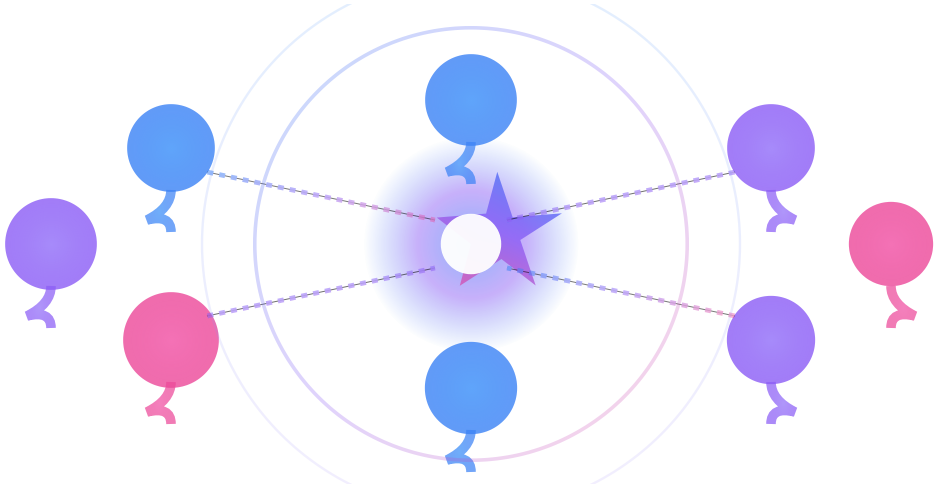
Mission Driven Teams

A practical framework for the age of agentic AI

Build mission-driven, human-centered teams that excel at deciding what to build and why. When AI handles implementation, human ingenuity, empathy, and strategic reasoning become the competitive advantage.

Introduction



I have been building teams and software solutions for about 27 years, working with teams across seven countries, multiple industries, and a variety of organizational cultures and structures. I've served as CTO for more than 15 years, worked with federal agencies, Fortune 500 companies, startups, and everything in between. Throughout this journey, I've observed a pattern that transcends geography, industry, and company size: even technically skilled teams struggle to build products that truly matter when they don't fully understand the overall mission and how their work contributes to it. Have you heard the story about the three bricklayers?

> ### The Three Bricklayers
>
> When asked what they're doing, the first says, "I'm laying bricks." The second says, "I'm building a wall." The third says, "I'm building a cathedral."
>
> The difference isn't in the work they're doing; it's in their understanding of why they're doing it. Teams that see only the bricks and walls (the immediate tasks and technical challenges) miss the cathedral. They miss the mission, the impact, the reason their work matters.

As we begin 2026, we're witnessing a transformation in software development that's both exhilarating and unsettling. Agentic AI systems can now design interfaces, write code, generate tests, and deploy infrastructure, handling much of the technical implementation that once defined our profession. This shift is accelerating rapidly, and it's fundamentally changing what teams need to excel at.

As AI handles more of the "how," the uniquely human capabilities become more valuable than ever. Human ingenuity shines in understanding deeper missions, empathizing with real users, making strategic decisions about what problems are worth solving, and reasoning through complex trade-offs. When leveraged properly, these human capabilities uniquely complement AI and multiply its impact.

In 2026, as agentic AI becomes increasingly capable, building and strengthening these human capabilities isn't just nice to have: it's essential. Teams that can't connect their work to mission, understand user needs deeply, or make strategic decisions about what to build will find themselves outpaced, even with the best AI tools. This is why the Mission-Driven Teams framework matters now more than ever. It provides a structured approach to developing exactly these capabilities: mission alignment, user connection, strategic thinking, and outcome ownership.

I developed this framework through years of working directly with teams in services organizations, consulting firms, and system integrators. I watched what worked and what didn't. I saw teams that understood their mission deliver better outcomes, even when they faced the same constraints as teams that didn't. I experimented with different approaches, learned from failures, and refined practices based on what I observed in complex environments where knowledge was distributed across multiple layers, user access was limited, and the gap between specifications and real needs was often vast.

Over time, these scattered observations and experiments revealed clear patterns. I noticed that the most effective teams shared certain characteristics: every member understood the "why" behind their work, they found ways to connect with users even when direct access was limited, and they felt ownership over outcomes, not just outputs. These patterns became the foundation of what I now call the Mission-Driven Teams framework.

The principles and practices I share here have been proven effective across diverse contexts. They complement and enhance whatever workflow you're already using: Agile, Scrum, Kanban, or any other methodology. They work in startups and enterprises, in healthcare and telecommunications, in government agencies and private sector companies.

While I developed many of these ideas and ran most of the experiments, I learned a great deal from peers and teams across many organizations. Credit is due to everyone who contributed their insights, experiences, and feedback along the way.

> *"I'm sharing it with you not as a finished product, but as a starting point."*
>
> ***Murali***

This is a practitioner's guide, built from real experience. I'm sharing it with you not as a finished product, but as a starting point. Read it, discuss it with your team, and make it your own. Experiment with the principles and activities. Refine them based on what works in your context and what doesn't. And if you're willing, I'd appreciate hearing from you: what's working for your team, what isn't, and what you're learning along the way. Your experiences will help make this guide better for everyone.

Ready to get started? Let's begin.

The Reality of Today's Teams

Walk into any product team meeting, and you might hear conversations like these:

> *"I just need to finish my tickets. QA will catch any issues, and the product manager will tell me if users don't like it."*
>
> Developer, during sprint planning

> *"I designed this feature based on the requirements document, but I'm not sure how it fits into the user's actual workflow. I've never actually seen someone use our product."*
>
> UX Designer, during design review

> *"The contracting officer approved the requirements, but I'm not sure if they've actually talked to the end users. I'm building what was specified, but I worry it might not match what users really need."*
>
> Developer, working on a federal project

> *"I received feedback from the program office, but I'm not sure if that reflects what the actual field agents need. There are so many layers between us and the end users. I feel like we're making*

> *decisions in the dark."*
>
> Product Manager, during stakeholder review

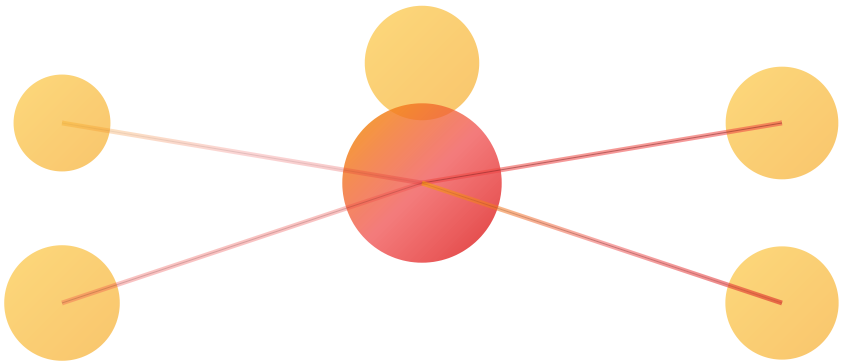
> *"I test what the developers build, but I don't really understand why we're building it or who it's for. I just make sure it matches the specs."*
>
> QA Engineer, during bug triage

> *"We have requirements from three different stakeholder groups, and they don't always align. I'm trying to reconcile them, but I wish I could talk directly to the people who will actually use this system to understand what they really need."*
>
> Business Analyst, during requirements gathering

These quotes reveal a common pattern I've seen time and again: team members are completing their work diligently, yet they're operating without full connection to the mission, the users, and the impact of their efforts. In complex organizational environments (especially in federal agencies and large enterprises), this pattern becomes more pronounced: information flows through multiple layers, each stakeholder makes decisions with the context they have, and access to end users is often limited by organizational structures. The framework I've developed addresses exactly this challenge: how teams can build deeper connections to mission and users, even within these constraints.

