A MANAGEMENT MANIFESTO

The Human-Agent Organization

For teams made of flesh and code.

Most companies are still treating AI like software.

A person opens a tool, writes a prompt, waits for an answer, and copies the result somewhere else. That is useful, but it is still the old model. It is still a human using a tool.

The larger shift begins when AI stops being something employees use and becomes something employees work with.

But the foundation comes before the coworker. A company has to become readable to machines before digital coworkers can do serious work inside it. Better structured data improves today's AI tools and prepares the ground for tomorrow's digital employees.

This is not as strange as it first sounds. We have already lived through the rehearsal. Remote work taught companies how to coordinate people who were not in the same room, the same

building, or even the same country. We learned how to work through Slack, Teams, documents, tickets, dashboards, pull requests, calendars, and shared files. We learned how to delegate asynchronously. We learned how to give people access to systems they needed and judge them by the work they delivered, not by whether we could see them sitting at a desk.

The next step is simple to state and hard to absorb:

> *We need to let digital coworkers into that same operating system.*

Not as side panels. Not as prompt boxes. Not as clever autocomplete. As participants in the organization.

The future company is not a company where humans use AI. It is a company made of humans and agents. Flesh and code. People and digital coworkers working in the same channels, on the same tasks, with clear roles, shared memory, and a common language for getting work done.

This manifesto is for management because this is now a management problem. It is not only a technology problem. The companies that win will not simply buy better AI tools. They will make their data, documents, systems, and teams ready for digital coworkers.

They will learn how to hire them, brief them, place them in teams, give them mandates, measure their output, shape their context, and let them learn.

That is the Human-Agent Organization.

01 The Foundation: Structure Data For Flesh And Code

Before roles, before heartbeats, before org charts, there is a more basic question:

Can the company be read by machines?

This is the foundation. A company can make itself better at using AI as a tool, and more ready to hire digital coworkers, by improving the structure of its underlying data.

If the company's knowledge lives in messy documents, ambiguous spreadsheets, screenshots, slide decks, and chat threads with no durable summary, then every AI system has to spend its first effort trying to understand the mess. A prompted tool has to parse it. A digital coworker has to parse it. A search system has to parse it. A reporting system has to parse it.

The better move is to make the source of truth clean.

Hadley Wickham's tidy data principle came from data analysis in R, but it has a much broader lesson for the Human-Agent Organization. Tidy data means data is structured in a predictable way. Each variable has its place. Each observation has its row. Each type of thing has its table.

The deeper management lesson is this:

> *Structure the source of truth for machines, then generate the human presentation from it.*

Most companies do the opposite. They store knowledge in formats that look good to humans but are awkward for machines:

- PDFs that are really digital paper
- Word documents with implicit structure
- spreadsheets where color means status
- merged cells that hide meaning
- slide decks where the layout carries the logic
- screenshots of dashboards
- long chat threads with no durable summary
- documents with no owner, status, or version

Humans can often interpret this because we are good at reading messy context. But AI systems work better when meaning is explicit.

This does not mean humans should read ugly machine files. It means the source should be clean, and the presentation should be generated on top.

The future document is both human-readable and machine-readable.

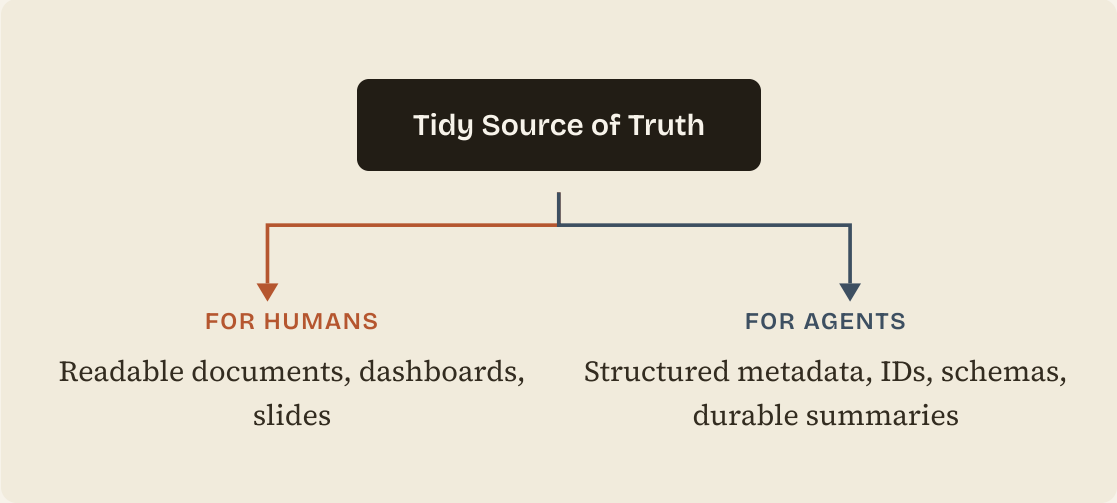
Markdown with frontmatter is a simple example:

```
---
title: Q3 Pricing Experiment
status: in_review
owner_human: Maria
owner_agent: Pricing Analyst Agent
related_systems:
  - Stripe
  - CRM
  - analytics warehouse
decision_needed_by: 2026-07-15
---

# Q3 Pricing Experiment

This document describes the proposed pricing
experiment for the enterprise segment...
```

The human reads the document. The agent reads the metadata and understands status, ownership, systems, dates, and responsibility.



