

Anti-Pattern	Warning Signs & Early Signals	Why It Happens (Root Causes)	Impact on Delivery & Team Health	How to Fix / Prevent	Workshop Prompt / Exercise
Hero Developer / “Rock Star” (Over-reliance on one individual)	– One developer is always rescuing projects or making key decisions. – Knowledge and critical skills are concentrated in one person. – Others frequently wait for <i>their</i> input or approvals.	– Organization rewards firefighting & individual heroics. – Lack of cross-training or fear of job insecurity causing <i>knowledge hoarding</i>. – Past crises created dependency on the “star” solver.	– Team becomes over-dependent on one person (low “bus factor”), heightening risk if they leave. – Other members feel disempowered, leading to low morale. – Burnout for the hero and frustration for the rest of the team.	– Establish <i>knowledge sharing</i> practices: pair programming, code reviews, documentation. – Rotate critical responsibilities to reduce silos. – Recognize collaborative success, not just individual “saves.”	Discussion: “How would our team cope if our go-to person went on leave for a month? What skills or info would we miss? How can we share that knowledge?”
Silo Mentality (“Not My Job” & Over-the-Wall Handoffs)	– Different functions (dev, QA, design, ops) work in isolation, communicating only via formal handoffs. – Stakeholders complain of <i>being left out</i> or surprised by late changes. – Team members	– Organizational silos: teams or departments are structured separately, discouraging daily cross-functional contact. – <i>One person per specialty</i> supporting many teams (e.g.	– Misaligned priorities and delays due to late feedback or rework. – Frustration: team members feel like <i>blockers</i> when stretched across projects,	– Form cross-functional teams where all roles collaborate on goals. – Bring all disciplines into early planning, design, and reviews (no more “over-the-wall” throwing requirements or code). – Invest in adequate staffing to avoid one	Exercise: Role swap: Each function (dev, QA, product, design, etc.) explains their processes & pain points to others. Discuss where handoffs

	show little understanding of each others' roles or constraints.	single QA or designer shared) leads to context switching and limited participation. – Lack of processes encouraging early involvement of all roles.	causing stress and demotivation. – Missed opportunities for innovation due to narrow perspectives.	person spread across many teams.	occur and how to involve each other earlier.
Us vs. Them Mindset (Product vs. Engineering “Wars” & blame game)	– Team members align more with their <i>function</i> (e.g., dev vs product) than with the project. – Frequent complaints or finger-pointing about “the other side” (“They always change requirements” vs “They always over-complicate things”). – Issues tossed over the wall and “owned” by a different team instead of solved collectively.	– Competing goals or silos (e.g., separate hierarchies for product vs engineering) breed rivalry. – Historical mistrust or past conflicts between departments. – Lack of shared success metrics (each function optimizes for its own KPIs).	– Slower, less creative delivery as energy is wasted on turf battles instead of problem-solving. – Poor morale: “us vs them” environment harms trust and teamwork. – Risk of <i>waterfall behavior</i>: requirements thrown over the wall with minimal collaboration.	– Align on shared goals (customer outcomes vs departmental KPIs). – Build empathy: hold joint retrospectives/meetings with both groups to discuss issues openly and set ground rules for collaboration. – Cross-train and perhaps <i>co-locate (physically or virtually)</i> to build relationships.	Discussion: “Reflect on a time when our team said ‘they’ (another team) were the problem. What was really happening? What could we have done together instead of working at odds?”
Lack of Trust & Psychological Safety (Fear of speaking up)	– Team members hesitate to share ideas or concerns; meetings are quiet except for a few	– No psychological safety: fear of repercussions or judgment for raising issues leads to	– Undiscovered errors, delays: problems fester until they become crises.	– Set the tone: leaders should model vulnerability, admit mistakes, and ask for input.	Exercise: <i>Psychological safety check:</i> Have team members