Before we explore the solution, let's examine the common patterns that create these disconnects. Understanding these patterns helps teams recognize when they're falling into these traps and provides context for why the principles I'll share next are so critical.

Common Patterns



Common patterns include separate stand-ups per function, isolated work boards for each role, and sequential "hand-offs" between disciplines. Through my work with teams across different organizations and countries, I've observed these same patterns repeatedly. While often well-intentioned, they create significant challenges that undermine team effectiveness. Here's how these patterns manifest and the consequences they produce:

Fragmented Understanding

When individuals focus primarily on their own tasks or user stories, they often develop deep expertise in their domain. However, this narrow focus can lead to a fragmented understanding of the product as a whole. Team members may optimize for their specific function without seeing how their work connects to broader product goals or user needs. The result is features that meet technical specifications but may miss the mark on solving real user problems effectively.

Communication Gaps

When work flows through multiple hand-offs (Researchers → Analysts → Designers → Developers → QA → Operations), information can evolve or be interpreted differently at each stage. Teams working in

separate silos often develop their own communication patterns, terminology, and assumptions. Critical context gets lost as work moves from one function to another. Misunderstandings accumulate, leading to rework, delays, and features that don't match the original intent.

Filtered User Feedback

In many traditional setups, user feedback flows through intermediaries like product managers, business analysts, researchers, designers, or support teams before reaching developers. By the time feedback reaches the team, it's been filtered, summarized, and potentially distorted. The nuances of user struggles, the context of their environment, and the emotional impact of problems get lost in translation. Team members build features based on second-hand information rather than authentic user understanding, leading to solutions that address symptoms rather than root causes.

The Complexity of Modern Organizations

In many environments, particularly when working with federal agencies and large organizations, teams face additional layers of complexity. Multiple stakeholder groups, each with their own priorities, constraints, and perspectives, make decisions that impact the product. Access to end users may be limited due to security protocols, organizational structures, or geographic constraints. In these environments, information flows through many layers, and no single person or team may have the complete picture of how all the pieces fit together.

This is a natural outcome when complex organizational structures meet the need for rapid, user-centered product development. Each stakeholder makes decisions based on their understanding of the mission, their interpretation of requirements, and their view of user needs. Everyone is working with good intentions, yet without mechanisms to share context and build bridges between these perspectives, misalignment accumulates. Decisions made in isolation compound over time, creating products that satisfy individual stakeholder requirements but fail to deliver cohesive user experiences.

Diluted Ownership of Outcomes

When responsibilities are split by role, team members naturally develop deep accountability for their specific domain. However, this can lead to a narrow sense of ownership where individuals feel responsible only for their portion of the work. When a feature fails in production, developers may point to design, designers may point to requirements, and QA may point to incomplete specifications. No one feels fully accountable for the end-to-end user experience or business outcomes. This fragmented ownership means problems go unaddressed, and opportunities for improvement are missed.

Origins of Mission Driven Teams Framework

This framework evolved gradually through my work with teams in services organizations, consulting firms, and system integrators. I kept noticing the same pattern: teams that were technically capable but disconnected from their mission and users struggled to build products that truly mattered. I started paying closer attention to what made some teams effective while others struggled, and began experimenting with different approaches to bridge that gap.

I began experimenting with different approaches to help teams build deeper connections to mission and users. I tried joint planning sessions where all roles collaborated from the start, user shadowing programs that brought developers face-to-face with end users, and outcome-based goal setting that shifted focus from tasks to impact. I also experimented with recording knowledge transfer sessions, organizing trivia quizzes to surface real-world stories and use cases, role-playing exercises, internal domain discussions, and storytelling exercises like asking team members to explain what they do at a dinner table. Some experiments worked. Others didn't. But each attempt revealed something new about what actually helps teams align around mission and build genuine user understanding.

Patterns started to emerge from these experiments. Teams that shared a clear product vision made better decisions autonomously. Teams that engaged directly with users, even when access was limited, built products that truly solved problems. Teams that measured outcomes rather than outputs stayed focused on what mattered. I tested these practices across different contexts: in healthcare systems, telecommunications platforms, supply chain solutions, and educational technology. They worked in startups and in enterprises, across cultures and organizational structures.

These patterns became the foundation of what I now call the Mission-Driven Teams framework. I codified the practices that consistently worked, tested them again in new contexts, and refined them based on real outcomes. The framework helps transform fragmented teams into **mission-driven teams** where every member understands the "why" behind their work, has connection to users (even when direct access is limited), and feels ownership over outcomes, not just outputs. It's designed to work within complex organizational structures, helping teams build shared context and alignment even when information must flow through multiple layers.

> The principles and practices I've outlined here have been proven effective across diverse cultural contexts, organizational structures, and industry domains: from federal agencies to startups, from healthcare to telecommunications. They complement and enhance any workflow

> you're already using (Agile, Scrum, Kanban, etc.), building on what works while introducing new ways to connect your teams to mission and users.

Now that we've examined the common patterns that create disconnects, let's turn to the solution. The framework I've developed consists of six guiding principles that, when implemented together, transform fragmented teams into mission-driven units. Each principle builds on the others, creating a comprehensive approach to building teams that excel at the human work that matters most in the age of AI.

Guiding Principles

At its core, this framework enables mission alignment, user-centric collaboration, and outcome-focused execution. I've organized these guiding principles and practices into six core areas:

Shared Product Vision

A unifying vision understood by all team members, regardless of role

Truly Cross-Functional

Teamwork without intra-team silos, working together as one cohesive unit

Direct User Engagement

Direct engagement with users for empathy and genuine understanding

Outcomes Over Outputs

Focus on impact and results alongside task completion

Continuous Domain Learning

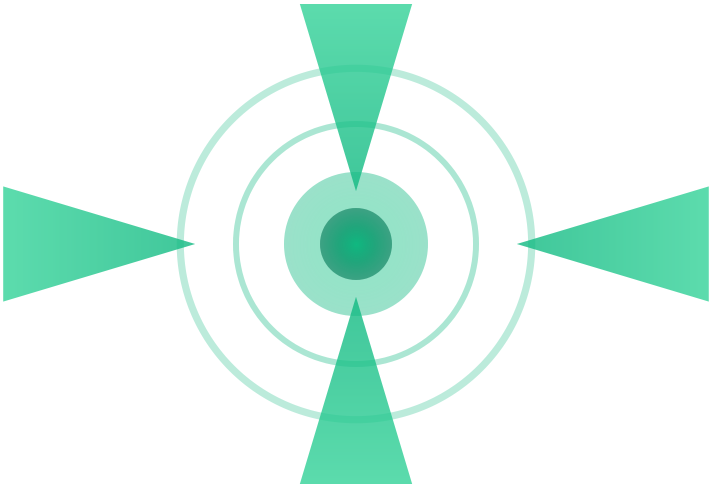
A culture of continuous learning about the domain, users, and ecosystem

The Art of Storytelling

Using stories to clarify, translate, and connect ideas across different audiences

> *"When product teams understand the "why" of what they're building, they can maintain productivity and innovate even in tough times. A team aligned to a compelling mission will "move mountains" to achieve it."*
>
> ***A Product Manager***

1. Shared Mission and Vision



Start by ensuring everyone on your team grasps the big picture: why the product exists, which user needs it addresses, and how it fits into the wider ecosystem. Over the years, I've learned that teams need a clearly communicated, unifying product vision: a "North Star" that every team member can connect their daily work to. Without this shared understanding, even the most technically skilled teams will struggle to build products that truly matter.