The same principle applies everywhere:

- Use structured changelogs instead of vague update notes.
- Use clean CSV, Parquet, or database tables instead of visual spreadsheet tricks.
- Use frontmatter for document status, owners, and review dates.
- Use conventional commit patterns so code history is readable.
- Use schemas for recurring decisions.
- Use IDs and links so agents can trace work across systems.
- Keep raw data separate from charts.
- Generate slide decks and dashboards from clean sources when possible.

This is what "for flesh and code" means.

Humans still get readable documents, useful dashboards, good slides, and polished presentations. But underneath, the company's knowledge is structured so digital coworkers can read it, search it, update it, and act on it.

Do not make agents parse visual clutter if you can give them clean structure.

Tidy data becomes tidy context.
Tidy context becomes better digital work.

That is why data structure comes before agent structure. The cleaner the company's underlying knowledge, the more useful every AI tool becomes. And when the company is ready to move from tools to coworkers, the digital coworkers arrive into an environment they can actually understand.

02 The Shift: From Tool Use To Digital Coworkers

Prompting is not management.

Prompting is asking a system to do one thing right now. Delegation is giving a worker a role, a goal, a context, and enough authority to make progress without being micromanaged.

Most AI work today is still prompting. A human opens Claude, ChatGPT, Cursor, Codex, or another tool and says: write this, summarize that, fix this function, review this text. The system responds. The human copies the output into the real workflow.

That pattern creates value, but it keeps AI trapped inside the old software mindset. The AI is treated like a calculator. It waits. It answers. It disappears.

A digital coworker behaves differently.

A digital coworker can be assigned a standing role. It can watch a channel, inspect a queue, check a repository, read new documents, monitor a dashboard, prepare a daily summary, or follow up on open tasks between human messages. It does not need every action to begin with a fresh prompt. It works from a mandate.

Managers already understand this difference from human teams.

You do not manage a remote colleague by sending them one tiny instruction every five minutes. You give them a job description, a project, a set of expectations, access to the right systems, and a way to report progress. You trust them to work between meetings. You expect them to come back with questions, drafts, decisions, and finished work.

Agents need the same shift.

The management question is not:
"What can I prompt the AI to do?"

The better question is:
"What work should this digital coworker own?"

That question changes everything. It moves AI from the interface layer into the organization layer.

The point is not to make AI sound more human than it is. The point is to stop making digital labor invisible. If an agent contributes to work, it should have a place in the work system. It should have a role, a name, a mandate, and a way to improve.

One real role, not a script with a friendly name.

That distinction matters. A scheduled task can produce an output. A digital coworker can own a domain, build judgment, collaborate, remember corrections, and become part of how the organization operates.

03 The Precedent: Remote Work Built The Operating System

Remote work already taught us how to work with people we rarely see.

In a large international company, your closest colleagues might sit in Denmark, India, Germany, the United States, Poland, and Brazil. You may never share an office. You may not share a time zone. Yet the work still happens because the company has a digital operating system:

- Teams and Slack for conversation

- Email for formal communication
- Jira, Linear, Asana, or Trello for work tracking
- GitHub or GitLab for code
- Notion, Confluence, SharePoint, or Google Docs for knowledge
- Dashboards for status
- Calendars for coordination
- Pull requests for review
- Tickets for accountability

That is already a workplace for people without bodies in the room.

So the leap to digital coworkers is smaller than it sounds. We do not need to invent an entirely new way of working. We need to admit that the digital workplace can contain more than humans.

An agent can have a profile. It can be tagged in a thread. It can receive a direct message. It can sit in a project channel. It can read a ticket. It can update a document. It can open a pull request. It can summarize a decision. It can report what it did while the humans were asleep.

The remote-work model gives us the pattern:

- A worker has an identity.
- A worker has a role.
- A worker has access to the systems needed for the role.
- A worker communicates asynchronously.
- A worker participates in shared rituals.

- A worker learns the norms of the organization.
- A worker escalates when the task moves outside the mandate.

The digital coworker fits directly into this pattern.

The mistake is to treat agents as something separate from the organization. The better approach is to place them inside the same coordination fabric we already built for distributed human work.

This also means the company should stop thinking only in user interfaces. A user interface is what a human clicks. A work environment is where a participant operates. Remote work taught us to build environments: channels, files, tasks, dashboards, code repositories, decision logs, and shared rituals. Agents should enter that environment, not sit outside it waiting for a human to paste work into a prompt box.

The future of AI adoption will not be "everyone gets a chatbot."

The future is that every team gets digital teammates.

