before that is survival.

---

6. **Appendix: further reading**

Wicked problems (design theory). Horst Rittel and Melvin Webber, “Dilemmas in a General Theory of Planning” (1973). The original formulation: wicked problems cannot be definitively

described, have no stopping rule, and every solution attempt changes the problem. Game development at small scale is a wicked problem by this definition.

Cynefin framework. Dave Snowden's complexity model distinguishes between simple, complicated, complex, and chaotic domains. Most small studios operate in the complex or chaotic domains while applying tools designed for the complicated domain. The mismatch explains why "best practices" often fail to transfer.

The Abilene Paradox. Jerry Harvey (1974). A group collectively decides on a course of action that no individual member actually wants, because each assumes the others want it. The comfort pattern (pattern 5) is often an Abilene paradox: nobody on the team believes the current direction is working, but nobody says so because they assume everyone else is committed.

Thinking in Bets. Annie Duke (2018). The distinction between decision quality and outcome quality. A good decision with a bad outcome is still a good decision. Studios that conflate the two learn the wrong lessons from both successes and failures.

Skin in the Game. Nassim Nicholas Taleb (2018). Asymmetry of risk. When the people making the decisions are not exposed to the consequences, the decisions predictably degrade. Grant-funded studios with no revenue pressure are structurally insulated from consequence, which is precisely what *The Dream That Won't Compound* *(https://gerolds.github.io/posts/the-dream-that-wont-compound/)* describes.

The Mythical Man-Month. Fred Brooks (1975). Adding people to a late project makes it later. The coordination tax described in pattern 3 and in *Hiring for Games* *(https://gerolds.github.io/posts/hiring/)* is Brooks's law applied to game development.

Further reading within this textbook:

- *The Studio Primitive* *(https://gerolds.github.io/posts/studio-os/)* — the Promise → Proof → Cut → Ship operating loop that addresses patterns 2, 5, 6, and 7
- *Prototyping the Loop* *(https://gerolds.github.io/posts/prototyping-the-loop/)* — finding the invariant, the core of pattern 2

- *Making the Thing (https://gerolds.github.io/posts/making-the-thing/)* — deriving a coherent game from a governing commitment
- *Finding the Commitment (https://gerolds.github.io/posts/finding-the-commitment/)* — why held commitments compound and swapped ones don’t
- *What We Do Here Today Will Echo in Eternity (https://gerolds.github.io/posts/echo/)* — the ossification mechanism behind pattern 8
- *Hiring for Games (https://gerolds.github.io/posts/hiring/)* — taste, evaluation, and the structural cost of wrong hires
- *Studio DNA (https://gerolds.github.io/posts/studio-dna/)* — reading what a team is adapted for and where its blind spots live
- *The Dream That Won’t Compound (https://gerolds.github.io/posts/the-dream-that-wont-compound/)* — the studio-level version of these project-level patterns
- *The Price of Seeing Clearly (https://gerolds.github.io/posts/the-price-of-seeing-clearly/)* — why mindset change cannot be willed, only survived

---

Drafting assistance: Claude. All claims mine; errors my responsibility.