In the age of agentic AI, when AI can generate code and implement features rapidly, the critical question shifts from "can we build it?" to "should we build it?" and "why are we building it?" Teams that lack a shared understanding of mission and purpose will find themselves building features that AI can technically implement, but that don't serve real user needs or advance meaningful goals.

I've observed that when teams operate without a clear, shared mission, each role interprets goals differently. Developers optimize for technical elegance, designers focus on aesthetics, product managers chase feature requests, and the result is a product that meets specifications but misses the mark on what users actually need. This fragmentation becomes even more dangerous when AI accelerates implementation: teams can build the wrong things faster than ever before.

A shared mission acts as a North Star that guides every decision, from the strategic to the tactical. When every team member understands why the product exists and who it serves, they can make autonomous

decisions that align with the mission. This becomes essential when AI handles implementation: human team members must excel at the strategic reasoning that determines what gets built, and that reasoning must be grounded in a clear, shared understanding of purpose.

🎯 The Goal

Work toward creating a unifying product vision that every team member, regardless of role, understands and can connect their daily work to. The goal is to help everyone on your team understand **why** they're building what they're building, not just **what** they're building. This alignment enables autonomous decision-making and helps all efforts contribute to the same mission.

❌ Before: Fragmented Understanding

A typical team without shared mission alignment

- Developers focus on completing tickets without understanding user impact
- Designers optimize for aesthetics without business context
- QA tests features in isolation, missing the bigger picture
- Each role has different interpretations of product goals
- Team members are developing their ability to explain how their work serves users
- Decisions require constant escalation to leadership

✅ After: Unified Mission Alignment

A mission-driven team with shared vision

- Every team member can articulate the product's purpose
- Developers understand how their code solves user problems
- Designers make decisions aligned with business outcomes
- QA tests with user context and business goals in mind
- All roles share the same understanding of success
- Team makes autonomous decisions aligned with mission

When all departments and roles align behind a common vision, collaboration deepens and employees discover how their contributions matter to the mission. In teams I've led, this alignment consistently produces stronger autonomous decision-making: team members communicate more effectively and are empowered to focus on true customer value while maintaining their commitment to delivery. This alignment also enables teams to work more independently, because every team member understands what they're trying to achieve and why.

📋 Practical Approaches

Mission Kickoffs

Begin new projects (or sprints) with a brief review of the product's mission, target users, and business goals. For example, discuss real user stories or market facts that highlight the product's purpose. This keeps the "why" front and center.

Visible Artifacts

Create simple artifacts like a product vision statement, mission poster, or a user persona wall. Place these in the team space or digital workspace. They serve as daily reminders of the end goal and user context.

Align Objectives and Key Results (OKRs)

Set team objectives linked to user or business outcomes (e.g., `"Improve checkout success rate by 15%"` alongside `"Build feature X"`). Tie individual goals or Key Results to these outcomes. This way, success is measured by both impact on users/customers and the quality of deliverables.

Frequent Storytelling

Encourage product managers or leaders to share customer stories regularly. A short anecdote in a stand-up about how a real user benefited (or struggled) with the product can powerfully remind specialists (developers, testers, etc.) of the human side of their work.

AI-Assisted Mission Alignment Reviews

Use generative AI to help draft mission statements or product vision documents, but always have the team review and refine them together. The AI can help structure ideas, but human team members must validate that the mission reflects real user needs and business goals. Schedule regular sessions where the team reviews AI-generated proposals against the mission and makes human judgments about what aligns and what doesn't.

🎯 Expected Outcomes

- ✓ **Autonomous Decision-Making:** Team members make better decisions independently because they understand the mission and can evaluate options against shared goals
- ✓ **Increased Engagement:** Employees see how their contributions matter, leading to higher motivation and ownership of outcomes
- ✓ **Better Collaboration:** Shared understanding creates a common language and reduces miscommunication between roles
- ✓ **Customer-Focused Work:** Team prioritizes true customer value over just completing tasks or hitting velocity metrics
- ✓ **Reduced Escalation:** Less need for constant direction from leadership because team understands the "why" behind their work

> *"After shifting to outcome-based goals, we saw a massive increase in team engagement and ownership, as members knew their work directly influenced product success."*
>
> ***A Product Manager***

2. Break Down Silos

Once your team shares a clear mission and vision, the next step is to break down the organizational barriers that prevent them from working together effectively. A shared mission means little if teams still operate in silos, passing work over walls and losing context with each hand-off.

To fully realize cross-functional potential, your team must evolve toward operating as one cohesive unit. This requires integrating workflows and creating direct collaboration paths. I've led teams where all roles (engineering, design, QA, and others) learned to collaborate on the same work simultaneously, swarming around user stories or problems together, building on each function's expertise while working in unity. The transformation is significant, but the results speak for themselves.

As agentic AI handles more implementation work, the value of human collaboration becomes even more critical. When AI can write code, design interfaces, and generate tests, the differentiator isn't in individual technical skills; it's in the collective human intelligence that decides what to build and how different perspectives come together to solve complex problems.

Traditional siloed structures create information bottlenecks and context loss. When work flows sequentially through separate functions, each hand-off loses nuance, and decisions get made in isolation. In an AI-accelerated world, this fragmentation means teams can build features quickly, but they're building based on incomplete understanding. A designer makes assumptions without developer

input, a developer implements without understanding user context, and QA tests without seeing the bigger picture.

Breaking down silos isn't just about efficiency; it's about leveraging the unique human capabilities that AI cannot replicate: empathy, strategic reasoning, and the ability to synthesize diverse perspectives. When teams collaborate simultaneously, they bring together different ways of thinking about problems. A developer's technical perspective combines with a designer's user empathy and a product manager's business understanding to create solutions that are both technically sound and genuinely valuable. This collaborative intelligence becomes the competitive advantage when AI handles the implementation.

🎯 The Goal

Transform from segregated functional teams that hand off work sequentially to a unified, cross-functional team that collaborates simultaneously. The goal is to eliminate the "over-the-wall" effect where work passes through multiple hand-offs, losing context and creating delays. Instead, build a team that swarms around problems together, creates shared ownership, and makes decisions collectively.

❌ Before: Siloed Hand-offs

Traditional functional team structure

- Separate stand-ups: Developers stand-up, Designers stand-up, QA stand-up
- Work flows sequentially: Researchers → Analysts → Designers → Developers → QA → Operations
- Each function has its own Jira board and priorities
- Information gets lost in hand-offs between teams
- Developers are seeing designs earlier in the process
- QA finds issues late, requiring rework
- Team members feel accountable only for 'their part'
- Decisions require coordination across multiple teams

✅ After: Unified Collaboration

Cross-functional feature team structure

- Single unified stand-up with all roles present
- Work happens simultaneously: All team members collaborate from start
- One shared backlog with all functions represented
- Continuous communication prevents information loss
- Developers and designers pair during implementation
- QA involved from design phase, catching issues early
- Collective ownership: team succeeds or fails together
- Feature teams make decisions quickly without external dependencies

Through joint planning and frequent collaboration, teams evolve beyond the "over-the-wall" effect and build shared ownership of outcomes. In the teams I've transformed, members learn to collaborate informally and constantly while maintaining their role expertise. This journey unlocks faster feedback and problem-solving. Feature teams develop the ability to make decisions quickly while staying connected to external groups, and they learn and grow together, building collective responsibility for quality.