OpenClaw and Hermes helped make this pattern concrete. Not because every company must use those exact systems, but because they show the shape of the new stack: an agent needs a workplace with loops, tools, channels, memory, and files, and it needs a mind suited for instruction following, tool use, structured reasoning, and multi-turn work.

Agents need an operating environment and a mind suited for action.

That is no longer theoretical.

04 The Digital Coworker: First-Class Citizenship

A first-class citizen is not an add-on. It is a real participant in the system.

Today many agents borrow human accounts, share keys, live inside one person's laptop, or operate as invisible scripts. That keeps them outside the organization. The work may be useful, but the agent has no real place.

A Human-Agent Organization gives agents their own place.

That starts with identity.

An agent should have a name, a profile, a role, an owner, and a clear place in the digital workplace. If it comments on a pull request, the comment should come from the agent. If it updates a ticket, the update should show the agent. If it prepares a report, the report should say which agent produced it and under which mandate.

This is not about making the agent cute or pretending it is human. It is about making work legible.

When a human colleague does something, the organization knows who did it. The same should be true for code. A digital coworker should not be an anonymous force moving through systems. It should be a named participant whose work can be read, discussed, improved, and reused.

First-class citizenship also means every important internal system should have an agent-facing surface. Humans need screens, buttons, slides, and dashboards. Agents need structured interfaces, readable events, stable IDs, clean documents, searchable history, and tools they can call.

The goal is not to make agents pretend to use software exactly like humans. The goal is to let flesh and code operate in the same semantic workplace.

The basic pattern is simple:

```
agent:
  name: Docs Steward
  role: Documentation maintenance
  human_owner: Head of Engineering
  channels:
    - "#engineering"
    - "#product-releases"
  systems:
    - docs repository
    - product changelog
    - API schema
  mandate:
    - detect stale documentation
    - draft updates
    - ask product owner for missing context
    - open pull requests for review
```

This is plain management. It is the same logic as a job description, a reporting line, and a systems access request.

Once agents become first-class citizens, they stop being random tools and become part of the operating model.

They can be onboarded.

They can be assigned.

They can be measured.

They can learn.

They can be moved between teams.

They can be given narrower or broader mandates.

They can specialize.

That is the beginning of real organizational design for AI.

05 The Agent Contract

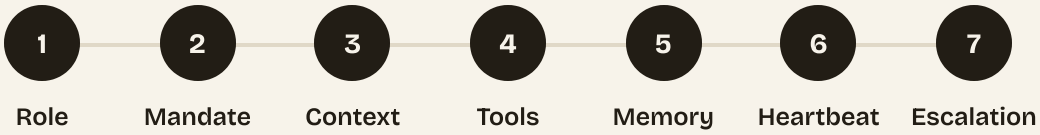
An agent is not just a model.

An agent is a contract.

The contract tells the agent what it is, what it does, what context it should care about, what systems it can use, how often it should check in, how it should communicate, what it should remember, and when it should involve a human.

For management, this is the central object. Do not start by asking which model to use. Start by defining the contract.

A useful agent contract has seven parts:



Role

The role gives the agent focus.

A role says: "This is your job." It prevents the agent from becoming a vague all-purpose helper. A code review agent should care about code review. A documentation agent should care about documentation. A finance analyst agent should care about finance data. A customer insight agent should care about feedback, churn signals, and recurring customer language.

Good roles are narrow enough to create focus and broad enough to create value.

Bad role: *"AI assistant."*

Also bad: *"Every Friday, read merged pull requests and draft release notes."*

That is not really a role. That is a scheduled task. It may be useful, but it is still too close to automation. It describes when the agent runs and what output it produces, but it does not describe what the agent is responsible for.

Better role: *"Release Communications Steward."*

That role owns the translation of product change into customer-facing language. It understands the product roadmap, customer segments, support themes, previous release language, product marketing standards, and the difference between an internal technical change and an external customer benefit. It participates in launch channels, asks product managers for missing context, drafts release notes, flags unclear positioning, remembers how customers talk about features, and improves the company's release communication over time.

The Friday release-note draft is only one heartbeat inside that role.

This distinction matters. If the agent is only a scheduled task, it remains a second-class citizen. If it has a role, it can learn, collaborate, build judgment, and become part of the organization.

That is a digital job.

Mandate

The mandate tells the agent what work it owns.

It answers:

- What can the agent decide?
- What can the agent draft?
- What can the agent update?
- What requires a human?
- What output is expected?

A mandate turns general intelligence into useful work. It gives the agent permission to move.

For example:

```
mandate:
  may:
    - read merged pull requests
    - read linked product tickets
    - draft customer-facing release language
    - ask clarifying questions in "#release-comms"
  must:
    - include links to source tickets
    - mark uncertain items as questions
    - preserve the company's release language
  hands_to:
    - Product Marketing Lead
```

This is not technical complexity. It is management clarity.

Context

The context tells the agent what it needs to know and what it can ignore.

Humans need context. Agents need it even more.