	voices. – People hide problems or avoid admitting mistakes until issues explode later. – Little initiative taken beyond assigned tasks; no one asks for help or offers it proactively.	silence. – Past blame or punitive responses to mistakes. – Rapid deadlines or pressure making vulnerability feel risky.	– Low innovation: valuable suggestions and ideas are lost due to fear of speaking out. – Stress & disengagement: individuals feel isolated and anxious, impairing team morale.	– Establish team norms for respectful listening; explicitly <i>reward those who raise risks or new ideas</i>. – Do <i>trust-building exercises</i>: personal introductions, work style “user manuals,” or fun icebreakers.	anonymously write down whether they feel safe admitting mistakes or disagreeing in the team, and why/why not. Discuss results collectively.
Avoiding Conflict & Debate (False harmony / Groupthink)	– The team rarely has lively debates or differing opinions; decisions are either <i>stalled</i> or always unanimous under surface. – Team “meetings” are short and cheerful, but problems resurface repeatedly (issues aren’t truly resolved). – The most senior or dominant person’s opinions always prevail without challenge.	– Fear of tension: desire to keep things pleasant means tough issues are <i>swept under the rug</i>. – Confusing <i>agreement with harmony</i>: teams think lack of conflict = good teamwork. – Power dynamics: deferring to hierarchy or loudest voice to avoid disputes.	– Poor decisions or <i>slow decision-making</i> due to unspoken disagreements or suppressed concerns. – Hidden resentments leading to <i>gossip and passive-aggressive behavior</i>. – Innovation stifled by groupthink (alternative ideas not considered).	– Normalize healthy conflict: emphasize that dissent and debate are signs of engagement, not disloyalty. – Use facilitation techniques in meetings: e.g., designated “devil’s advocate” role, or silent brainstorming before discussion to surface more viewpoints. – Set rules: critique ideas, not people; ensure <i>all voices</i> are heard (ask quieter members for input).	Discussion: “Do we feel comfortable disagreeing with each other? Why or why not? Brainstorm one <i>benefit of healthy debate</i> we’ve seen in past projects.”

Lack of Shared Commitment & Alignment (Goals unclear or externally dictated)	– The team has difficulty making decisions as a group or feels decisions are <i>imposed</i> from above. – Team members are confused about priorities or keep revisiting decisions (lack of clarity). – Low engagement in planning; individuals “go through the motions” without buy-in.	– Top-down directives: Goals, deadlines, or standards handed down without team input. – No clear <i>vision</i> or “<i>why</i>”: team doesn’t understand how their work connects to outcomes. – Frequent changes or shifting priorities eroding confidence in plans.	– Inefficient execution: People working at cross-purposes or on low-value tasks due to misalignment. – <i>Low ownership:</i> Team members feel it’s “not my plan,” reducing motivation to deliver. – Potential for missed targets or churn as focus drifts.	– Clarity and involvement: collaboratively define team objectives and how each person contributes. Summarize decisions at end of meetings for clarity. – Use visual radiators (roadmaps, OKRs) in team space to keep goals visible. – <i>Empower the team</i> to adjust scope or approach when needed (with stakeholder communication), rather than waiting on orders.	Prompt: “Think of a goal or decision where we lacked buy-in. How could we involve everyone next time so we’re all pulling in the same direction?”
Lack of Accountability & Feedback (No peer-to-peer challenge)	– No one addresses underperformance or broken commitments; problems get escalated to managers instead of discussed in team. – A “brilliant jerk” on the team (someone who is technically strong but toxic) is tolerated because of their skills.	– Avoidance of difficult conversations: team not practiced in giving constructive feedback. – Overemphasis on “being nice” or fear of hurting feelings. – Leadership not modeling feedback or addressing toxic behavior.	– Quality & trust degrade: issues persist as errors are not corrected directly, hurting product quality and relations. – Top performers may leave if toxic behavior goes unchecked. – Team stagnation: lack of	– Build a feedback culture: regular peer feedback rounds (paired with positive recognition). – Train team on <i>giving and receiving feedback</i> safely, emphasize feedback’s role in growth. – Set clear expectations that everyone is accountable for team outcomes, and that	Exercise: <i>Peer feedback role-play:</i> In pairs, practice giving respectful, specific feedback on a hypothetical scenario (e.g., late delivery or dominating a meeting). Debrief how it

	– Team members avoid giving feedback to each other, even when it’s needed, or only praise one person consistently.		growth since <i>no one challenges each other to improve.</i>	respectful challenge is welcome.	felt and what was learned.
Over-Engineering & Scope Creep (“Gold-Plating” features)	– Developers frequently propose unrequired improvements beyond scope (e.g., adding extra features or complexity “for future use”). – Team velocity is inconsistent – planned features slip because additional work was done that wasn’t asked. – Tension with Product/PM: Product managers express surprise at features or refinements that weren’t part of agreed scope.	– Desire for perfection: engineers pride themselves on elegant solutions and may add unnecessary polish or complexity. – Miscommunication on requirements or Definition of Done leads devs to assume extra work is needed. – Lack of iterative mindset – trying to deliver “<i>all the bells and whistles</i>” in one go instead of incrementally.	– Delayed delivery and missed deadlines due to expanding scope mid-iteration. – Wasted effort on low-priority features or overbuilt solutions. – Friction between developers and product or other stakeholders because of mismatched expectations.	– Reinforce Definition of Done & scope agreements: ensure everyone understands what “done” includes. – Encourage a “<i>simplest thing that works</i>” philosophy (iterate and refactor later if needed). – Frequent check-ins between devs and product to confirm priorities and avoid surprises.	Prompt: “Recall a time we overbuilt or gold-plated a feature. What caused it? Brainstorm how we can catch ourselves earlier – e.g., more clarity in <i>Definition of Done</i> or better mid-sprint check-ins.”