📋 Practical Approaches

Unified Backlog and Stand-ups

Maintain one backlog for the whole team that includes tasks from all functions needed to deliver a feature (development, testing, UX, docs, etc.). In daily stand-up meetings, have everyone present (developers, designers, analysts, etc.) discuss progress together. This joint forum creates visibility into

each other's work and surfaces issues early. It helps teams develop shared care for the entire feature, not just individual tickets.

Feature Teams

Organize teams around features or product areas while maintaining specialty expertise. A feature team is a small, self-sufficient unit with all skills needed to go from idea to production for a given slice of product. For example, combine a few frontend devs, backend devs, a QA, and a designer into one squad focused on a feature or user journey, while still maintaining connections to specialty communities for knowledge sharing.

This team works together from design through coding, testing, and release. They can make decisions quickly without waiting on external groups. Importantly, they succeed or fail together, which builds a collective responsibility for quality.

Cross-Functional Pairing

On a day-to-day basis, mix disciplines when tackling work. For instance, a developer can pair with a QA engineer to write unit tests, or a UX designer can sit with a developer while implementing a UI to jointly refine it. This cross-pollination helps teams evolve beyond "my work vs your work" thinking and builds empathy for each other's considerations.

Rotate Meeting Roles

Create opportunities for diverse voices by rotating meeting leadership. Encourage different team members to lead portions of meetings or planning sessions based on their expertise. For example, a developer might demo a feature and also describe the user problem it solves, while a UX designer might contribute insights on technical feasibility. This practice helps teams develop equal voice and respect for each role's input, building a true team mentality.

Communities of Practice

When you have multiple cross-functional teams, you might worry about specialists not interacting across teams. Solve this by forming communities of practice (i.e., guilds or informal groups for each craft, such as engineering, design, etc.) that meet periodically. In these meet-ups, people share lessons, maintain standards, and disseminate best practices so that expertise is spread even though day-to-day teams are mixed.

Cross-Functional AI Code Reviews

When AI generates code, designs, or documentation, ensure cross-functional review. Have designers review AI-generated interfaces for user experience, have product managers review AI-generated features for mission alignment, and have QA review AI-generated tests for coverage. This collaborative

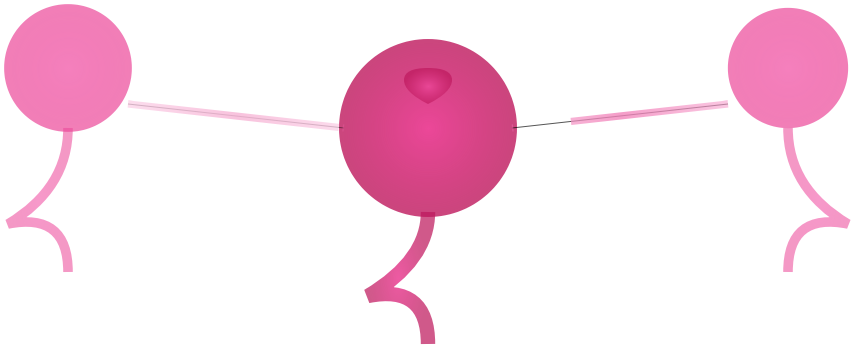
review process ensures that AI-generated work serves the mission and users, not just technical requirements. The human judgment of diverse perspectives catches what AI might miss.

🎯 Expected Outcomes

- ✓ **Faster Delivery:** Reduced hand-offs and parallel work mean features reach users faster with fewer delays
- ✓ **Higher Quality:** Early collaboration catches issues in design phase, reducing costly rework later
- ✓ **Shared Ownership:** Team members feel collectively responsible for outcomes, not just their individual tasks
- ✓ **Better Communication:** Constant informal collaboration replaces formal documentation-heavy hand-offs
- ✓ **Faster Problem-Solving:** Issues are caught and resolved quickly through immediate cross-functional collaboration
- ✓ **Reduced Context Loss:** Information stays within the team throughout the entire development cycle

> Breaking down silos requires intentional effort and leadership support. It may involve reorganizing team structures, changing meeting formats, and encouraging behaviors that feel uncomfortable at first but lead to much faster feedback and problem-solving.

3. User Engagement



With a shared mission and collaborative structure in place, teams are ready for the next critical step: building direct connections with users. This principle transforms abstract "users" into real people with real needs, creating the empathy and understanding that drives better product decisions.

After leading teams for over two decades, I can say with certainty: one of the most powerful ways to build empathy and domain knowledge is to have your team members interact directly with users. When developers, designers, and QA engineers hear from users firsthand (not filtered through intermediaries), they gain authentic insights that fundamentally change how they approach product decisions.

In an era where AI can generate user personas, analyze user data, and even create user journey maps, the one thing AI cannot do is truly empathize with real humans. AI can process information about users, but it cannot feel their frustration, understand their unspoken needs, or connect emotionally with their struggles. This human capacity for empathy and connection becomes the foundation for deciding what problems are worth solving.

I've seen teams build features based on data and assumptions, only to discover that users don't actually need or want what was built. When user feedback flows through intermediaries (product managers, analysts, support teams), critical context gets lost. The emotional nuance of a user's struggle, the

context of their environment, the unspoken needs that don't make it into requirements documents: all of this gets filtered out. Teams end up solving symptoms rather than root causes.

Direct user engagement transforms abstract "users" into real people with real problems. When a developer watches a nurse struggle with a clunky interface during a patient emergency, that experience creates motivation that no requirement document can match. When a designer hears a teacher describe how a feature saves them 30 minutes per day that they can now spend with struggling students, that story becomes the foundation for better design decisions. This human connection is what enables teams to make the strategic decisions about what to build: decisions that AI cannot make because they require genuine understanding of human needs, emotions, and contexts.

🎯 The Goal

Create direct, unfiltered connections between your team members and end users. The goal is to replace second-hand, "watered down" user feedback with authentic, firsthand experiences. When developers, designers, and QA engineers see users' struggles and successes directly, they build genuine empathy and make better product decisions based on real user needs rather than assumptions.

❌ Before: Filtered User Feedback

Traditional indirect user feedback flow

- Developers only hear user feedback through product managers, researchers, designers, or support
- User insights get 'watered down' as they pass through intermediaries
- Team builds features based on assumed requirements, not real needs
- Developers never see how users actually interact with their code
- Support tickets are abstract problems, not real user struggles
- Team operates on guesses without talking to users
- Features technically meet specs but don't solve real user problems
- No personal connection to the people using the product

✅ After: Direct User Engagement

Mission-driven team with direct user connections

- Developers observe users in usability tests and interviews
- Team members shadow users in their natural environment
- Features are built based on firsthand observations of user needs
- Developers see users' reactions to their work in real-time
- Support tickets become learning opportunities with direct user contact
- Team regularly talks to users and understands their world
- Features solve real problems because team understands user context
- Strong personal connection and advocacy for end users

> *"True user empathy comes only from meaningful interaction. Many teams operate on guesses without talking to users."*

I've learned through hard experience that having a single intermediary for product knowledge makes the vision myopic and skews priorities. When team members interact directly with users, they gain authentic insights that can't be captured in documentation or second-hand reports. Watching a user struggle with a feature you built creates powerful motivation to fix it. Hearing a user's excitement about a solution you created reinforces the value of your work. This direct connection transforms abstract "users" into real people with real needs, and that transformation changes everything.

📋 Practical Approaches

User Shadowing and Job Shadowing

Have team members spend time observing users in their natural environment: watching how they actually use the product, understanding their workflows, and seeing the context in which your product fits. This is especially valuable for developers who rarely see the end-user experience. They learn users' routines, pain points, and how your product fits into their daily work.

Usability Testing Observation

Invite developers and QA engineers to observe usability testing sessions. Watching real users struggle with (or succeed with) features they built creates powerful motivation and learning. It's one thing to read a bug report; it's another to see a user's frustration firsthand. This experience often leads to immediate improvements and better design decisions.