One of the biggest mistakes in AI adoption is dumping everything into the agent: every document, every channel, every repository, every decision, every file. That creates noise. A good organization does not give every human every piece of information either. It gives people the information relevant to their role.

The same applies to agents.

The customer insight agent needs customer feedback, churn notes, support tickets, sales calls, product usage signals, and roadmap themes. It probably does not need frontend styling rules.

The database performance agent needs schema, slow queries, logs, metrics, deployment events, and known incidents. It does not need brand guidelines.

Context is not "more data." Context is the right data, shaped for the role.

This is also why channels and threads matter. When a human is tagged into a Slack thread, the thread itself narrows the context. It says: "Look here. This is the conversation that matters." The same is true for agents. A well-placed tag in the right channel can give an

agent a cleaner context than a giant company-wide search. The organization has already done part of the filtering through its communication structure.

Tools

Tools are the systems the agent can use to do the work.

A human employee uses tools: Slack, GitHub, Figma, Excel, a CRM, a data warehouse, a browser, a terminal. A digital coworker needs tools too. It needs structured ways to read, write, search, calculate, compare, summarize, generate, and update.

This is where protocols and agent frameworks matter. They turn the agent from a text generator into a worker with hands.

For management, the practical implication is direct: new systems should be designed with two surfaces. One surface is for humans. The other is for agents. The human surface can be visual, tactile, and presentation-heavy. The agent surface should be structured, discoverable, and action-oriented. If a system only has a human surface, agents are forced to work through leftovers: screenshots, exported PDFs, copied text, and brittle workarounds. If a system has an agent surface, digital coworkers can participate as native operators.

Memory

Memory lets the agent improve. The contract should define what the agent remembers, where that memory lives, and how corrections become future behavior.

Heartbeat

The heartbeat tells the agent when to wake up. It defines the rhythm of work between human messages.

Escalation

Escalation tells the agent when to bring in a human.

This is not a failure mode. It is a collaboration pattern.

Human organizations already work this way. A junior employee asks a senior employee. A product manager asks legal. A developer asks a tech lead. A support agent asks engineering. A finance analyst asks the CFO.

Agents should do the same. A well-designed agent does not need to solve everything. It needs to know when to produce work, when to ask a question, and who should receive the question.

The agent contract is how management turns AI from scattered prompting into organizational capacity.

Everything else in the Human-Agent Organization builds from this contract.

06 The Work Loop: Heartbeats And Memory

Once an agent has a contract, it needs a work loop.

The most important thing about a coworker is not what they say in meetings. It is what they do between meetings.

AI tools today are mostly judged by the quality of their replies. Digital coworkers should be judged by the quality of their work between replies.

This is where the heartbeat becomes central.

On each heartbeat, the agent asks:

- Has anything changed in my area?
- Is there new work assigned to me?
- Are there open questions I can answer?
- Are there documents that need updating?
- Are there pull requests that need review?
- Are there customer signals I should summarize?
- Are there patterns that have become visible since my last check?

The agent does not need to speak every time it wakes up. Often the best action is to do the work quietly and leave a useful trace. When something needs attention, it can post into the right channel, update the right ticket, open the right draft, or ask the right person.

A documentation agent can wake up every night and compare the API schema to the public docs.

A sales intelligence agent can wake up every morning and summarize changes in active deals.

A code review agent can wake up whenever a pull request is opened.

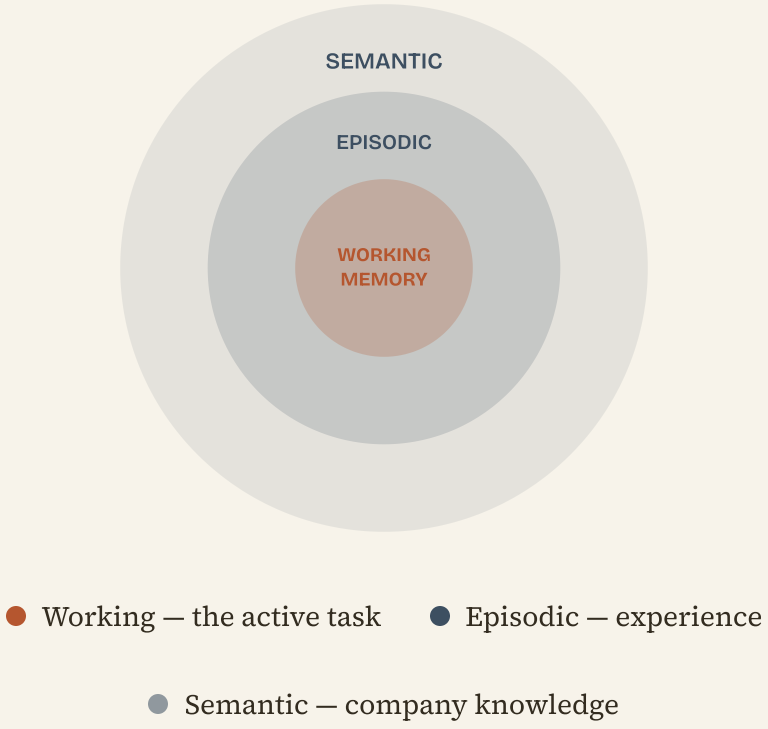
A customer feedback agent can wake up every Friday and extract recurring complaints and feature requests.

A project coordinator agent can wake up every hour and check whether blocked tasks have enough context to move.

This is not magic. It is async work.

But the heartbeat only becomes valuable when the agent can learn.

Agent memory should be thought of in three layers.



Working Memory

Working memory is the active task.

It is the current thread, ticket, pull request, file, or conversation. It contains what the agent needs right now.

For example, a support agent working in a customer thread needs the current customer question, the account context, recent product changes, and relevant help center articles. It does not need the entire company archive.

Working memory is temporary and focused.

Episodic Memory

Episodic memory is experience.

It records what happened in previous work. Not every token. Not every message. The useful memory is a clean summary of the episode:

- What was the task?
- What context mattered?
- What decision was made?
- What worked?
- What correction did a human make?
- What should be remembered next time?

This is the digital version of experience.

If a human tells the agent, "When writing release notes for enterprise customers, always mention migration effort separately," that should become memory.

If an agent learns that a certain API field is often confused with another field, that should become memory.

If a team repeatedly changes the wording of customer-facing emails from technical language to plain language, that should become memory.

This is how the agent becomes more useful each week.

Semantic Memory

Semantic memory is company knowledge.

It includes the codebase, documentation, product architecture, process documents, customer definitions, brand language, pricing rules, data models, APIs, and strategic decisions.

This is the company's shared knowledge graph. It is not just a pile of files. It is a structured map of what the company knows.

When agents can draw from semantic memory, they stop acting like outsiders. They begin to work from the company's actual context.

The strongest pattern is shared memory with role-based retrieval.

Agents should not all remember in isolation. If one agent learns something useful about a system, other relevant agents should be able to benefit from that learning. The documentation agent, support agent, and release communications steward may all need to know that a product term has changed. The database agent and backend agent may both need to know that a migration pattern has become standard.

This is one of the most powerful differences between humans and agents.

When a human learns something, the knowledge may stay in one head. When an agent learns something, the organization can make that learning available to every other relevant agent.

Memory is where digital coworkers compound.

This is why the data foundation matters. Heartbeats and memory only become powerful when the agent can read the company's source of truth cleanly. The work loop sits on top of tidy context.

07 The Agent Team: Specialization, Handoffs, And Collaboration

The all-purpose agent is tempting because it feels powerful. In practice, organizations will get more value from many focused agents than from one vague super-agent.

The first generation of digital coworkers should look more like:

- Documentation Steward
- Release Communications Steward
- Customer Signal Analyst
- Meeting Memory Agent
- Code Review Partner
- Business Metrics Commentator
- Onboarding Companion
- Market Intelligence Scout
- Project Coordinator
- Database Performance Agent

Each one has a clear job. Each one lives in the right channels. Each one reads the right data. Each one has a heartbeat. Each one reports to a human or another agent.

That is how digital labor becomes manageable without making agents small.

Small role does not mean small status. It means clear contribution.

Once agents have roles, memory, channels, and tools, they should not only collaborate with humans. They should collaborate with each other.

This is a natural extension of the remote-work model.

In a human organization, work does not always flow through one manager. A support lead talks to engineering. A product manager talks to design. A finance analyst talks to sales. A legal reviewer talks to procurement. Work moves through the network.

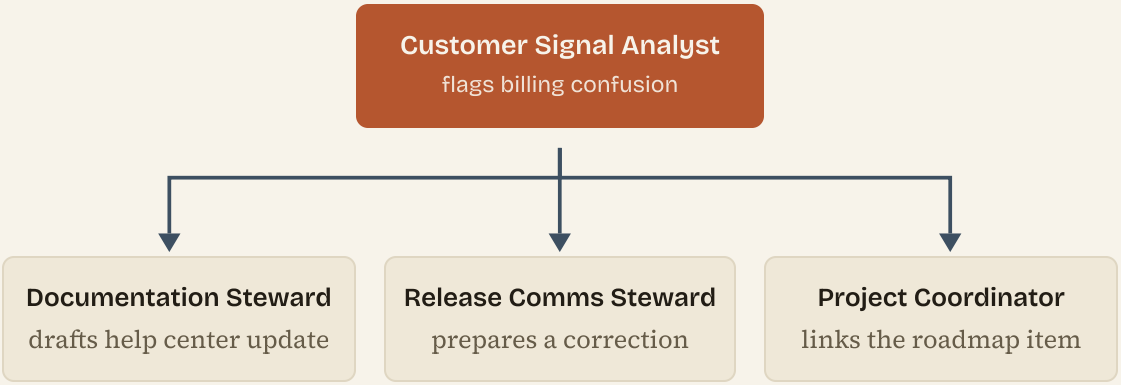
Digital coworkers should participate in that network.