Sprint Reviews with Users

Include actual users in sprint reviews or demos. Let them interact with new features and provide immediate feedback. This creates a direct feedback loop and helps the team see their work through users' eyes. Users can ask questions, share their thoughts, and the team can observe their reactions in real-time.

Support Ticket Rotation

Have team members (especially developers) periodically handle customer support tickets or join support calls. This exposes them to real problems users face and the language they use to describe issues. It also builds empathy for support teams and helps developers understand how their code impacts users in production.

User Interviews and Research

Involve engineers and QA in user interviews or research sessions. They can ask technical questions, understand constraints, and see how users think about problems. This helps bridge the gap between technical implementation and user needs, leading to better solutions that work in the real world.

AI-Enhanced User Research with Human Validation

Use AI to analyze user feedback, support tickets, and usage data to identify patterns and themes. However, always validate AI insights through direct user engagement. Have team members review AI-generated user personas and journey maps, then test them against real user interactions. Use AI to surface questions, but have humans ask them directly to users. This approach leverages AI's pattern

recognition while ensuring human empathy and connection remain central to understanding user needs.

🎯 Expected Outcomes

- ✓ **Genuine Empathy:** Team members develop real understanding and care for users because they've met them and seen their struggles
- ✓ **Better Product Decisions:** Features are built on real user insights, not assumptions, leading to solutions that actually solve problems
- ✓ **User Advocacy:** Team members become advocates for users, pushing back on features that don't serve real needs
- ✓ **Faster Problem-Solving:** Direct feedback loops mean issues are identified and fixed quickly, without information loss
- ✓ **Increased Motivation:** Seeing users benefit from their work creates powerful motivation and sense of purpose
- ✓ **Better Communication:** Team uses the same language as users, improving communication with stakeholders and customers

✓ The Result

These interactions build empathy, understanding, and collaboration. Team members become advocates for users because they've seen their struggles and successes firsthand. A developer can draw a line from the code they wrote to a real user's smile, and that is incredibly fulfilling.

4. Outcomes Over Outputs

Direct user engagement gives teams the empathy and understanding they need, but without the right measurement approach, teams can still fall into the trap of building features that don't create real value. This principle shifts how teams measure success, ensuring that the work they do (especially when AI accelerates implementation) actually serves users and advances the mission.

Shift your team's mindset from "did we build what we said we'd build?" to "did we achieve the desired impact?" This fundamental change means measuring success by user behavior, business metrics, and customer satisfaction, not just story completion or velocity. I've seen this shift transform teams from feature factories into outcome-focused organizations.

When AI can generate code and ship features rapidly, the risk isn't that teams will build too slowly; it's that they'll build the wrong things efficiently. I've seen teams celebrate shipping features on time, only to discover that users never adopt them. In an AI-accelerated world, this problem compounds: teams can build more features faster, but if they're measuring success by output (features shipped, story points completed), they'll build many things that don't create real value.

The shift to outcomes over outputs is fundamentally about human judgment and strategic reasoning. AI can measure outputs: lines of code written, features deployed, tests passed. But determining whether those outputs create meaningful outcomes requires human understanding: Does this feature actually solve a user problem? Does it advance the mission? Does it create business value? These are questions

that require empathy, domain knowledge, and strategic thinking, capabilities that define human value in the age of AI.

When teams focus on outcomes, they're forced to engage in the strategic work that matters most: understanding user needs, reasoning through trade-offs, and making decisions about what problems are worth solving. This outcome-focused mindset ensures that when AI accelerates implementation, teams are building things that truly matter. The teams that will thrive are those that excel at the human work of determining what outcomes to pursue, not just the technical work of building features.

🎯 The Goal

Transform how your team measures success from activity-based metrics (stories completed, velocity, features shipped) to impact-based metrics (user behavior changes, business outcomes, customer satisfaction). The goal is to help your team focus on building the **right** things that create real value, not just building things efficiently. Success can be measured by the difference made in users' lives and business results, not just by the volume of work completed.

❌ Before: Output-Focused Metrics

Traditional activity-based measurement

- Success measured by story points completed and velocity
- Team celebrates shipping features, regardless of usage
- Goals focus on deliverables: 'Build feature X by date Y'
- No measurement of whether features solve real problems
- Team hits individual KPIs but misses business objectives
- 60% of built features go unused by users
- False sense of achievement when stories are done
- No learning from what actually works or doesn't work

✅ After: Outcome-Focused Metrics

Impact-based measurement and learning

- Success measured by user behavior changes and business metrics
- Team celebrates features that achieve intended outcomes
- Goals focus on outcomes: 'Improve checkout success rate by 15%'
- Every feature has success metrics defined upfront
- Team aligned on product KPIs that drive business success
- Features built based on validated user needs
- True sense of achievement when outcomes are achieved
- Continuous learning from post-launch reviews and metrics

I've seen teams hit their individual KPIs or complete stories, yet still miss the mark for the business. This creates a false sense of achievement that later turns into confusion and low morale when real outcomes fall short. By shifting to outcome-based goals, your team can focus on product-centric outcomes like feature adoption and customer engagement. This drives higher ownership and motivation because members know their work directly influences product success. Asking "who/why/how" for each feature reveals whether it truly serves users.

📋 Practical Approaches

Outcome-Based User Stories

Write user stories that include the desired outcome, not just the feature. Instead of **"As a user, I want a search bar"**, try **"As a user, I want to quickly find products so I can complete my purchase faster"**. This frames the work around the problem being solved and makes the success criteria clear.

Define Success Metrics Upfront

When building a feature, consider agreeing on how you'll measure success upfront. What user behavior change are you expecting? What business metric might improve? Teams often find it helpful to make these metrics visible and review them regularly. For example, if building a search feature, you might define metrics like "search usage rate," "time to find product," or "conversion rate from search results."

Post-Launch Reviews

After releasing a feature, review whether it achieved the intended outcome. Did users adopt it? Did the metrics improve? What did you learn? This creates a culture of learning and continuous improvement. Schedule these reviews 2-4 weeks after launch to allow time for user adoption.

Celebrate Impact, Not Just Delivery

Recognize and celebrate when features achieve their intended outcomes, not just when they're shipped. This reinforces the outcome-focused mindset. Share success stories in team meetings, highlight metrics improvements, and acknowledge the team's impact on users and business.

Align Individual Goals with Product Outcomes

Consider shifting from activity-based individual goals (e.g., "Complete 20 story points per sprint") to product KPI-based goals (e.g., "Contribute to improving user retention by 10%"). This helps align everyone's work with product success, not just personal productivity.

Measure AI-Generated Work by Outcomes, Not Output

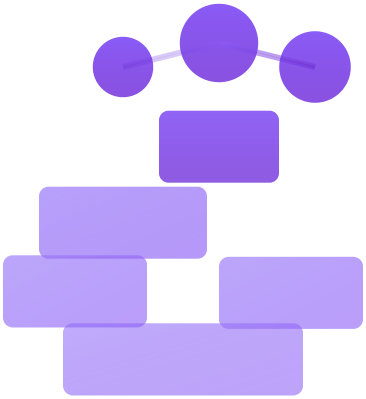
When AI accelerates feature development, resist the temptation to measure success by lines of code generated or features shipped. Instead, measure whether AI-generated features achieve intended user outcomes. Track adoption rates, user satisfaction, and business impact for AI-generated work just as you would for human-generated work. This ensures that AI acceleration serves the mission, not just technical velocity. Set outcome metrics before AI generates code, and review them after launch.

🎯 Expected Outcomes

- ✓ **Higher Feature Adoption:** Features are built to solve real problems, leading to actual user adoption and engagement
- ✓ **Business Impact:** Team's work directly contributes to business metrics and objectives, not just activity
- ✓ **Increased Ownership:** Team members feel accountable for outcomes, not just completing tasks
- ✓ **Continuous Learning:** Post-launch reviews create feedback loops that improve future decisions
- ✓ **Better Prioritization:** Focus on outcomes helps identify which features truly matter and which don't
- ✓ **Reduced Waste:** Less time building unused features, more time on work that creates real value

> *"After adopting a product mindset (asking who/why/how for each feature), we discovered that 60% of built features weren't being used by users. Switching from activity-based individual goals to product KPI-based goals drove higher ownership and motivation."*
>
> ***A Product Manager***

5. Domain Knowledge



Focusing on outcomes ensures teams build the right things, but to make truly informed decisions about what problems are worth solving, teams need deep understanding of the domain. This principle addresses the contextual knowledge that enables strategic reasoning: the human capability that determines what gets built when AI handles implementation.

Domain knowledge (understanding the industry, users' workflows, business context, and ecosystem) is crucial for building the right product. After leading teams across multiple industries, I've consistently seen that teams with deep domain understanding make better decisions, communicate more effectively with stakeholders, and build features that truly solve problems rather than just meeting specifications.

AI can learn from vast amounts of data and generate code based on patterns, but it cannot develop the deep, contextual understanding that comes from immersing oneself in a domain. Domain knowledge isn't just information; it's the nuanced understanding of how things work in practice, the relationships between different parts of an ecosystem, the unspoken constraints and opportunities that shape what's possible. This understanding is essential for making strategic decisions about what to build.

I've seen technically skilled teams build features that meet specifications but fail in practice because they didn't understand the domain context. A healthcare feature that doesn't account for clinical workflows, a financial tool that ignores regulatory constraints, an educational platform that doesn't understand classroom realities: these failures happen when teams focus on technical implementation

without deep domain understanding. When AI handles implementation, this risk increases: teams can build technically correct solutions that are contextually wrong.

Domain knowledge enables teams to make informed decisions about what problems are worth solving and how to solve them in ways that fit the real-world context. It's the difference between building a feature and building a solution. When teams understand the domain deeply, they can reason through trade-offs, anticipate edge cases, and design solutions that work in practice, not just in theory. This domain expertise becomes the foundation for the strategic reasoning that determines what gets built: the human capability that complements and amplifies AI's implementation capabilities.

🎯 The Goal

Build deep, continuous understanding of the domain: the industry, users' workflows, business context, and ecosystem, so your team can make informed decisions and build products that truly fit users' needs. The goal is to help your team members develop domain expertise alongside their technical expertise, understanding not just **how** to build, but **what** to build and **why** it matters in the broader context. This helps teams maintain a holistic view of the ecosystem and avoid "missing the forest for the trees."

❌ Before: Surface-Level Understanding

Team with limited domain knowledge

- Team understands technical implementation but not business context
- Features built without understanding industry regulations or constraints
- Communication gaps with stakeholders due to domain language barriers
- Team focuses narrowly on technical tasks, missing ecosystem context
- No understanding of how product fits into users' broader workflows
- Decisions made in isolation without considering upstream/downstream impacts
- Team relies entirely on product managers for domain knowledge
- New team members take months to understand the domain

✅ After: Deep Domain Expertise

Mission-driven team with continuous learning

- Team combines technical expertise with deep domain knowledge
- Features built with full understanding of industry context and constraints
- Effective communication with stakeholders using shared domain language
- Team maintains holistic view of ecosystem and business context
- Deep understanding of how product fits into users' complete workflows
- Decisions consider system-level impacts and ecosystem relationships
- Domain knowledge distributed across team, not siloed in one person
- New team members onboard quickly through knowledge base and mentoring

Intentionally connecting people across traditional boundaries allows them to "see the big picture" and integrate diverse expertise to generate innovative solutions. In my experience leading teams, when employees understand that learning about the user's world is a core part of their job (and their career growth), they actively seek that knowledge. Over time, team members become true domain experts and strong technical contributors, a powerful combination for building the right product. By making learning practices part of regular work, teams remain closely tied to the ecosystem they operate in and avoid the trap of "missing the forest for the trees."

📋 Practical Approaches

Domain Immersion Activities

Organize activities that expose the team to the domain: attend industry conferences, visit customer sites, read industry publications, or invite domain experts to speak to the team. The goal is to understand the world your users operate in. For example, if building healthcare software, have team members observe clinical workflows or attend medical conferences.

Ecosystem Mapping

Especially when a product is part of a larger ecosystem of services, it's helpful to visualize and discuss that big picture. The team can collaboratively map out how your product integrates with others, what upstream/downstream systems exist, and where the data flows. This can illuminate how a change in one component (maybe owned by another team) could affect your users, and vice versa. Cross-team workshops can be useful here if multiple teams own different pieces of the puzzle.

Service Design: Front Stage and Back Stage

Use service design techniques to visualize the front stage (what users and stakeholders experience) and the back stage (systems, services, and teams that make it work). Create journey maps and service maps that show the full ecosystem, highlighting how much of the team's effort goes into back stage systems that support the front stage experience. These visuals help teams see dependencies, handoffs, and opportunities to improve the end-to-end service.

Documentation and Wikis

Consider maintaining an easily accessible knowledge base with key domain information: user personas, common use cases, industry regulations, glossaries of domain terms, etc. Encourage team members to contribute to it whenever they learn something new from the field. For instance, when someone returns from a customer visit or a conference, they might write a short internal blog or wiki update on insights learned. This becomes a living resource that reinforces learning.

Communities and Bridging Ties

Encourage forming "bridging" relationships across departments (e.g., pairing an engineer with someone in customer support or operations for knowledge exchange). In practice, this might mean a mentorship program or buddy system where, say, a developer regularly chats with a support agent or a marketing specialist to learn how the product is perceived externally. These connections broaden each person's perspective beyond their usual circle.

Learning Culture

Cultivate a culture where continuous learning is valued. In practice, team leads often find it helpful to allocate some time for domain learning activities (rather than considering them a distraction from "real work"). This could be as simple as a "10% time" policy to learn about the customer or domain, or

scheduling an hour a week for the team to discuss a user article or support ticket trends. When employees see that understanding the user's world is a core part of their job, they tend to seek that knowledge more naturally.

AI as a Domain Learning Tool, Not a Replacement

Use generative AI to help teams learn about domains: summarize industry reports, explain domain terminology, or generate questions for domain experts. However, treat AI as a starting point, not the destination. Have team members validate AI-generated domain knowledge through direct engagement with users, domain experts, and real-world observation. Use AI to accelerate learning, but ensure human team members develop deep, contextual understanding through experience. The goal is domain expertise, not just domain information.