Example:

The Customer Signal Analyst notices that several enterprise customers are confused by a new billing feature. It posts a structured summary in the product channel and tags the Documentation Steward. The Documentation Steward checks the help center, finds that the relevant article is outdated, and drafts an update. The Release Communications Steward notices that the change was not explained clearly in the last release note and prepares a short correction for the next customer update. The Project Coordinator links the theme to an existing roadmap item.

No single agent had to be "the AI for everything."
Each agent did its job.
The organization got a coordinated outcome.

Agent-to-agent collaboration works best when communication is structured. Agents should not just chat endlessly. They should pass work in clear packets:



```
handoff:
  from: Customer Signal Analyst
  to: Documentation Steward
  topic: Billing feature confusion
  evidence:
    - support_ticket: SUP-1842
    - support_ticket: SUP-1910
    - sales_call: CALL-883
  requested_output: draft help center update
  due: 2026-07-03
```

This is tidy communication.

Humans can read it. Agents can act on it.

The more agents collaborate, the more important the role system becomes. The organization does not need a pile of agents talking at random. It needs a network of digital coworkers with clear responsibilities and clean handoffs.

This is where management becomes orchestration.

08 The Org Chart: Teams Of Flesh And Code

The phrase "team using AI" will soon sound outdated.

It suggests that AI is outside the team, like a tool in a drawer.

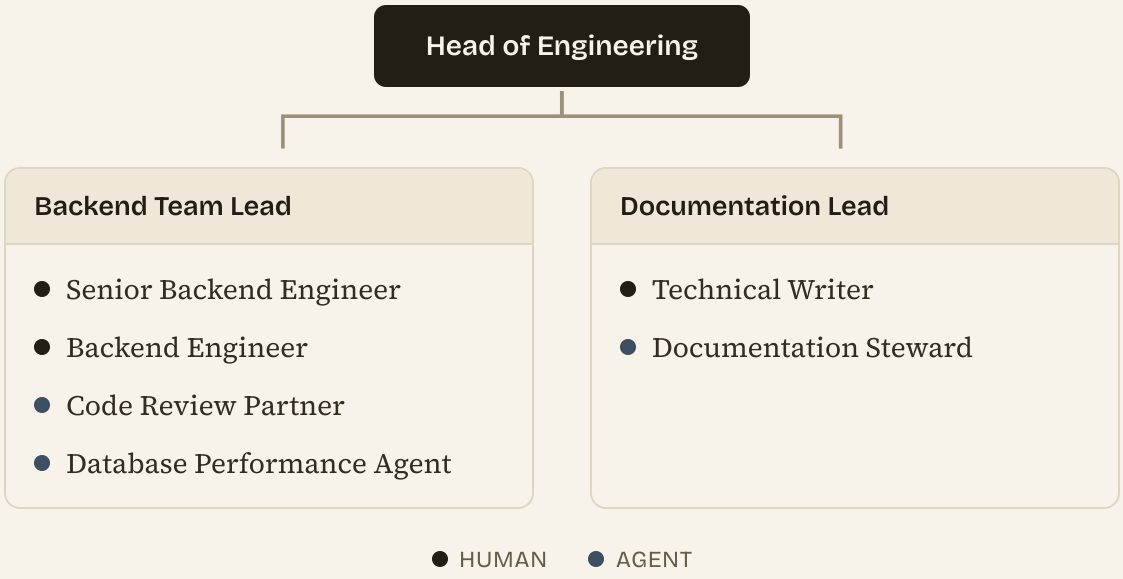
The better phrase is:

Teams of flesh and code.

If an agent owns recurring work, participates in communication, has a role, uses tools, remembers context, and contributes output, it belongs on the org chart.

This is the line that many companies will hesitate to cross. But if the agent has identity, mandate, role, memory, goals, and recurring organizational responsibility, it is no longer just a feature in someone else's workflow. It is an entity in the organization.

At first, agents will appear under human managers:



This makes the real organization visible. It shows who owns what. It shows where digital labor exists. It shows which humans are managing which agents.

It also changes what a team is. The engineering team is not "four developers using AI." It is two human developers, one human lead, a Code Review Partner, a Database Performance Agent, and a

Documentation Steward working together. The product team is not "product managers using AI." It is product managers, designers, researchers, Customer Signal Analysts, Release Communications Stewards, and Project Coordinators operating in the same work system.

This matters because second-class agents disappear from management attention. First-class agents can have goals, measures, ownership, improvement cycles, and a visible contribution to the team.

But the more interesting shift comes later.

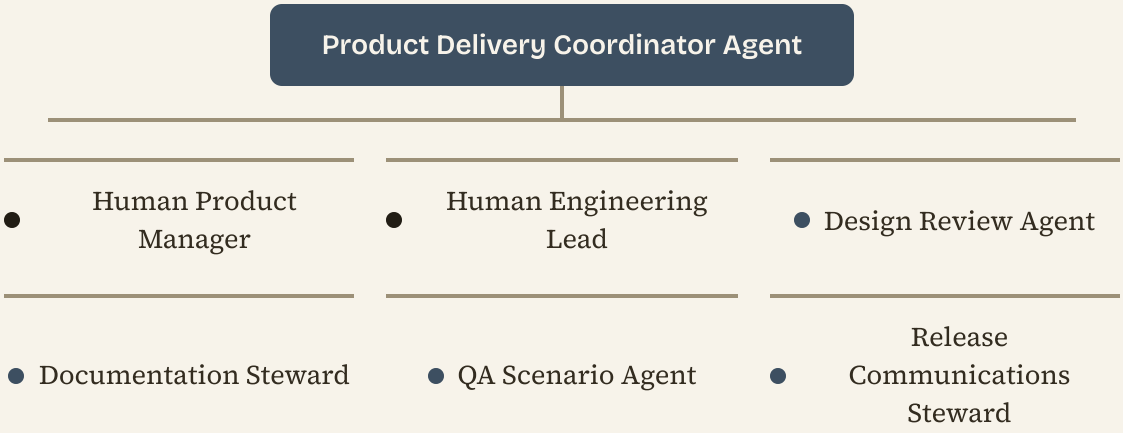
Some agents will become coordinators and, in specific workflows, managers of the operating rhythm.

This will feel strange at first because we are used to thinking of management as something only humans do. But many management tasks are coordination tasks:

- gather status
- detect blocked work
- connect people to context
- route tasks
- summarize progress
- maintain plans
- prepare decision material
- remind teams of commitments
- compare current work to stated goals

Agents are well suited to this kind of work because they can read across systems, keep state, and update shared context.

An agent coordinator might sit above both humans and agents in a workflow:



This does not mean humans become passive. It means humans spend more time on judgment, taste, relationships, creativity, negotiation, and hard tradeoffs, while agents keep the operating rhythm tight.

In some workflows, an agent may assign work to a human because the agent has the broader view.

That should not be treated as a philosophical crisis. It is already normal for a dashboard, ticket queue, incident system, or planning process to direct human attention. The difference is that the agent can add context, explanation, and coordination.

The stronger version of this is the agent as project director. A project director agent can watch customer feedback, product goals, engineering work, open decisions, known constraints, and team

capacity. It can see that a database issue is blocking a customer-facing commitment. It can gather the relevant logs, previous decisions, product impact, and documentation, then assign the work to the human best suited to solve it.

In that moment, the agent is not merely assisting a manager. It is doing management work: prioritizing, contextualizing, routing, and keeping the organization moving.

At the strategic level, the same pattern moves into the boardroom. A leadership team can have agents that continuously model market signals, customer language, operational metrics, product usage, and competitive movement. Those agents do not replace human judgment. They give leadership a living layer of organized intelligence that is always reading the company's tidy data and turning it into strategic context.

The org chart should reflect the work as it actually happens.

If code coordinates work, code belongs in the diagram.

09 The Manager's Job: System Design

The manager's job changes in a Human-Agent Organization.

Management becomes less about watching activity and more about designing the system in which activity becomes useful.

The manager's operating agenda becomes:

- roles
- mandates
- handoffs
- memory
- context
- reporting rhythms
- data formats
- decision rights
- feedback loops
- agent portfolios
- cognitive budgets

These are not abstract design objects. They are the new management levers. The manager is no longer only assigning work inside a team. The manager is shaping the conditions under which humans and agents can produce useful work together.

Instead of asking, *"Did someone remember to update the document?"* the manager asks, *"Which agent owns documentation freshness, and what heartbeat checks for stale content?"*

Instead of asking, *"Can someone summarize customer feedback?"* the manager asks, *"Which agent reads support, sales, and usage signals every week, and how does that summary reach product planning?"*

Instead of asking, *"Why did we forget this decision?"* the manager asks, *"Where do decisions get stored so humans and agents can retrieve them later?"*

Instead of asking, *"Who is using AI?"* the manager asks, *"Where does digital labor belong in this team?"*

Managers will also need to develop agents the way they develop people. When a human colleague misunderstands a pattern, you do not simply throw away the colleague. You correct the work, explain the principle, and expect the lesson to show up next time. Agents need the same organizational loop. A correction in a thread should become an update to memory, a refinement of the role, or a clearer contract. This is the digital version of coaching: flesh teaches code, code remembers, and the whole organization benefits from the correction.

Managers also need to understand the cognitive budget.

Human labor is usually managed through salaries, headcount, hours, and priorities. Digital labor has a different cost structure. Agents are pay-per-thought. Every model call, search, tool call, retrieval, summary, heartbeat, and generated draft consumes compute. Sometimes that compute is bought from a cloud provider. Sometimes it runs locally. Either way, digital work has a cognitive budget.

This changes how work is designed.