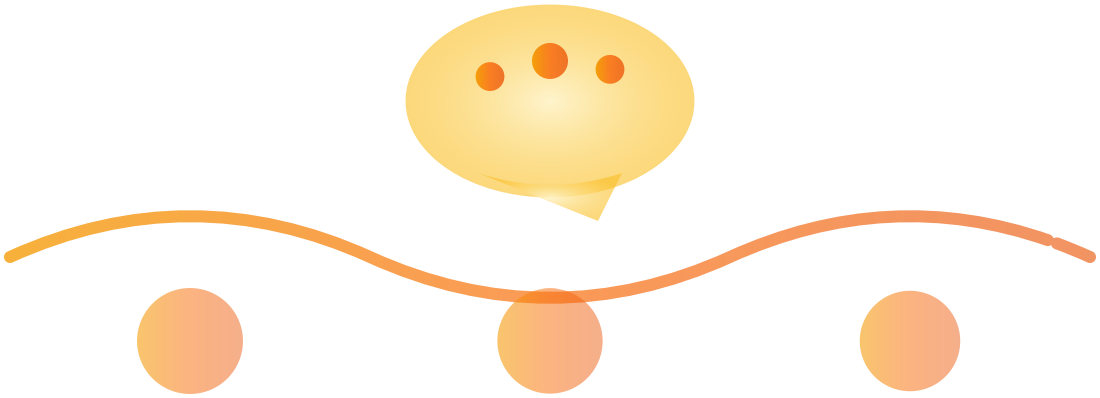
🎯 Expected Outcomes

- ✓ **Better Product Decisions:** Deep domain knowledge enables teams to make informed decisions that truly fit users' needs and industry context
- ✓ **Effective Communication:** Team can communicate effectively with stakeholders using shared domain language and understanding
- ✓ **System-Level Thinking:** Team understands how their product fits into the broader ecosystem and considers upstream/downstream impacts
- ✓ **Innovation:** Integration of diverse expertise across boundaries generates innovative solutions
- ✓ **Faster Onboarding:** Knowledge base and learning culture help new team members understand the domain quickly
- ✓ **Holistic Perspective:** Team maintains awareness of the "forest" (ecosystem) while working on the "trees" (individual features)

> 💡 **The Long-Term Benefit**
>
> When teams make these learning practices part of their regular work, they tend to remain closely tied to the ecosystem they operate in. They develop the ability to see both the forest and the trees, regularly zooming out to refresh their view of the bigger picture. This continuous

> alignment with domain realities helps ensure that even as the team scales or new members join, the collective understanding of the team stays oriented around the mission and user context.

6. The Art of Storytelling



Deep domain knowledge enables teams to make informed decisions, but those decisions only create impact when teams can communicate them effectively. The final principle addresses the human skill of storytelling: the bridge that connects technical work to human impact and helps teams articulate why their work matters.

Storytelling is one of the most powerful tools for building understanding, connection, and purpose in teams. In my experience leading teams across cultures and contexts, those that master storytelling can clarify complex ideas, translate technical concepts for different audiences, connect emotionally with their work, and build genuine pride in how their contributions impact real-world scenarios. When team members can explain what they do at a dinner table conversation, they've achieved a level of clarity and connection that fundamentally transforms how they see their work.

AI can generate documentation, create presentations, and even write explanations, but it cannot tell stories that connect emotionally or build genuine understanding across diverse audiences. Storytelling is fundamentally human: it requires empathy to understand what resonates with different listeners, creativity to find the right narrative structure, and emotional intelligence to connect technical work to human impact. In an age where AI handles implementation, storytelling becomes the bridge that helps teams communicate the "why" behind what gets built.

I've observed that teams struggle when they can't articulate why their work matters. Technical teams speak in jargon that excludes stakeholders. Business teams describe features without understanding technical constraints. Users describe problems that don't translate into clear requirements. This communication breakdown becomes more problematic when AI accelerates development: teams can build features quickly, but if they can't explain why those features matter or how they connect to user needs, they'll build things that don't create value.

Storytelling transforms abstract work into meaningful narratives. When a developer can explain how their code helps a nurse save time during patient care, that story creates purpose and pride. When a designer can articulate how their interface helps a teacher manage their classroom more effectively, that narrative connects technical work to real-world impact. This ability to tell stories helps teams make better decisions about what to build because they can communicate and reason through the human impact of their choices. In the age of AI, storytelling becomes essential for the human work of strategic reasoning and decision-making.

🎯 The Goal

Develop your team's ability to tell compelling stories that clarify ideas, translate concepts across different audiences, connect emotionally, and build a sense of purpose and pride. The goal is to help every team member become an effective storyteller who can explain their work in ways that resonate with technical peers, business stakeholders, end users, and even family members at a dinner table. This skill helps teams bridge communication gaps, build empathy, and maintain connection to the real-world impact of their work.

❌ Before: Technical Jargon and Disconnection

Team struggles to communicate impact and purpose

- Team members use technical jargon that excludes non-technical stakeholders
- Developers describe work in terms of code and features, not user impact
- Stakeholders don't understand how technical work connects to business goals
- Team members struggle to explain their work to family or friends
- No emotional connection between daily tasks and real-world outcomes
- Team members see their work as isolated tasks, not part of a larger story
- Miscommunication between technical and business teams due to language barriers
- Lack of pride or purpose because impact feels abstract and distant

✅ After: Compelling Stories and Purpose

Mission-driven team with storytelling skills

- Team members adapt their communication style to their audience
- Developers can explain their work in terms of user problems solved
- Stakeholders understand how technical work drives business outcomes
- Team members can explain their work clearly at a dinner table conversation
- Strong emotional connection between daily work and real-world impact
- Team members see their work as part of a meaningful narrative
- Effective translation of ideas between technical and business contexts
- Genuine pride and purpose from understanding real-world contributions

Through my work with teams across different countries and contexts, I've observed that storytelling is often the missing piece that transforms good teams into great ones. When team members can tell the story of their work, they're not just describing what they built; they're connecting it to why it matters. A developer who can explain how their code helps a nurse save time during patient care isn't just writing software; they're part of a story about improving healthcare. A designer who can articulate how their interface helps a teacher manage their classroom more effectively isn't just creating layouts; they're contributing to education outcomes. This ability to see and tell the story transforms how team members approach their work, building deeper engagement and a sense of purpose that goes far beyond completing tasks.

📋 Practical Approaches

The Dinner Table Test

Encourage team members to practice explaining their work as if they're talking to family or friends at a dinner table. This exercise forces them to strip away technical jargon and focus on the human impact. For example, instead of "I'm implementing a microservices architecture," they might say "I'm building a system that helps doctors access patient information faster, so they can spend more time with patients." This practice helps team members develop the skill of translating technical work into relatable stories.

User Story Narratives

Transform user stories from simple requirements into compelling narratives. Instead of "As a user, I want to filter results," develop the full story: "Sarah, a busy project manager, spends 15 minutes every morning searching through hundreds of tasks. She's frustrated because she can't quickly find what she needs. When we add filtering, she'll save 10 minutes daily, which adds up to over 40 hours per year. That's time she can spend on strategic planning instead of administrative tasks." These narratives help team members understand not just what they're building, but why it matters.

Impact Storytelling Sessions

Regularly share stories about real-world impact. In stand-ups or retrospectives, have team members share anecdotes about how their work affected users. For example, "Last week, a teacher told us that our new feature helped her save 30 minutes per day, which she now uses to provide one-on-one help to struggling students." These sessions help build emotional connection and pride in the team's contributions. Record these stories and create a library that team members can reference when they need to remember why their work matters.

Audience-Specific Translation

Practice translating the same idea for different audiences. For a technical feature, help team members develop three versions: one for technical peers (focusing on architecture and implementation), one for business stakeholders (focusing on business value and outcomes), and one for end users (focusing on how it solves their problems). This skill helps teams communicate effectively across organizational boundaries and ensures everyone understands the value of the work.

Story-Based Documentation

Create documentation that tells stories rather than just listing features. Instead of "Feature X allows users to Y," write "When Maria, a healthcare administrator, needs to process patient records, she used to spend hours manually entering data. With Feature X, she can complete the same task in minutes, reducing errors and allowing her to focus on patient care coordination." This approach helps new team members understand not just what the product does, but why it exists and who it serves.