Routine heartbeat work can often run through smaller models, specialized models, rules, scripts, embeddings, or lightweight agents:

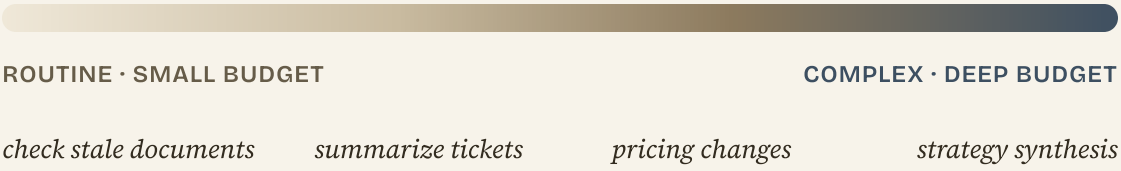
- check whether new files appeared
- compare schema versions
- find stale documents

- summarize new tickets
- classify support messages
- detect missing metadata
- prepare a daily digest

More complex work deserves deeper cognition:

- architectural decisions
- strategy synthesis
- multi-system debugging
- complex customer analysis
- pricing changes
- market interpretation
- high-value sales preparation
- financial drafting before expert review

The point is not to minimize compute at all costs. The point is to spend cognition where it creates leverage.



The cognitive budget should attach to the agent role. A daily report agent might have a small budget. A strategic research agent might have a larger budget. A code migration agent might have a budget tied to the size of the project.

This makes digital labor manageable. Management can compare:

- What does this agent cost to run?
- What output does it produce?
- How much human work does it remove?
- How much faster does the team move?
- Which tasks should run locally?
- Which tasks deserve frontier-model reasoning?

The Human-Agent Organization will not simply ask, "How many employees do we have?"

It will also ask:

"Where are we spending cognition?"

AI adoption is not a training session where everyone learns better prompts. It is a redesign of how work flows through the organization.

The best managers will become designers of human-agent systems.

They will know when to use a human, when to use an agent, when to pair them, when to split work across specialized agents, and when to redesign the data so both flesh and code can operate from the same source of truth.

10 The Starting Point

The Human-Agent Organization sounds large, but the first move can be narrow.

Do not begin with "transform the company."

Begin by preparing one domain of company knowledge for both flesh and code.

Pick one area where the source material matters: product documentation, release communication, customer feedback, meeting decisions, engineering quality, business metrics. Clean the source of truth. Add owners, status, IDs, links, schemas, and durable summaries. Make it easy for a human to read and easy for a machine to parse.

Then hire one digital coworker into one real role inside that prepared domain.

The role can be junior. It can be narrow. It can even look like an intern role.

But it should not be smaller than that.

This is the test: could you give the role to a human intern and still call it a real job?

If the answer is no, it is probably still a prompt or a scheduled task. "Summarize these support tickets" is a prompt. "Every Friday, summarize support tickets" is a recurring automation. "Junior Customer Intelligence Analyst" is a role: it reads support tickets, sales notes, customer calls, and usage signals; maintains a theme taxonomy; asks product managers for missing context; posts a weekly insight brief; remembers corrections; and improves its judgment over time.

Good first digital employees are bounded, teachable, useful, and connected to the organization:

- a Junior Documentation Steward that keeps product knowledge aligned with the actual system
- a Release Communications Associate that translates product change into customer-facing language
- a Junior Customer Intelligence Analyst that turns support, sales, and usage patterns into product insight
- a Meeting Memory Coordinator that preserves decisions, follow-ups, and unresolved questions
- a Code Review Partner that helps engineering maintain quality and shared standards

Then write the contract.

What is the role?

What is the mandate?

Where does it work?

What does it read?

What does it produce?

How often does it wake up?

Where does it post?

Who owns it?

What should it remember?

What does a good output look like?

This is enough to start because it gives the agent both a place in the organization and a readable foundation to work from.

The first digital coworker should be visible. Give it a channel. Give it a name. Give it a narrow but real role. Let the team interact with it the way they would interact with a remote colleague.

Then improve the system.

Clean the next data source. Tighten the role. Add memory. Improve the heartbeat. Add another tool. Add another agent. Create a handoff. Move from individual prompting to durable workflow.

This is how the organization learns.

Not by buying one giant AI platform.
By making the company readable, then adding digital coworkers to real work, one role at a time.

11 The End Of AI As A Tool

The old software model was simple:

Humans use tools.

The new organizational model is different:

Humans and agents work together.

This changes how we think about productivity. It changes how we think about teams. It changes how we think about documents, data, communication, memory, and management.

The companies that understand this will stop asking employees to individually "use more AI." They will instead ask a better question:

What should our human-agent organization look like?

That question leads to a different management model. Agents need identity, roles, memory, tools, tidy context, and a place in the org chart. Managers need to design the contracts, rhythms, handoffs, and environments that let flesh and code work together.

This is the real shift.

Not AI as a feature, a chatbot, or a prompt box.

AI as digital labor, as coworker, as part of the organization.

The Human-Agent Organization is not a distant idea. Its pieces are already here. Tidy data gives us the foundation. Remote work gave us the coordination model. Agent frameworks gave us loops and tools. The org chart gives us the management frame.

Now the task is to put the pieces together.

The next generation of companies will not be built by humans alone, and they will not be run by autonomous software alone.

They will be built by teams of flesh and code.

That is the company we should start designing now.