Before and After Narratives

When introducing new features or improvements, tell the story of the "before" state (the problem users faced) and the "after" state (how their lives are better). For example, "Before: Teachers spent their lunch breaks manually calculating grades, often making errors that affected student records. After: Teachers can generate accurate grade reports in seconds, giving them their lunch breaks back and ensuring students receive fair, accurate evaluations." These narratives help team members see the transformation their work enables.

Storytelling Workshops

Conduct workshops where team members practice storytelling techniques. Cover elements like setting the context, introducing the characters (users), describing the challenge, showing the journey (development process), and revealing the outcome (impact). These workshops help team members develop storytelling as a skill, not just an occasional activity. Role-playing exercises where team members practice explaining their work to different audiences can be particularly effective.

Human Stories Over AI-Generated Content

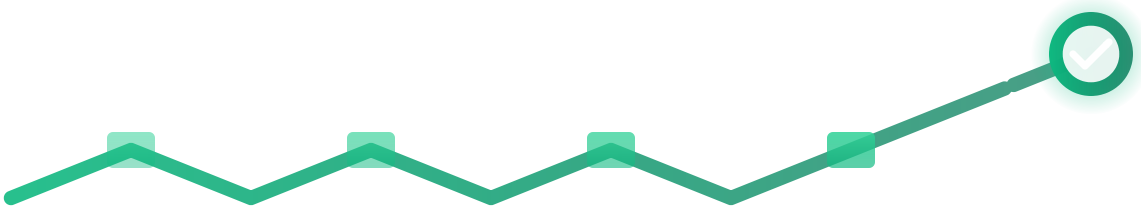
While AI can generate documentation and explanations, prioritize human storytelling that connects work to real user impact. Use AI to help structure stories or draft initial versions, but have team members refine them with real anecdotes, emotions, and personal observations from user interactions. When presenting work to stakeholders, lead with human stories about real users, then use AI-generated summaries as supporting material. This ensures that the emotional connection and human understanding remain central, even when AI assists with communication.

🎯 Expected Outcomes

- ✓ **Clear Communication:** Team members can explain complex ideas in ways that resonate with different audiences, reducing miscommunication and building shared understanding
- ✓ **Emotional Connection:** Stories create emotional connections between team members and their work, leading to deeper engagement and motivation
- ✓ **Sense of Purpose:** Team members develop a clear understanding of how their work contributes to real-world scenarios, building genuine purpose and pride
- ✓ **Better Stakeholder Alignment:** Effective storytelling helps stakeholders understand technical work and its business value, leading to better support and resources
- ✓ **Cross-Functional Understanding:** Stories bridge communication gaps between technical and non-technical team members, fostering better collaboration
- ✓ **User Empathy:** Storytelling helps team members see users as real people with real challenges, not just abstract personas or requirements
- ✓ **Pride in Contribution:** When team members can articulate how their work impacts real people, they develop genuine pride in their contributions and ownership of outcomes

Storytelling is more than a communication technique, it's a way of thinking about work that connects the technical to the human, the abstract to the concrete, and the task to the impact. When team members can tell the story of their work, they're not just describing what they do, they're articulating why it matters. This ability transforms how teams see themselves, their work, and their place in the larger narrative of solving real problems for real people. In my experience, teams that master storytelling don't just build better products, they build products with purpose, and they do so with a sense of pride and connection that makes the work meaningful beyond the daily tasks.

Next Steps



We've explored the six guiding principles that form the foundation of mission-driven teams. These principles work together as an integrated system: shared mission provides direction, breaking down silos enables collaboration, user engagement builds empathy, focusing on outcomes ensures value, domain knowledge enables informed decisions, and storytelling connects work to impact. Together, they create teams that excel at the human work that matters most in the age of AI.

Transforming your cross-functional teams into mission-driven units involves a blend of process evolution (how work is organized and delivered) and cultural growth (how people think and interact). The framework I've outlined here can be thought of as **mission-driven product development**: a journey of continuous learning and alignment.

Its core idea is simple: **everyone on your team, regardless of role, can benefit from first internalizing the mission, the user, and the domain context, and then bringing their expertise to bear to fulfill that mission**.

Role-Specific Workbooks

To make this framework actionable for every role, use the tailored workbooks. Each one contains role-specific activities and evidence so your teammates know exactly what to do next without guessing.

Start with the library: Workbook Library

- Generic Workbook (Any Role)
- Software Developer
- QA Engineer
- DevOps Engineer
- Architect
- Security Engineer
- Scrum Master
- Product Manager
- Product Analyst
- Project Manager
- HCD Researcher
- HCD Designer
- Policy Analyst
- Data Analyst
- Data Engineer
- Customer Support

> 💡 **Keep It Simple**
>
> This approach is deliberately kept generic and lightweight. It doesn't prescribe a strict methodology or heavy processes. You can implement these principles within Scrum, Kanban, or any project management style, or even alongside frameworks like SAFe or Spotify model, because they are fundamentally about mindset and communication.

The emphasis on keeping it simple means you start with small changes: maybe a joint stand-up here, a customer shadowing day there, an extra question in your user story template, and gradually build a

culture that aligns with these values. **Any team, whether a five-person startup squad or a 500-person multi-team enterprise, can adapt these ideas.**

The Benefits

- ✓ Teams deliver solutions that hit the mark more often, because they are truly tuned in to what users need
- ✓ They communicate and collaborate better because they have a common language of customer success
- ✓ Members find greater meaning and motivation in their work: a developer can draw a line from the code they wrote to a real user's smile

> *"Good developers don't need requirements: they connect to customers and own outcomes over following strict requirements."*

By implementing this framework, you'll eliminate the "forest for the trees" problem. No longer will a UI designer just push pixels, or a QA tester just log bugs, or a developer just commit code. Instead, each will be an advocate for the end-user and the product's purpose.

I've seen cross-functional teams operating in this cohesive, user-centered way become far more than the sum of their parts. They evolve into mission-driven units that consistently build products which resonate with users and succeed in the ecosystem. That is the ultimate goal: a team that not only does its work, but one that deeply understands the work's context and why it matters.

About the Author

Murali Mallina is an accomplished technology leader with over 27 years of experience in developing transformative software solutions and cultivating high-performing teams. As both a builder and leader, he has built multiple teams and software solutions and has served as Chief Technology Officer (CTO) for more than 15 years, driving innovation across supply chain management, edTech, workforce, telecommunications, and healthcare sectors. He has been working with federal agencies for the last 9 years.

With a robust expertise in team building and establishing Centers of Excellence (CoEs), Murali has implemented frameworks that enhance organizational capabilities, promoting innovation and operational excellence across various domains. A dedicated educator and mentor, he founded Teaching for Good, which inspires learners through computer science clubs and comprehensive training programs.

Recognized for his thought leadership, Murali frequently speaks at industry conferences and meetups, sharing valuable insights on AI/ML, DevSecOps, agility, and scalable architectures. An award-winning academic, he continues to merge innovation with purpose, creating impactful solutions for both businesses and communities.

This framework is born from practical experience. Throughout his career, Murali has spent significant time working with services organizations, consulting firms, and system integrators, often in complex environments with multiple stakeholders, limited direct user access, and intricate organizational structures.

The strategies and principles outlined here aren't theoretical concepts. They're practical approaches that have worked in the trenches: observations and solutions developed firsthand while navigating the challenges of siloed teams, fragmented communication, and the disconnect between technical execution and mission outcomes. This framework reflects what has worked when building mission-driven teams in the real world.

While Murali came up with many of these ideas and ran most of the experiments, he learned a great deal from his peers and teams across many organizations. Credit is also due to everyone who contributed their insights, experiences, and feedback along the way.